ESCOLA NAVAL

talant de bien faire

Gonçalo Daniel Castanheira Rosa

Validação Computacional tendo em vista a Interoperabilidade entre os Veículos Aéreos Não Tripulados

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenharia Naval, Ramo Armas e Eletrónica

Alfeite
2016

Gonçalo Daniel Castanheira Rosa

Validação Computacional tendo em vista a Interoperabilidade entre os Veículos Aéreos Não Tripulados

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, com especialização em Engenharia Naval, Ramo Armas e Eletrónica

Orientação de: Mário Rui Monteiro Marques

O Aluno Mestrando — O Orientador

_____ — _____

[Gonçalo Rosa] — [Mário Marques]

Alfeite

2016

Epigrafe

"Quando planeamos a nossa atividade, é fundamental ter a noção de que 20% daquilo que fazemos é responsável por 80% dos nossos resultados. Começar pelos 20% mais importantes é a melhor decisão, ainda que, normalmente, estas sejam as tarefas mais difíceis e que muitas vezes menos nos apetece iniciar."

- Paulo de Vilhena

Dedicatória

Dedico este trabalho a todos aqueles que acreditaram e me apoiaram durante toda a minha vida.

Agradecimentos

A elaboração deste projeto teve a duração de um ano letivo e teve a intervenção de várias pessoas, cada um com o seu contributo. De entre as quais gostava de demonstrar o meu agradecimento:

Ao meu orientador CTEN EN-AEL Mário Rui Monteiro Marques por me ter acompanhado e ajudado a realizar esta dissertação.

Ao Doutor Vitor Lobo que, como diretor do CINAV, proporcionou a elaboração desta dissertação.

Aos Srs. Engenheiros Nuno Simões e Marco Simões, não só pela preocupação ao longo deste ano, mas também pelo acompanhamento, ajuda e disponibilidade demonstrados na realização desta dissertação.

Ao Departamento de Formação de Engenharia Naval de Armas e Eletrónica da Escola Naval, por todo o apoio prestado.

À minha família e amigos que me apoiaram durante este período em que estive envolvido na realização desta dissertação de mestrado, e compreenderam os momentos em que não estive presente.

A todos aqueles que ao longo destes anos me apoiaram e tornaram esta dissertação possível. Aqui fica um sentido muito obrigado.

Resumo

Nos dias de hoje, com o contínuo desenvolvimento e inovação no campo dos UAVs (*Unmanned Aerial Vehciles*), o mundo já tem como adquiridos os benefícios que estes sistemas podem fornecer. Os benefícios obtidos com a aplicação destes sistemas abrange tanto as forças armadas como industrias e organizações civis.

Todas as nações e indústrias querem ter uma cota parte no futuro desta tecnologia. Diferentes UAVs foram desenvolvidos, mas estes, diferem em termos de arquitetura e protocolos de comunicação. Protocolos como o STANAG 4586, MAVLink, JAUS e ROS são só alguns exemplos.

A proliferação de informação através destes sistemas e as suas consolas de comando e controlo é uma das principais preocupações, principalmente pelas forças armadas. Uma das principais prioridades é combinar forças de diferentes nações, principalmente pelos membros NATO. A necessidade de uma consola para cada tipo de sistema devido à falta de padronização apresenta assim um problema.

É conhecida a necessidade de uma padronização em termos de arquitetura por camadas e de comunicação tendo em vista a interoperabilidade entre estes sistemas. Não existe nenhuma que esteja a ser implementada como documento padrão. Pretende-se que o STANAG 4586 seja o documento padrão para os membros NATO e, por conseguinte, todos os esforços estão direcionados em desenvolver sistemas que o consigam implementar.

Os diferentes UAVs já existentes possuem o seu próprio protocolo de comunicação e a alteração de toda a sua estrutura não é fácil. A ideia de fazer uma conversão de linguagens como alternativa surge como uma solução teórica ótima.

Utilizando um piloto automático que comunica com a sua consola através da linguagem MAVLink esta dissertação tem como objetivo desenvolver um programa computacional que converta as mensagens MAVLink em STANAG 4586 e estudar se o tempo de conversão é operacionalmente válido tendo em conta os requisitos operacionais dos sistemas.

Palavras-chave: UAV; interoperabilidade; MAVLink; STANAG 4586; Análise Temporal.

Abstract

Nowadays, in the continuous technological development and innovation regarding UAVs (*Unmanned Aerial Vehciles*), the world has acknowledged the benefits that these systems bring to our environment. The profits received with the application of these robots cover almost all the fields regarding the armed forces, environmental and agriculture industries and civil protection organizations.

Every nation and industry wants to take part in this future main technology. Different UAVs have been designed and developed that differ in terms of architecture and communication protocols. Frameworks like STANAG 4586, MAVLink , JAUS and ROS are some examples.

The proliferation of information through these systems and their command and control consoles is one of the main concerns, mainly by armed forces. Combining forces from different nations, mainly by NATO members, in exercises and real time crisis fights are one of the primary priorities due to the benefits that they can combine.

It's known the necessity of a standard in terms of layered architecture and communication towards the interoperability between these systems. Many standards have been created trying to fulfil this gap but there isn't one that is currently implemented as the main standard document. The STANAG 4586 is intended to be implemented as the main standard for NATO members and, therefore, all the efforts go towards to develop systems that implement these architecture and communication protocol.

The different UAVs already created have their one communication protocol and the redesign of the entire architecture of the systems by one company isn't easy and could not be convenient or affordable. The idea of doing a conversion of languages instead of redesign all the system architecture emerges as the optimal theoretical solution.

Using a PIXHAWK autopilot that communicates to the Ground Control Station (GCS) through MAVLink it's possible to develop computational software that converts MAVLink messages to the format of STANAG 4586. Starting from there, the objective is to check if it is viable for the protocol requirements and study the delay that the

conversion introduces to the system in comparison with the channel without the conversion. The delay must not be inviable for the communication requirements between the GCS and the UAV.

Key Words: UAV; Interoperability; MAVLink; STANAG 4586; Temporal Analysis.

Índice

Índice de figuras

Índice de tabelas

Índice de gráficos

Lista de Acrónimos, Abreviaturas e Siglas

3D – Dull, Dirty and Dangerous

ADCP – Acoustic Doppler Current Profiler

ASCII – American Standard Code for Information Interchange

BAM – Binary Angular Measurement

BLOS – Beyond Line Of Sight

C4I – Command, Control, Communication, Computers and Intelligence

CCI – Command and Control Interface

CCISM – Command and Control Interface Specific Module

CDT – Control Data Terminal

CEMA – Chefe de Estado-maior da Armada

CIS – Computerized Information System

COE – Common Operating Environment

CONOPS – Concept Of Operations

COTS – Commercial Of-The-Shelf

CPU – Central Processing Unit

CRC – Cyclic Redundancy Check

CRD – Common Route Definition

CUCS – Core UAV Control Systems

DLI – Data Link Interface

DVL – Doppler Velocity Log

ET – Exploratory Team

EUA – Estados Unidos da América

GCS – Ground Control Station

GPS – Global position System

GSM – Global System for Mobile Communications

I&R – Intelligence and Reconnaissance

IA – Influence Activities

ID – Identification

IDD – Interface Definition Document

IEC – International Electronic Commission

IEEE – Institute of Electrical and Electronics Engineers

ISO – International Standard Organization

ISR – Institute for Systems and Robotics

JAUS – Joint Architecture for Unmanned Systems

JCGUAV – Joint Capability Group on UAVs

LIDAR – Light Detection and Ranging

LOI – Level Of Interoperability

LOS – Line Of Sight

MAVLink – Micro Aerial Vehicle Link

MIO – Maritime Interdiction Operations

MIS – Management Information System

MPDES – Mission Planning and Data Exploitation Stations

MSB – Most Significant Bit/Byte

NATO – North Atlantic Treat Organization

NIAG – NATO Industrial and Advisory Group

NIIA – NATO ISR Interoperability Architecture

NISP – NATO Interoperability Standards and Profiles

NNEC – NATO Network Enabled Capability

OMG – Object Management Group

OSI – Open Systems Interconnection

R&A – Robótica e Automação

RADAR – Radio Detection and Ranging

RAS – Robotic and Automation Systems

RF – Radio Frequency

ROS – Robot Operating System

SAR – Search and Rescue

SAR – Synthetic Aperture Radar

SCI – System Concepts and Integration

SNI – Sistema de Navegação Inercial

STANAG – Standardization Agreement

TCP/IP – Transmission Control Protocol / Internet Protocol

UART – Universal Asynchronous Receiver/Transmitter

UAS – Unmanned Aerial System

UAV – Unmanned Aerial Vehicle

UCS – UAV Control System

UDP – User Datagram Protocol

UMI – Unidade de Medida Inercial

USB – Universal Serial Bus

UTC – Universal Time Coordinated

UTM – Universal Transverse Mercator

UxS – Unmanned Systems of all Surfaces

UxV – Unmanned Vehicles of all Surfaces

VDT – Vehicle Data Terminal

VSI – Vertical Speed Indicator

VSM – Vehicle Specific Module

WGS – World Geodetic System

XML – eXtensible Markup Language

Introdução

Sistemas não tripulados são plataformas que não são controladas por um piloto a bordo da mesma. Cada robot/plataforma é criada com um objetivo principal sendo que, será possível diversificar com a utilização e introdução de equipamentos e sensores que não faziam parte do sistema original mas que serão necessários para as várias tipologias de missões (Dudek, 1993). Este tipo de sistemas que opera pelo ar é denominado Sistema Aéreo Não Tripulado (UAV – *Unmanned Aerial vehicle*). Estes mesmos veículos têm ganho uma importância significativa nos últimos anos devido à sua eficácia e eficiência operacional substituindo as aeronaves tripuladas, tornando-se assim parte relevante no corrente mundo da aviação.

A razão pela qual eu escolhi abordar este tema na minha dissertação de mestrado deve-se ao facto de que este tipo de tecnologia, hoje em dia, ser uma tecnologia com avanços contínuos, alienando o facto de estar a crescer exponencialmente em termos de visualização mundial na comunicação social bem como a perspetiva futura de ter uma crescente utilização em várias áreas, no dia-a-dia, de todo o mundo (Cai, 2010, Manley, 2008).

A introdução destes sistemas na Marinha de Guerra Portuguesa é uma das suas principais prioridades devido aos benefícios e complementaridade que este tipo de tecnologia pode fornecer nos diversos cenários e missões operacionais. É importante fazer referência que, para além da sua capacidade óbvia de expansão do raio de ação dos meios da Marinha, a utilização de UAVs irá permitir uma substancial redução dos custos operacionais e de prevenção da vida humana (Manley, 2008, Fletcher, 2000, Rosa, 2016).

As plataformas dos veículos autónomos não tripulados são, maioritariamente, de pequena ou média dimensão, automatizadas de modo a serem semi controladas remotamente (por meio de uma *Ground Control Station - GCS*) ou completamente autónomas sendo transmitido um objetivo que irá ser alcançado autonomamente (Cai,

2010, Watts, 2012). Estes mesmos sistemas, quando totalmente autónomos, devem incorporar todo um conjunto de subsistemas substanciais e complexos que incorporam tanto a área de sensores como de controlo (Fletcher, 2000, Watts, 2012).

Missões como a entrega de material médico em ambientes hostis ou em locais em que o ambiente não seja favorável em termos de condições meteorológicas são exemplos onde este tipo de tecnologia pode ser usado e benéfico. Balanceando o risco quase inexistente para o operador da plataforma bem como o baixo custo associado, torna exequível a utilização deste equipamento em qualquer missão, em qualquer lugar do mundo (Dixon, 2005, Rosa, 2016). São evidentes as diferenças quando comparada uma destas plataformas com um helicóptero. Se usarmos como exemplo o helicóptero Super Lynx que é ocupado por dois pilotos e um operador necessários na sua operação e tem um custo de aquisição de 13 milhões de euros (sem contemplar o custo do combustível) com um Phantom 3 (UAV utilizado na entrega de mercadorias) que pode ser adquirido por 727.5 euros e não requer nenhuma pessoa a bordo para ser operado, as diferenças são imensas. Se tivermos em conta as consequências caso ocorra um desastre a um helicóptero de 13 milhões de euros com três pessoas a bordo, estas são obviamente gigantescas em comparação com um Veiculo Aéreo Não Tripulado de 727.5 euros, conforme pode ser verificado na Tabela 1.

Tabela 1: Comparação entre um helicóptero Lynx com um UAV Phantom

Material	Número de Pessoas a bordo	Preço (€)	Perdas em caso de acidente (€)
Helicopter Super Lynx	3	13M	13M + custo do treino + 3 pessoas
Phantom 3	0	727.5	727.5

UAVs, como robots que são, estão capacitados para serem adaptados a vários tipos de missões dependendo da tipologia de sensores ou comunicação com que estão equipados. Os robots são máquinas e um dos benefícios que apresentam é que, como tal, não requerem treino. Se tomarmos como exemplo uma missão de espionagem num ambiente hostil, a diferença entre perder um militar, ou seja, um humano com vários anos de educação e treino ou um UAV que pode recolher a mesma informação para depois ser analisada por peritos é abismal. É passível de reconhecer neste exemplo a diferença entre existirem baixas humanas ou perdas materiais, assim sendo, os valores morais e éticos que esta questão levanta não podem ser negligenciados (Rosa, 2016).

O desenvolvimento nesta área tecnológica é assim de elevada importância devido à mínima necessidade logística que é requerida, de modo a que um humano possa operar com este tipo de sistemas de cariz robótico (Cai, 2010). Sistemas que não comem reduzem os custos logísticos. Não requerem descanso ou tempo para repor energias e por essas razões podem estar operacionais e serem utilizados 24h por dia (Bertram, 2008). Restrições ambientais como o calor ou o frio não se apresentam como problemas para estes veículos pois não apresentam restrições, dentro do razoável, e não necessitam de apoio logístico em termos de roupa, por exemplo. Quando um veículo consegue alcançar sucesso numa missão está implícito que um outro veículo semelhante irá alcançar o mesmo resultado e por isso não requer treino. A introdução de *software* e atualizações do mesmo são de fácil implementação e reduzem consideravelmente o tempo de preparação para cada diferente tipo de missão (Manley, 2008).

Com a crescente popularidade destes sistemas em todo o mundo, é expectável tanto um aumento de procura como de desenvolvimento. Com o aumento da procura prevê-se também um aumento da quantidade de utilizadores a querer prosperar nesta nova indústria e a contínua procura de novos mercados, o que permite um desenvolvimento exponencial da mesma. Logística, *software*, sensores e comunicação são algumas das áreas que vão sofrer um maior desenvolvimento e consequentemente

um maior "*stock*" tal como uma diminuição do seu preço (DeGarmo, 2004). Emerge assim a necessidade de ser criado e adotado um método comum de comunicação entre estes sistemas de modo a que, sistemas de países e forças diferentes possam cooperar e colaborar entre eles.

A interoperabilidade é um tema bastante estudado na atualidade, especialmente por vários grupos NATO *(North Atlantic Treaty Organization)* como: *NATO ET012 (Exploratory Team) Affordable Robotics for Military Operations; SCI (System Concepts and integration) -ET-009 Command and Reporting Standards and Development Tools for UxS; NIAG (NATO Industrial and Advisory Group) 202 Study on development of conceptual data model for multi-domain unmanned platform control system*. Existe também na Marinha Portuguesa um grupo de trabalho focado no desenvolvimento da capacidade de operação dos veículos não tripulados (GT-VENT). Este grupo foi criado por Despacho do Almirante CEMA (Chefe de Estado-Maior da Armada) nº 06/15 de 12 de Fevereiro[1], cumprindo com a Diretiva de política Naval que prevê a capacidade da Marinha de operar e manter veículos autónomos não tripulados tanto aéreos como terrestres.

A interoperabilidade entre sistemas não tripulados é uma tarefa de difícil implementação devido ao facto de que, desde a criação destes sistemas, várias linguagens de comunicação entre UAVs e GCS foram desenvolvidas com o objetivo de serem implementadas nos veículos com os mais diferentes tipos de missões tendo sempre em mira aproveitar os benefícios que cada uma pode oferecer em cada caso. Algumas das principais linguagens são: STANAG (*Standardization Agreement*) 4586; JAUS (*Joint Architecture for Unmanned Systems*); MAVLink (*Micro Aerial Vehicle Communication Control*); ROS (*Robot Operating System*). Cada uma com a sua orientação e benefícios associados devido ao tipo de objetivo para que foi originalmente estruturada e desenvolvida.

Estruturas de comunicação padrão facilitam o *design*, manufatura, produção, transferência tecnológica, compra e o uso destes robots. São também um meio para

[1] Processo disponível em anexo C.

obter um quadro de desenvolvimento jurídico comum. As atividades de desenvolvimento de um *standard* deste meio estão associadas à indústria, investigação, utilizadores e autoridades pois os seus interesses e necessidades em comum irão ser suplantados pelo desenvolvimento do mesmo (Rowe, 2008).

A padronização é essencial no desenvolvimento de sistemas de Automação e Robótica (R&A). A falta de orientação e padronização em robótica causa, sem sombra de dúvidas, um desenvolvimento mais lento e ainda divergência a esse nível, causando frustração não só para os utilizadores ou consumidores como para os fabricantes. A padronização deste meio facilita tanto a comercialização como a transferência de conhecimentos e ainda pode ser usado como meio de orientação para um objetivo comum por parte das atividades de pesquisa e desenvolvimento (Madhavan, 2009).

A Organização Internacional de Padronização (ISO) foi uma das pioneiras na criação de *standards* robóticos, tendo lançado o seu primeiro *standard* para a indústria robótica em 1980. Além desta, a Comissão Internacional Eletrotécnica (IEC), a *Object Management Group* (OMG) e outras organizações começaram também a desenvolver estruturas padrão para este tipo de serviço robótico emergente. De acordo com o *Institute of Electrical and Electronics Engineers* (IEEE) *Robotic and Automation Society* (RAS) *Standing Committee for Standards* os objetivos para a padronização incluem:

- Medidas e definições comuns em R&A;
- Mensurabilidade e comparabilidade de tecnologia R&A;
- Integrabilidade, portabilidade e reutilização.

Refere também que a padronização do desenvolvimento R&A deverá ser orientada a:

- Definir precisamente os conceitos de domínio robótico;
- Certificar um entendimento comum entre as partes interessadas envolvidas em qualquer parte do ciclo de vida dos sistemas robóticos;

- Habilitar uma eficiente e fiável disseminação de dados e transferência de informação entre os sistemas robóticos;

- Permitir a transferência inequívoca de conhecimento entre qualquer grupo de seres humanos, robots e outros sistemas.

Os *standards* existentes são específicos a uma certa área que necessitava de ser colmatada como: vocabulário; critério de performance e métodos de teste relacionados; interfaces mecânicas; requisitos de segurança; interface gráfica do utilizador para programar e operar ou terminologia robótica.

Os benefícios da interoperabilidade destes sistemas são compreensíveis, nomeadamente a aquisição de componentes que são facilmente obtidos a um custo reduzido pois existiria uma maior comunidade de fabricantes. O desenvolvimento de *software* seria exponencial tendo em conta que a maior parte dos engenheiros informáticos estariam focados em objetivos semelhantes e a partilhar sinergias para o mesmo tipo de sistema ou linguagem (Madhavan, 2009).

Nos dias que correm existem vários veículos não tripulados que operam pelo ar, solo, superfície ou sub-superficie (UxV) usados por forças NATO. É previsível que no futuro esse número aumente significativamente. Existe também a tendência para aumentarem as interações entre estes sistemas, tornando-os conscientes da presença uns dos outros, a executarem tarefas que requerem a cooperação entre eles e finalmente a implementação de comportamentos em frota (Casbeer, 2006). Apesar de terem sido propostas varias padronizações, a maioria dos sistemas possui o seu próprio protocolo de comando e controlo e, consequentemente, uma *GCS* própria. Esta profusão de protocolos torna muito difícil a implementação da cooperação entre sistemas. A operação e manutenção destes sistemas em ambientes assoberbados de veículos apresentam uma carga excessiva pela falta de padrões comuns. Uma solução possível será desenvolver *gateways* de uns sistemas para outros. Esta solução apresenta-se como hipótese, no entanto, é computacionalmente ineficiente (Grocholsky, 2006).

Caso os investigadores e a indústria estivessem mais familiarizados com as normas existentes desde início, teria sido mais fácil adotá-los desde os seus primórdios, para assim, o processo ser mais tarde simplificado com o objetivo de promover a cooperação. Este mesmo pressuposto poderia ter levado a uma mudança qualitativa na forma como usamos os veículos não tripulados hoje em dia, permitindo a implementação de novos conceitos de operação para cenários onde se insiram múltiplos veículos (Casbeer, 2006, Grocholsky, 2006).

Cada UAV comunica com a GCS associada com uma linguagem própria e sempre que alguém compra um UAV será também necessário comprar uma consola que comunique no mesmo tipo de linguagem (Bailliel, 2007, Scattolini, 2009). Caso existisse uma única linguagem, não seria necessário ter este aspeto em conta e a compra de outro veículo seria também com um custo mais reduzido pois a consola comprada anteriormente poderia ser utilizada neste. É possível então referir dois aspetos que podem ser considerados os dois principais benefícios de uma arquitetura de comunicação de referência. O primeiro é o aspeto logístico que, com o desenvolvimento de uma arquitetura padrão, será então possível ter acesso a qualquer tipo de equipamento que estará perfeitamente associado a qualquer tipo de sistema. O outro benefício está relacionado com a formação de pessoal para desenvolver este tema. É possível fazer a analogia com a criação do modelo OSI (Open Systems Interconnection) em sistemas de telecomunicações (em detrimento do modelo TCP/IP) com o objetivo da criação de um modelo para os protocolos de comunicação entre sistemas de comunicação ponto a ponto. Este modelo aglomera todos os complexos processos e assim reduz a complexidade de cada camada bem como a correção de erros e inserção de novas funcionalidades em cada camada sem ser necessário alterar todo o sistema. Ao abrigo destas possibilidades, todo o pessoal envolvido neste tipo de sistema focou-se inteiramente no desenvolvimento desta facilidade (Scattolini, 2009). A partir deste momento passou a existir uma arquitetura de referência facilitando a formação de estudantes e novos *designers* neste tema menos complexo e menos disperso. Com a aceitação de uma arquitetura de referência

para a comunicação entre UAVs e GCS, benefícios semelhantes poderiam ser usufruídos por todos as partes envolvidas (Baillieul, 2007).

Tendo em conta o descrito anteriormente, o principal objetivo desta dissertação de mestrado é a experimentação de um protótipo de *software* que faça a conversão da linguagem de comunicação MAVLink para o protocolo de comunicação padrão STANAG 4586. O objetivo seguinte será abordar a conversão com uma análise temporal de modo a verificar a validade deste processo tendo em conta os requisitos operacionais dos sistemas alvo.

Para a realização desta dissertação foi assinado um protocolo entre a Marinha e a UAVision (empresa aeronáutica, sediada em Torres Vedras) com a qual irei cooperar e serão um vetor de indicação de qual o caminho a seguir para alcançar o pretendido, tendo também interesse no sucesso da mesma. Os sistemas utilizados por esta empresa comunicam via a linguagem de comunicação MAVLink e por conseguinte, essa será a linguagem que será convertida.

Objetivo do Trabalho

Este trabalho surge da necessidade existente proveniente da não padronização de protocolos de comunicação. O objetivo é implementar um *software* computorizado associado a um piloto automático que comunica em MAVLink, de modo a ser possível validar a conversão da linguagem para STANAG 4586, bem como monitorizar o tempo de resposta do sistema a cada uma das estruturas de comunicação.

Este *software* deverá corresponder a alguns requisitos de modo a validar a conversão, como estar em total concordância com a linguagem de comunicação MAVlink; possuir os requisitos mínimos de comunicação estrutural que o STANAG 4586 pretende implementar; estruturar uma comunicação de protocolos válida tanto do MAVlink como do STANAG 4586; possuir tempos de resposta válidos tendo em conta o sistema onde está inserido e ainda possibilitar a visualização do mesmo de forma intuitiva.

Estrutura da Dissertação

Para além da introdução, esta dissertação contém 3 capítulos. No capítulo 1 é descrito o estado da arte no que a sistemas aéreos não tripulados e linguagens de comunicação diz respeito, começando nos seus primórdios até ao que atualmente se utiliza. As linguagens abordadas são o STANAG 4586 e o MAVLink. No capítulo 2 é explanada a metodologia de investigação em vigor durante todo o desenvolvimento desta dissertação. O capítulo 3 descreve todo o trabalho desenvolvido em termos de *software* e descreve o hardware utilizado. Finalmente, a conclusão faz uma síntese do trabalho realizado, onde são apresentadas algumas conclusões consoante a analise dos resultados obtidos. São ainda especificadas algumas propostas para trabalho futuro.

Este documento inclui ainda um apêndice onde se inclui todas as linhas de código do programa final oriundo do trabalho desenvolvido. De referir que os comentários no código (a verde) estão localizados com o intuito de melhor compreensão do código (comentários em inglês).

1. Estado da Arte

1.1 Sistema Aéreo Não Tripulado

1.1.1 Historia

Tabela 2: Analise temporal dos UAVs

Séc. XIX		1910s		1930s		1940s		1960s	1980s	1990s
Bomba Aérea de Perley	Kite Vigilância de Perry	Torpedo Aéreo de Sperry	Torpedo Aéreo Kattering	Queen Bee	Aviões Radio-comandados	V – 1 Alemão	PB4Y - 1	Ryan Firebee	Scout	RQ – 1 Predator

A história dos UAVs desde os seus primórdios até aos dias de hoje pode ser descrita tendo em conta uma ordem cronológica crescente que irá ser descrita como demonstrado na tabela 2.

- **Séc. XIX**

  Anos antes da primeira aeronave pilotada, em 1903, tecnologia de UAVs primitivos foi utilizada para combate e vigilância em pelo menos duas guerras.

Bomba Aérea de Perley

Em Fevereiro de 1863, dois anos depois do início da guerra civil dos Estados Unidos da América (EUA) (1861-1865), um inventor de *New York City* chamado Charles Perley registou a patente de um bombardeiro aéreo não tripulado. Perley projetou um balão de ar quente levava um cesto carregado de explosivos ligados a um mecanismo ativado por tempo (Figura 1). O temporizador ativava o cesto articulado que largava os explosivos, iniciando a ignição no processo (Henderson, 2014).

Perley recomendou que os operadores enviassem primeiro balões de teste para que pudessem testemunhar os efeitos das correntes de ar de modo a configurar o temporizador adequadamente, o que não se verificou. As duas forças utilizaram este sistema de bombardeamento mas com sucesso limitado (Henderson, 2014).

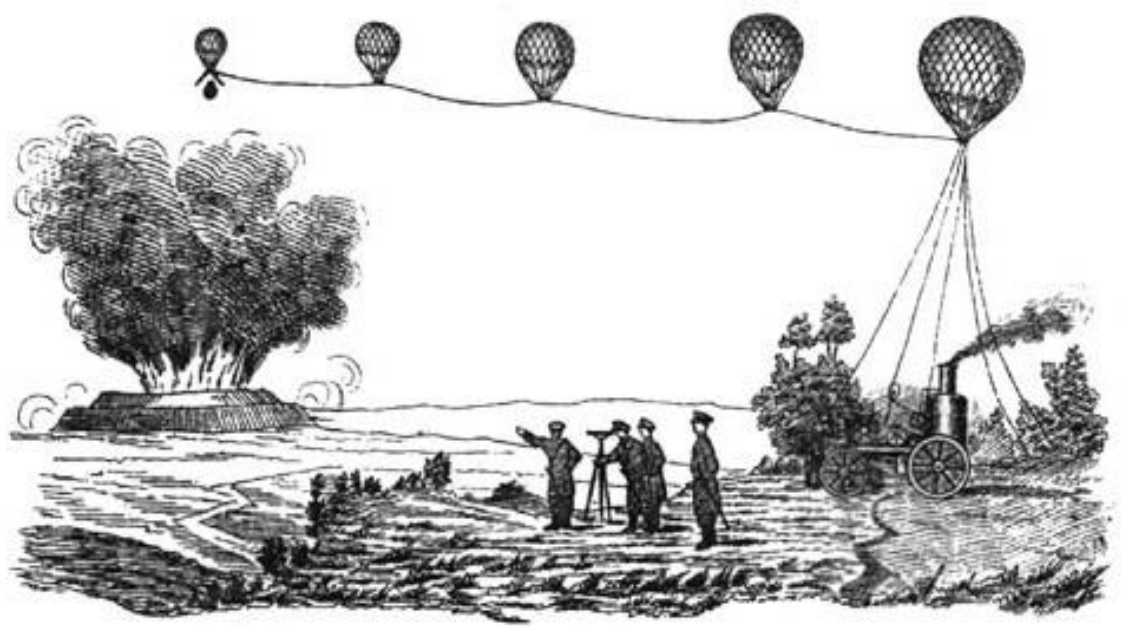

Figura 1: Bomba Aérea de Perley[2]

Kite de Vigilância de Eddy

Douglas Archibald, um Inglês que testava kites e a velocidade do vento, foi quem tirou as primeiras fotografias aéreas com recurso a um kite de grandes dimensões em 1883. As fotografias de Douglas foram amplamente divulgadas (Figura 2) até que um soldado norte-americano reconheceu no projeto uma potencialidade de aplicação militar (McGrew, 2009).

Durante a guerra Hispano-americana de 1898, o Cabo William Eddy tirou centenas de fotografias de vigilância a bordo de um kite semelhante ao de Douglas, com umas pequenas diferenças. Muitas das fotografias capturadas por Eddy (as primeiras fotografias de vigilância em tempo de guerra da historia) forneceram informação crucial às tropas americanas sobre as posições e fortificações dos seus adversários (McGrew, 2009).

[2] www.ctie.monash.edu.au, Fevereiro de 2016

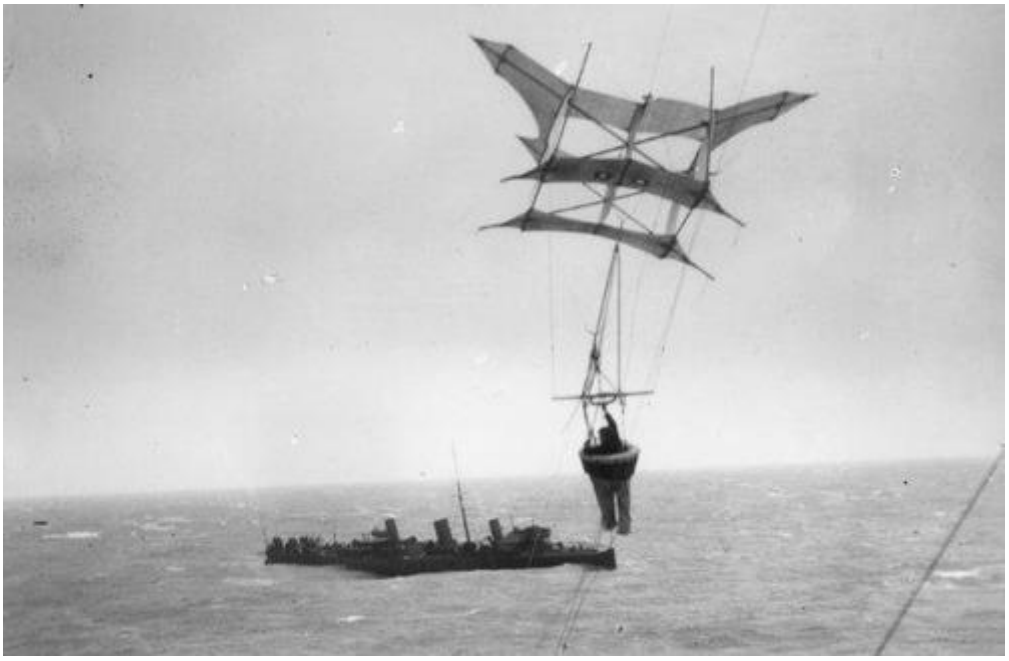

Figura 2: Kite de Vigilância de Eddy[3]

- **1910s**

  Durante a I Guerra Mundial, o primeiro UAV voo nos EUA. Embora os voos de teste fossem irregulares, os militares reconheceram o seu potencial no combate.

Torpedo Aéreo de Sperry

Em 1917, o Dr. Peter Cooper e Elmer Sperry inventaram o giroscópio estabilizador automático, que possibilita uma aeronave a voar direita e nivelada. Cooper e Sperry utilizaram a sua descoberta tecnológica para converter aeronave de treino U.S. Navy Curtiss N-9 no primeiro UAV controlado via rádio. O Torpedo aéreo de Sperry (Figura 3) voou 50 milhas a transportar uma bomba de 136 kg em vários voos de teste nunca chegando a ser utilizado em combate (Keane, 2013).

[3] feedback.arma3.com, Fevereiro de 2016

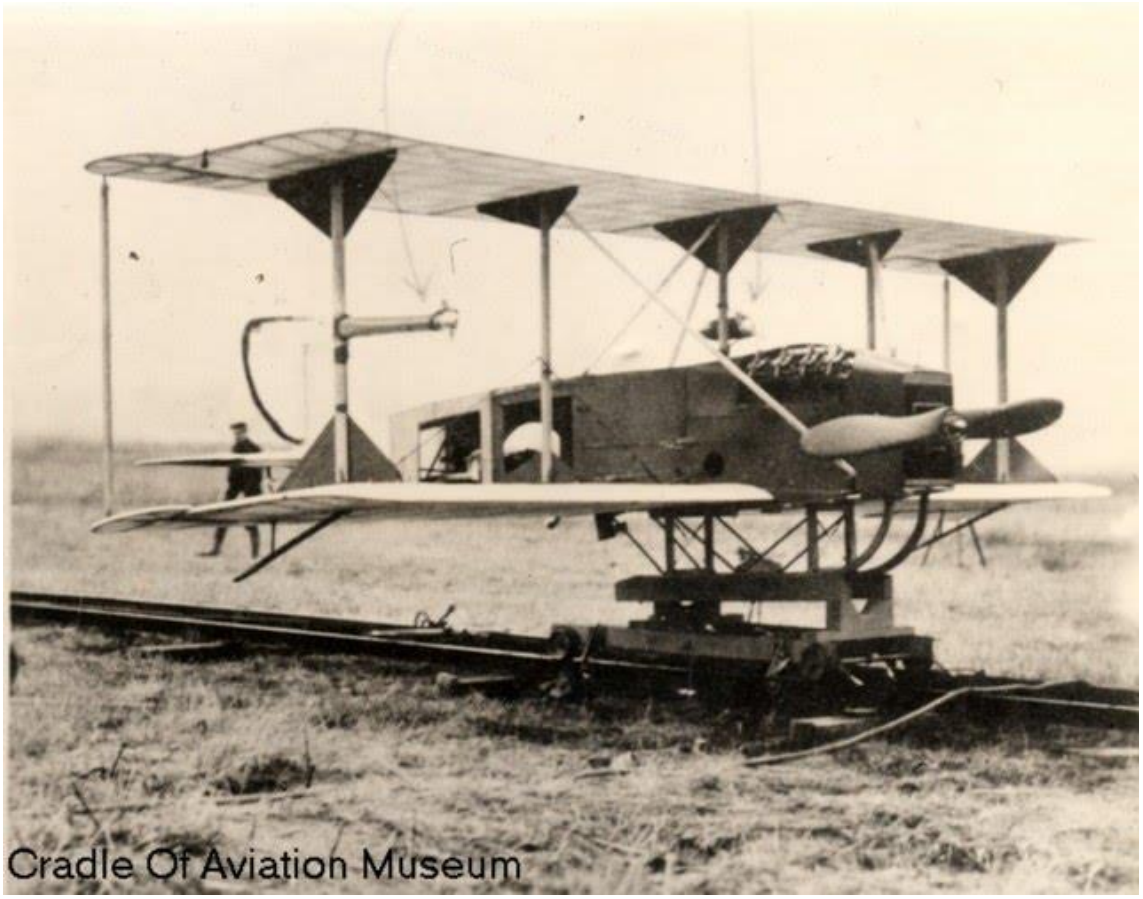

Figura 3: Torpedo Aéreo de Sperry[4]

Torpedo Aéreo *Kattering*

Feito de madeira e lona por 360 euros, o "*Kattering Bug"* (Figura 4) foi um pequeno *biplane* (avião de asa fixa com duas asas principais uma em cima da outra) equipado para carregar material explosivo de peso equivalente ao seu (136kg). Kattering da *General Motors* projetou o *Bug* para descolar de um carrinho com rodas e, em seguida desfazer-se das asas, permitindo á sua fuselagem mergulhar verticalmente em direção a um alvo pré-programado. Os EUA encomendaram grandes quantidades de *Bugs* durante a I Guerra Mundial, mas aquando do término desta foram cancelados (Keane, 2013).

[4] sites.google.com, Fevereiro de 2016

Figura 4: Torpedo Aéreo Kattering[5]

- **1930s**

  Mais de uma década depois na I Guerra Mundial, o desenvolvimento de aeronaves não tripuladas diminui significativamente. No final dos anos 30, novos UAVs emergiram como uma importante ferramenta de treino de combate.

DH.82B *Queen Bee*

O Queen Bee, o primeiro UAV reutilizável, foi projetado para uso como um alvo aéreo durante as missões de treino para os artilheiros anti-aéreos da *Royal Navy*. Os *biplanes* de madeira compensada voaram pela primeira vez em 1935 e deram origem a flutuadores para o uso dos mesmos no mar. O *Queen Bee* (Figura 5) era controlado por rádio e podia percorrer uma distância máxima de 300 milhas em mais de 100mph (metros por hora). Um total de 380 destes veículos serviu como UAVs alvo na *Royal Air Force* e *Royal Navy* até serem retirados em 1947 (Kamrani, 2007).

[5] www.cdsg.org, Fevereiro de 2016

Figura 5: DH.82B Queen Bee[6]

Aviões Radio-comandados

Em 1939, o inglês e ator de Hollywood Reginal Denny formou a *RadioPlane Company* em Los Angeles. Um entusiasta da aviação ao longo da vida e ex artilheiro aéreo, Denny formou uma equipa de engenheiros e especialista de rádio da vizinha *Lockheed Company* e começaram a trabalhar no desenvolvimento de um grande avião controlado remotamente. Nos anos seguintes, eles conseguiram produzir uma serie de UAVs de grande sucesso chamados *OQ Targets* (Figura 6) (Ross, 2008).

[6] www.airplane-pictures.net, Fevereiro de 2016

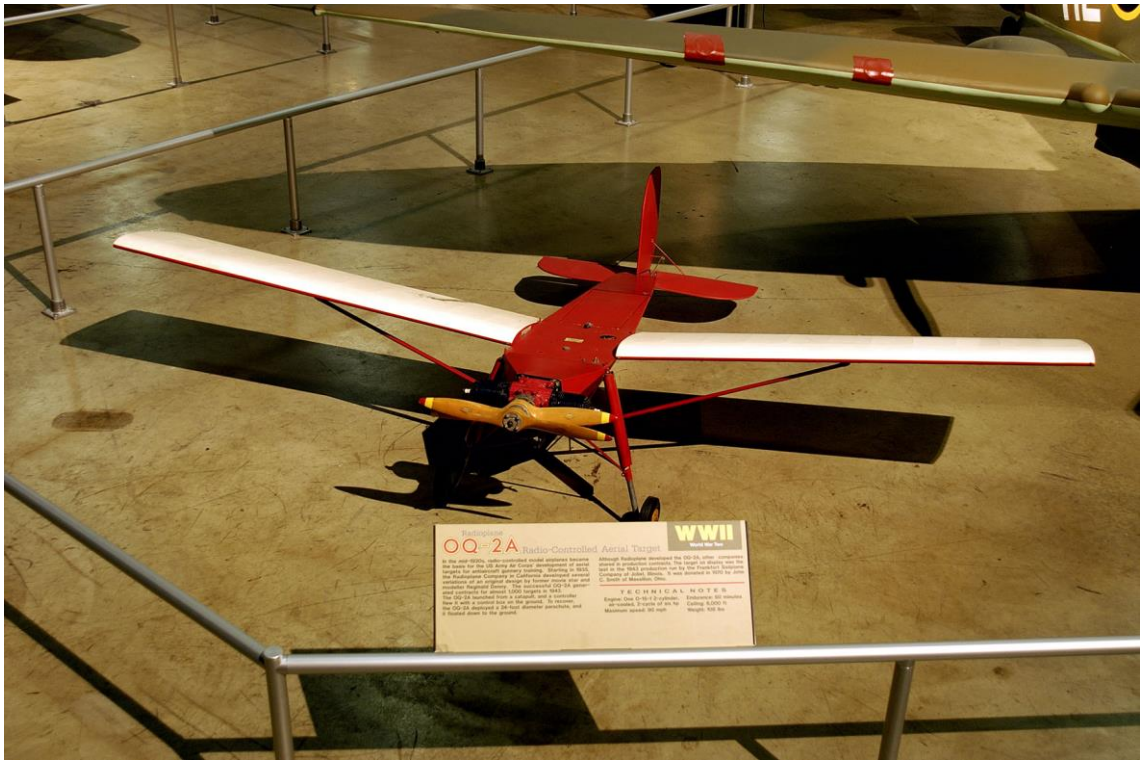

Figura 6: OQ Targets[7]

- **1940s**

  Durante a II Guerra Mundial, o inovador V – 1 da Alemanha nazi demonstrou a ameaça que um UAV podia constituir em combate.

V-1 Alemão

No início da II Guerra Mundial, Adolf Hitler encomendou uma bomba voadora com o objetivo de ser usado contra alvos não militares. Fieseler Flugzeuhau projetou o Fieseler Fi – 103, mais conhecida como Vergeltungswaffe – 1 ou V-1 (arma de vingança, Figura 7), para ser lançado através de uma longa rampa e voar a 470 mph. O UAV V-1 era alimentado por um propulsor de pulso jet que produzia um som característico semelhante a um zumbido. O primeiro lançamento foi contra a Grã-Bretanha em 1944 em que os V-1 mataram mais de 900 civis e feriram mais de 35 mil pessoas (Mira, 2014).

[7] www.nationalmuseum.af.mil, Fevereiro de 2016

Figura 7: Vergeltungswaffe – 1 ou V-1[8]

Ainda durante a II Guerra Mundial, a marinha dos USA desenvolveu UAVs com o objetivo de destruir os locais de lançamento dos V-1. O PB4Y-1 e o BQ-7 (Figura 8) descolaram com dois pilotos a bordo que pilotavam o avião até uma certa distância e configuravam a rota para os alvos antes de se ejetarem. Estas operações foram um sucesso e marcam a primeira vez que um UAV foi usado contra outra UAV (Raithel, 1994).

[8] www.nationalmuseum.af.mil, Fevereiro de 2016

Figura 8: O PB4Y-1[7]

- **1960s**

Após o seu uso inicial como drones alvo e veículos de combate controlados remotamente, os UAV desempenharam um novo papel durante a guerra do Vietnam: espionagem.

AQM-34 Ryan Firebee

Em 1960 a Força Aérea dos EUA começou o seu primeiro programa de aeronaves camufladas e modificaram os UAVs de combate para missões de reconhecimento reduzindo a assinatura radar colocando cobertores de absorção radar nos lados da fuselagem e pintando o UAV com tinta anti radar (Figura 9) (Bone, 2003).

Figura 9: AQM-34 Ryan Firebee[9]

- **1970s**

O sucesso do Firebee continuou após o término da guerra do Vietnam. Em 1973, durante a guerra Yom Kippur os UAVs foram usados pela primeira vez como *decoy (isco).*

- **1980s**

A Força Aérea Israelita desenvolveu vários novos UAVs importantes, versões que foram integradas nas frotas de outros países.

Scout

Em 1978, Israel constrói o Scout (Figura 10), uma aeronave com motor a êmbolo feita de fibra de vidro que emitia uma assinatura radar extremamente baixa o que juntamente com o pequeno tamanho do UAV o tornou quase impossível de

[9] www.flickr.com, Fevereiro de 2016

abater. Este UAV de custo reduzido podia transmitir em tempo real dados de vigilância de 360 graus através de uma câmara de televisão (Merola, 1996).

Figura 10: Scout[10]

- **1990s até hoje**

  UAVs têm uma posição permanente e crucial no arsenal de tecnologia militar de todos os países. Passaram também a desempenhar tarefas de cariz pacífico como monitorização do ambiente.

RQ-1 Predator

Tendo demonstrado o seu valor tanto nos Balcãs como no Afeganistão, este UAV (Figura 11) é um ativo da Força Aérea dos EUA. Com um alcance de 450 milhas este UAV pode fornecer de 14 a 16 horas de vigilância, câmaras infravermelhas e SAR (*synthetic aperture radar)* antes de regressar á base. Uma equipa terrestre numa estação de controlo terrestre remota controla a aeronave seja por ligação rádio em linha de vista ou por comunicação satélite. Pode ser equipado com misseis antitanque Hellfire e as suas missões foram um sucesso (Valdes, 2009)

[10] www.pbs.org, Fevereiro de 2016

Figura 11: RQ-1 Predator[11]

1.1.2 Definição

Apesar de o elemento mais visível ser o Veiculo Aéreo Não Tripulado, é mais correto considerar todo o Sistema Aéreo Não Tripulado (UAS). Sistema não tripulado (*Unmanned System*) é um desígnio que abrange todos os veículos que não contemplam um humano a bordo com capacidade de controlar e/ou manobrar o sistema (Heisenbeiss, 2004). Existem vários modos de operação e controlo destes sistemas como completamente autónomos, semiautónomos (*way points*), telecomandados ou controlados remotamente. (Huang, 2008). Estes sistemas estão divididos de acordo com o ambiente onde operam como o mar (superfície ou sub-superficie), pelo ar ou pelo solo. A sua essência é semelhante entre eles, no entanto, cada um é adaptado tendo em conta o ambiente onde opera bem como os componentes que comporta. A interoperabilidade entre cada um destes sistemas está neste momento a ser desenvolvida um pouco por todo o mundo e cada vez mais se demonstra que o futuro deve seguir essa direção (Board, 2005, Dudek, 1996).

[11] www.avionale.com, Fevereiro de 2016

Em particular, de acordo com o STANAG 4586, é necessário ter em conta os seguintes elementos (Figura 12) distintos quando a um UAS se faz referência:

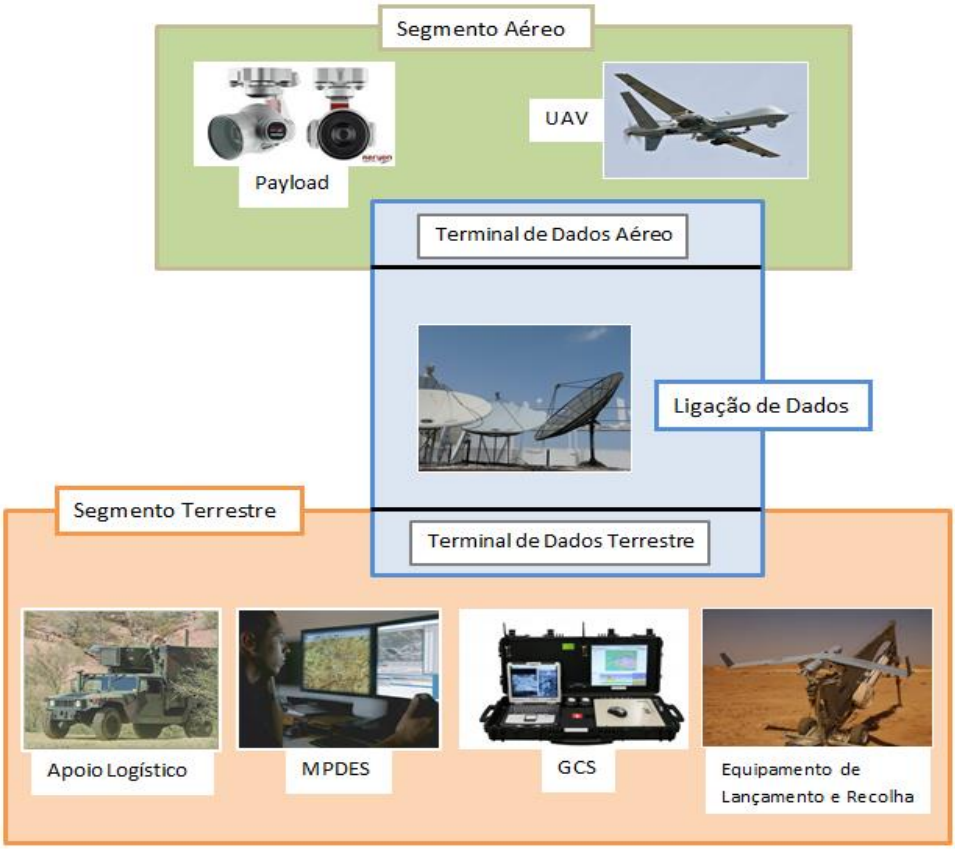

Figura 12: Elementos de um UAS

A principal distinção é entre os elementos terrestres e os elementos aéreos que comunicam via uma ligação de dados (*datalink*). Cada um destes segmentos pode ser decomposto pelos vários elementos como descrito a seguir.

1.1.2.1 Segmento Aéreo

O segmento aéreo é constituído pelo veículo e pela sua carga útil (payload). De referir que o UAV não deve ser confundido com veículos balísticos, misseis cruzeiro ou projeteis de artilharia por mais tecnologia inerente que lhes seja comum (Weatherington, 2005).

Veículos Aéreos não tripulados ou *Unmanned Aerial Vehicles* são aeronaves autónomas não tripuladas equipadas com sistemas de controlo igualmente autónomos.

Navegação autónoma ou controlada remotamente é um sistema dinâmico que mantém a elevação e impulso aerodinâmicos necessários para que um UAV possa voar, em detrimento de comandos acionados manualmente por um piloto a bordo.

O *American Institute of Aeronautics and Astronautics* (AIAA) define o UAV como "uma aeronave desenhada ou modificada, sem a necessidade de um piloto a bordo que é operada por um *input* eletrónico inicializado pelo controlo de voo ou por um sistema de controlo a bordo de gestão de voo que não requer qualquer intervenção de controlo de voo".

Existem algumas tecnologias necessárias para o voo de um UAV:

- Microprocessadores e sensores de navegação
    - Com toda a lógica, este tipo de componentes são necessários á navegação aérea de um UAV de modo a alcançar com sucesso o objetivo da sua missão (Valanis, 2008).
- Sistemas de comunicação
    - É usualmente designado de ligação de dados e tem de assegurar a adaptabilidade, flexibilidade, segurança e controlo cognitivo da largura de banda, frequência e o fluxo de informação/dados para que um UAV consiga comunicar (perceber e ser percebido) por *Ground Control Stations* ou por outros sistemas não tripulados (Valanis, 2008).
- Comunicação, comando e controlo da estação fixa
    - A infraestrutura por camadas em desenvolvimento nesta matéria vai permitir num futuro próximo que um único operador controle vários veículos não tripulados (Valanis, 2008).
- Inteligência a bordo da aeronave

- A capacidade informática de um UAV em processar toda a informação recebida e requerida para alcançar certas tarefas está diretamente relacionada com a capacidade de processamento de informação dos processadores que o sistema consegue suportar (Valavanis, 2008).

Minimizar o ruido e maximizar a autonomia e a simplicidade são algumas das maiores preocupações sobre a propulsão. Grande parte dos UAVs estão equipados com motores elétricos, pois são aqueles que melhor se adequam às condições referidas. Dependendo da organização, o custo também poderá ser um aspeto a ter em conta, mas será sempre uma decisão particular de cada empresa ou organização.

Como a propulsão, nestes sistemas, é frequentemente 60% do seu peso, a sua otimização passa a ter um papel preponderante neste tópico. A propulsão inclui: hélices; motor elétrico; fonte de energia; caixa de velocidades e sistema de arrefecimento (ambos opcionais) (Gur, 2009).

Quando se discute os tipos de propulsão que os sistemas autónomos podem usar podem ser abordados quase todos os conhecidos, mas o que é necessário ter em conta é a eficiência necessária alienada com o mais baixo custo possível (Duelley, 2010, Kolomeitsev, 2008). A propulsão deste tipo de plataformas pode ser com motores *diesel* a dois ou quatro tempos, elétrica ou combinada. Baterias para armazenar e fornecer os motores elétricos são também uma possibilidade. Outra possibilidade é ter geradores a *diesel* para fornecer a propulsão elétrica a não ser que as necessidades sejam menores que duas horas entre carregamentos (Bettner, 1995). Este sistema é bem mais caro que a propulsão a motores a *diesel*. Em contrapartida os sistemas elétricos requerem baterias e equipamento de controlo que em comparação também não é muito económico (Sebastian, 2011, Izadi-Zamanabadi,1999). Atualmente existem alguns veículos que combinam diferentes formas de propulsão para que possam usufruir dos vários benefícios das várias formas combinadas (Chan, 2002).

A propulsão elétrica perfeita depende dos requisitos e características da missão designada para o UAV, mas é claro que terá de ter em consideração simultaneamente as hélices, o motor elétrico e a bateria. Da interação destes três componentes resulta grande parte da taxa de desempenho de um UAV (Nagel, 2006, Adkins, 1994).

Payload ou carga útil é considerado todo o conjunto de equipamentos ou sensores que não fazem parte de uma plataforma quando foi originalmente construída, mas que podem ser introduzidos de modo a facilitar, ajudar ou possibilitar o veículo a alcançar o sucesso da sua missão. RADAR (*Radio Detection and Ranging*) é um sistema que utiliza ondas rádio para determinar a distância, direção e velocidade de contactos que se localizem no nosso ambiente envolvente (Skolnik, 1962). O Radar Sintético de Abertura (SAR) utiliza o movimento de rotação da antena radar sobre uma região alvo de modo a proporcionar uma maior resolução espacial de imagens que podem ser representações a duas ou três dimensões de um objeto (Cutrona, 1990). Sistema Radar de multifeixe é uma antena que forma dois lobos, um para receber e transmitir dados e o outro unicamente para receber de modo a suprimir ecos falsos (Urabe, 2000). O Sonar é um equipamento que utiliza a propagação do som na água para detectar objectos na coluna de água (Blondel, 1997). O *Light Detection and Ranging* ou LIDAR é um sensor que mede e mapeia a distância a alvos (Levinson, 2011). Câmaras são instrumentos óticos capazes de gravar imagens e vídeo passiveis de serem transmitidos em tempo real para uma *Ground Control Station*. A câmara omnidirecional é usada em odometria visual e para resolver simultaneamente os problemas de localização e mapeamento devido á sua capacidade de visão de 360° (Gregor, 2002). Câmaras de imagem térmica transformam radiação infravermelha em luz visível de modo a ser percetível pelo ser humano (Nanda, 2002). Com uma câmara híper-espectral é possível de elevar o espectro continuo ou a assinatura espectral de uma imagem contrariamente a valores de intensidade em comprimentos de onda discretos (Brelstaff, 1995).

Sensor é o equipamento que é utilizado para medir um parâmetro que é depois processado nos processadores adequados de modo a ser utilizado, por exemplo, no cálculo da otimização da melhor rota a seguir.

Global Position System ou GPS é um equipamento que disponibiliza aos seus utilizadores estabilidade a longo prazo e cobertura mundial em quaisquer condições ambientais, fornecendo a localização e tempo real. Os satélites GPS transmitem continuamente o seu tempo atual e posição. Um recetor GPS monitoriza múltiplos satélites e resolve equações para determinar a posição exata do recetor e o seu erro em relação ao tempo real (Kim, 2003). A Unidade de Medida Inercial (UMI) mede dados como orientação ângular e forças gravitacionais. O Sistema de Navegação Inercial (SNI) utiliza acelerómetros e giroscópios que trabalham com os dados fornecidos por outros sensores como bússola*, Acoustic Doppler Current Profiler* (ADCP) e *Doppler Velocity Log* (DVL) (Bryson, 2004, Lustosa, 2011), de modo a melhor planear a rota para o destino pretendido. O velocímetro é um indicador que exibe a velocidade instantânea de um objeto (Sojourner, 1990). O altímetro, como o nome indica, mostra a altitude a que se encontra relativamente a um determinado nível fixo (Barber, 2007). O Barómetro indica a distância percorrida pelo objeto (Amidi, 1998). O anemómetro é um dispositivo utilizado para medir tanto a velocidade como o ângulo de aproximação do vento (Moat, 2005). O tubo de pitot é um instrumento de medida de pressão que calcula a velocidade de fluxo de um fluido e pode calcular a velocidade de um UAV no ar ou de um USV na superfície do mar. Se estiver ligado a um sensor de pressão diferencial pode indicar a pressão dinâmica (Grasmeyer, 2001). O variómetro ou indicador de velocidade vertical (VSI) indica a taxa instantânea de descida ou subida de um corpo (Alexandrov, 2004).

1.1.2.2 Segmento Terrestre

Caracterizando mais especificamente o segmento terrestre, o elemento principal é a Estação de Controlo Terrestre onde os operadores controlam o veículo e/ou a carga útil. GCS é um equipamento de controlo de veículos que pode estar localizado tanto em terra como no mar e providencia o controlo dos mesmos.

Disponibiliza a facilidade de criação ou alteração de waypoints, a redefinição de objetivos ou tarefas bem como a monitorização de todos os movimentos e dados fornecidos pelos mesmos (Segor, 2010, Jovanovic, 2008). O interface homem-máquina é intuitiva mas numa situação em que seja necessário tomar uma decisão espontânea em que tanto o tempo de decisão humano como o tempo para introduzir os comandos sejam demasiado longos para o pretendido, existe já em desenvolvimento uma GCS que receberá os comandos por voz de modo a preencher esta lacuna (Draper, 2003).

Waypoints são um conjunto de coordenadas que identificam um ponto físico no espaço. As coordenadas usadas podem variar consoante o tipo de aplicação. Para navegação terrestre, são partes constituintes destas coordenadas tanto a latitude como a longitude. Em navegação aérea estas incluem também a altitude. Este tipo de orientação só passou a ser difundido para uso em navegação desde o desenvolvimento de sistemas de navegação avançados como o GPS. *Waypoints* localizados na superfície da terra são normalmente definidos em duas dimensões, o que não é viável quando se opera na atmosfera terrestre em que são utilizadas três dimensões (Storey, 2006).

As estações de planeamento de missão e exploração de dados (Mission Planning and Data Exploitation Stations- MPDES) são elementos opcionais usualmente utilizados em sistemas avançados. Equipamento de Lançamento e Recolha (Launch and Recovery) é característico de UAVs de pequenas dimensões não capacitados de realizar essas mesmas ações de forma convencional ou como forma de provisionamento para espaços limitados. O lançamento ou descolagem dos veículos, principalmente os aéreos, é uma ação que requere grande atenção no seu processo. Alguns destes veículos requerem plataformas elétricas, mecânicas ou hidráulicas devido tanto ao seu peso como dimensão. Veículos de baixa dimensão podem ser facilmente lançados manualmente.

A recolha ou aterragem dos veículos é um processo mais complexo devido ao risco de quebra ou destruição de material com quedas, por exemplo. Um UAV necessita de um solo consistente onde possa fazer uma aterragem mas, como já foi utilizada, existe a possibilidade de utilizar uma rede de abordagem como meio de

aterragem quase sem risco para o equipamento (Garcia-Pardo, 2002, Johnson, 2001).O suporte logístico é fundamental para a operabilidade do sistema como por exemplo a unidade terrestre fornecedora de energia.

1.1.2.3 Ligação de dados

O *Datalink* é uma ligação de dados entre dois sistemas que, neste caso, é utilizado para descrever a conectividade entre um veículo e uma GCS ou entre dois veículos com o objetivo de transmitir e receber informação (Pascoal, 2000). *Wi-Fi* é uma rede local sem fios baseada no protocolo 802.11 do IEEE que possibilita a comunicação entre todos os equipamentos que estejam ao alcance desta rede de curto alcance (Wang, 2006). GSM (Global System for Mobile Communications) é o sistema global para comunicações móveis que é usado pelos telemóveis mas pode também ser usada por veículos autónomos e GCSs com recurso a um protocolo digital de 2ª geração (2G) (Wzorek, 2006). Comunicações subaquáticas são possíveis devido a *modems* que utilizam um tipo de modulação que suprime a dispersão da propagação na água e não necessita de grandes larguras de banda (Freitag, 2005). A comunicação via satélite é útil em mar aberto porque retransmite e amplifica os sinais radio com a utilização de um *transponder* (Sherman, 2001).

O subsistema de ligação de dados é dividido em dois terminais, o terrestre e o aéreo. É possível de referenciar que existem ocasiões em que são utilizadas duas ligações de dados para a comunicação em linha de vista (Line of Sight- LOS): uma encarregada das funções de comando e controlo específicos da plataforma e a outra de controlo da carga útil. No caso de operações fora da linha de vista (Beyond Line of Sight- BLOS) existem duas possibilidades: retransmissão rádio ou comunicação satélite. Independente de cada caso descrito anteriormente, existem dois canais que podem ser distinguidos em cada ligação:

- *Uplink*: comandos enviados do segmento terrestre para o segmento aéreo;
- *Downlink*: dados enviados do segmento aéreo com destino ao segmento terrestre.

Cada elemento do Sistema Aéreo Não Tripulado varia em tamanho, capacidade e características de acordo com a categoria do sistema, porém, a descrição geral do sistema é a descrita anteriormente.

1.1.3 Classificação

Existem diversas classificações de UAVs que comparam diversos parâmetros e critérios. Não existe de momento nenhuma classificação universal e certificada o que possibilita a integração de várias dessas classificações ou a escolha de uma classificação geral que satisfaça os objetivos do trabalho em questão.

O critério utilizado na classificação presente na tabela 3 tem como intuito dar uma ideia geral das características mais relevantes associadas á operação e necessidades logísticas no manuseamento de um UAV.

Tabela 3: Classificação UAV (Bendea, 2008, Dalamagkidis, 2008)

Classe	**Alcance (km)**	**Endurance (horas)**	**Peso (kg)**
I (micro-mini)	<10	<2	<150
II (curto e médio alcance)	10 - 500	2 - 18	150 - 600
III (longo alcance e endurance)	>500	>24	>600

1.1.4 Missões

Os veículos não tripulados foram historicamente concebidos com o objetivo de substituir os humanos na execução das chamadas missões 3D (*Dull, Dirty and Dangerous*). As missões *Dull*, ou seja, longas são tipicamente representadas por longos voos e/ou longa execução de tarefas. Exemplos deste mesmo tipo de missões são missões de vigilância ou fases de viagens longas. Nestes casos, as limitações psicofisiológicas humanas podem afetar o sucesso dos objetivos da missão e o endurance efetivo do sistema. Problemas que seriam suplantados com a utilização de um UAS, uma vez que a persistência do veículo na área de missão apenas depende do endurance da aeronave enquanto um operador garante o correto nível de vigilância e carga de trabalho pela GCS.

As missões *Dirty*, ou seja, sujas são caracterizadas pela operação num ambiente perigoso para a saúde e bem-estar do operador, como por exemplo por radiação, poluição, ameaças químicas ou biológicas. Exemplos deste tipo de missões foram a recolha de amostras da nuvem nuclear realizada em 1946-1948, ou as atividades de monitorização sobre o reator de Fukushima em 2011.

Finalmente, missões *dangerous* ou perigosas são definidas no contexto militar, onde existem varias ameaças á vida do piloto. Em particular, a ideia original foi a de substituir os bombardeiros com humanos a bordo por veículos não tripulados unicamente para as tarefas mais perigosas como por exemplo o reconhecimento de uma área fortemente defendida.

Com o passar dos anos, o conceito das missões 3D tem vindo cada vez mais a alargar o seu âmbito a todas as áreas operacionais que nos envolvem atualmente, tanto em termos militares como civis. As missões/tarefas que os UAVs podem realizar podem então ser divididas em âmbito militar e civil, tendo em atenção que algumas destas podem ser inseridas em qualquer um dos âmbitos pois todas as forças militares já dispõe a faculdade de multiuso.

Missões/tarefas civis (Nehme, 2006):

- Transportar carga ou passageiros: entrega ou recolha de carga ou de passageiros em ambientes com condições nocivas ou perigosas;
- Monitorização: multidões, animais, florestas, poluição e recolha de amostras de ar;
- Busca e Salvamento (SAR): procura e prestação de ajuda a pessoas em perigo;
- Pesquisa: explorações de petróleo, gás e minerais;
- Exploração científica: validação da pesquisa geológica e auxiliar investigadores a ganhar um mais profundo conhecimento do ambiente de Marte, por exemplo;
- Agricultura e silvicultura: serviços de aplicação de pesticidas;
- Transmissões desportivas e realização de filmes: o público valoriza e admira as imagens captadas por uma camara voadora como se de um pássaro se tratasse e ainda todos os efeitos especiais;
- Engenharia e Construção: com o auxílio de camaras de alta resolução ou camaras de vídeo é possível conduzir inspeções em linhas de transmissão de energia entre torres de alta voltagem (Cai, 2010);

Figura 13: Monitorização de Agricultura e Assistência Médica (Rosa, 2016)

Algumas das missões/tarefas que podem ser realizadas por este tipo de sistemas no âmbito militar são:

- Reconhecimento e Recolha de informação (I&R): análise, proteção, inspeção de divulgação e exploração;
- Entrega de carga útil: apoio a operações de forcas especiais e Time-Critical Strike (tempo necessário para detetar, decidir, medir e aceder a alvos) (Brickner, 2005);
- Influence Activities (IA): enganar, dissuadir e impedir inimigos;
- Segurança Marítima: proteger portos/bases aliadas tal como os seus navios e infra-estruturas (ancoradouros e armazéns) do espectro de ameaças desde ataques convencionais a ataques terroristas;
- Guerra antiaérea;
- Vigilância: processo de monitorizar o comportamento de pessoas, objetos ou processos de conformidade com normas expectáveis ou desejáveis;
- Operações de Interdição Marítima (MIO): desviar, interromper, retardar ou destruir os inimigos da marinha mercante;
- Patrulha Fronteiriça: monitorizar, regular ou controlar o comportamento de pessoas, animais ou bens tanto na entrada como na saída de um país;
- Inspeção/identificação (ID): esta capacidade irá ajudar os departamentos de Defesa Nacional, defesa antiterrorismo e as necessidades de destruição de material explosivo;
- Apoio a forças de operações especiais: conduzir operações em que integrem formas não convencionais de combate, contra terrorismo, reconhecimento e assistência militar;
- Comunicação (voz e dados): podem funcionar como elo de ligação entre unidades, incluindo comunicação com o comando superior.

Figura 14: Aterragem de um UAV no navio e UAV com objetivos de SAR (Rosa, 2016)

1.2 STANAG 4586

1.2.1 Introdução

Em 1998 foi constituída uma equipa de especialistas da NATO composta por membros tanto do governo como pertencentes à indústria com o objetivo de criar um documento que servisse como padrão a sistemas de controlo dos veículos autónomos não tripulados. Alcançou-se então a estruturação do NATO *Standardization Agreement 4586* que serve como guia de orientação para a padronização das interfaces que privilegiam a interoperabilidade entre sistemas autónomos não tripulados (NSA, 2006), tendo a sua última edição (3ª) sido publicada em 2012.

O STANAG 4586 é um documento padrão NATO que tem como principal objetivo a definição das interfaces que devem ser implementadas de modo a atingir o nível de interoperabilidade (LOI – *Level of Interoperability)* requerido tendo em consideração os conceitos de operações (CONOPS) assumidos. O que só será possível através da implementação padrão das interfaces do sistema de controlo para comunicar com os diferentes UAVs e os seus respetivos *payloads*, assim como, os diferentes sistemas de *Command, Control, Communication, Computers and Intelligence (C4I)* (Feitshans, 2008). Estações de planeamento externas e apoio logístico estão incluídos nos sistemas C4I. Esta implementação facilita a integração de componentes provenientes de diferentes origens e premissas legais. A conformidade destes sistemas deverá ser certificada e irá aumentar a flexibilidade das forças conjuntas NATO através da partilha de ativos (NSA, 2006).

De acordo com este documento, um sistema aéreo não tripulado pode ser dividido em 5 elementos diferentes como ilustrado na figura 15. O elemento designado como *air vehicle element* é constituído pela plataforma, o sistema de propulsão e todo o sistema subjugado à gestão de voo. O elemento da carga útil (*payload*) contempla vários pacotes que tanto podem ser sensores e os seus dispositivos de gravação associados ou então um sistema de armamento. Como ilustrado, o elemento de ligação de dados é composto por dois terminais, o terminal de dados do veículo (*Vehicle Data Terminal- VDT*) e o terminal de controlo de dados

(*Control Data Terminal- CDT*). Este último pode estar localizado em plataformas de superfície, sub-superficie ou aéreas. O controlo de todo o sistema é alcançado pelo UCS (*UAV Control System*) e o elemento de ligação de dados (*Data Link*). Apesar de na figura estar inserido na componente de superfície, o UCS e o terminal de ligação de dados associado podem estar localizados em qualquer tipo de plataforma (e.g. plataforma aérea). O UCS incorpora a funcionalidade de gerar, carregar e executar a missão do UAV e ainda disseminar informação útil para os vários sistemas C4I.

De referir que a figura 15 ilustra um caminho comum para o comando e controlo do veículo, carga útil e produtos. Estas funcionalidades podem ser alcançadas em separado, em ligações de dados independentes. O elemento de lançamento e recolha incorpora a funcionalidade de lançamento e recolha do veículo aéreo.

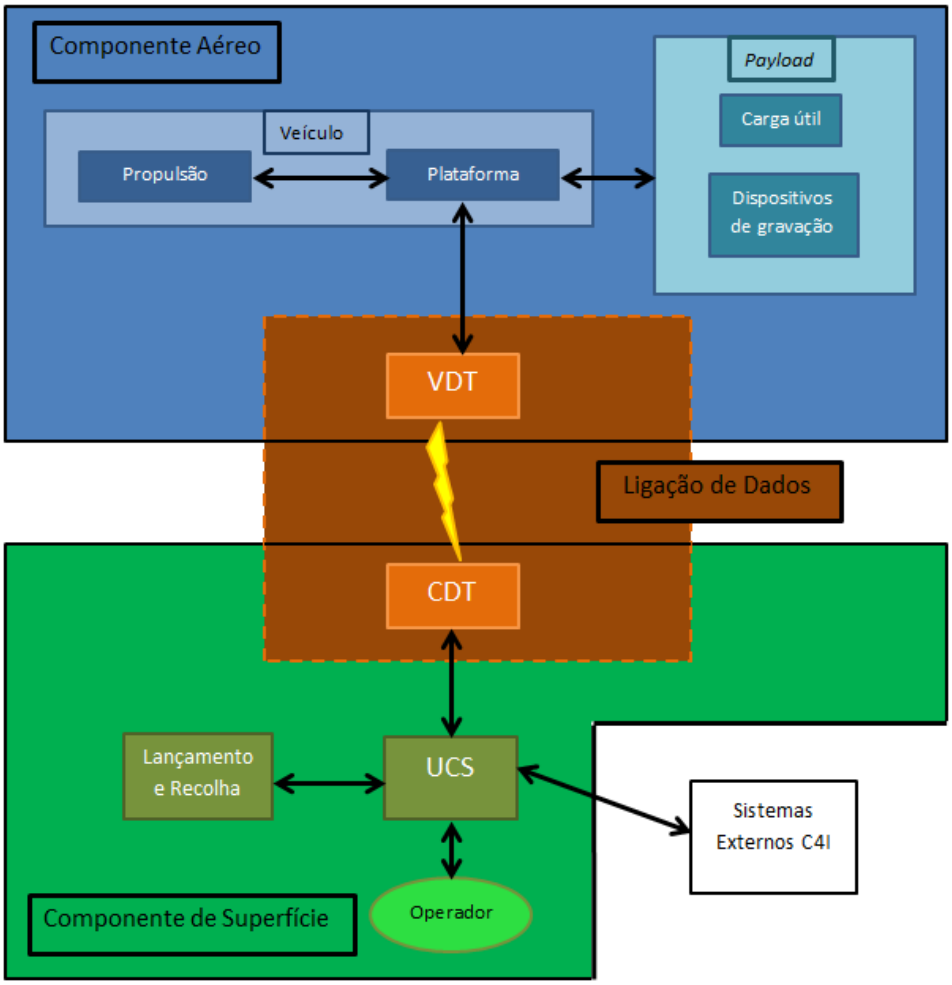

Figura 15: Sistema UAV

Atualmente muitos UASs foram desenvolvidos como sistemas de interfaces especificas e arquiteturas únicas de *software/hardware*, o que não permitem alcançar a interoperabilidade com diferentes UASs. Considerando que usualmente existem vários UASs que trabalham em conjunto com o mesmo objetivo ou missão, existe uma proliferação de GCSs e uma dificuldade de partilha de dados por parte dos sistemas C4I.

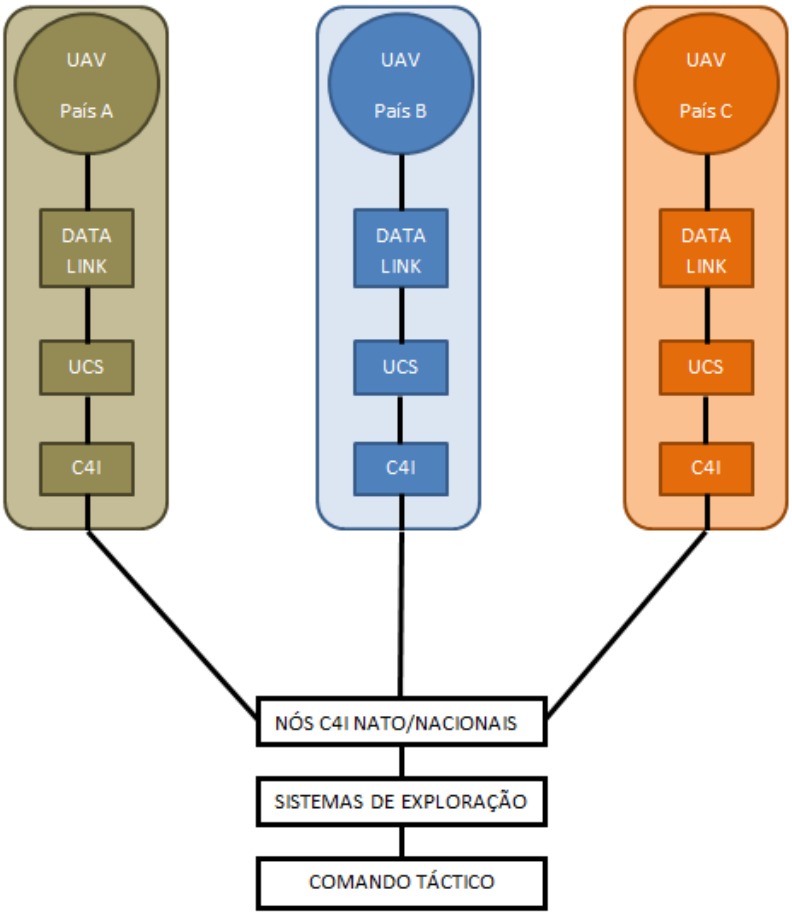

Figura 16: Organograma conjunto

Objetivando a redução de custos operacionais e o aumento de eficiência da missão em termos de exploração, disseminação e análise dos dados recolhidos, a necessidade de interoperabilidade entre UASs é visível (Figura 16). Por esta razão foi então desenvolvido o STANAG 4586 e assim, uma GCS compatível deverá idealmente estar capacitada para controlar cada UAV/carga útil compatível e trocar informação com sistemas C4I compatíveis, como ilustrado na figura 16. Na prática, tal não é

diretamente possível devido á necessidade de módulos específicos, com os seus problemas associados de integração e qualificação (críticos para a certificação civil) (Haddon, 2003). Atualmente, uma GCS será capaz de interagir com todos os UAVs e sistemas C4I para os quais foi integrada e qualificada. Com o auxílio de um guia padrão e subsequente seguimento e utilização, os processos de integração e qualificação serão mais curtos e mais fáceis, em comparação com o desenvolvimento de raiz de uma solução específica. Apesar do STANAG 4586 ter sido concebido para aplicações militares, é também diretamente aplicável a missões civis, desde que tenham os mesmos requisitos de interoperabilidade. A integração de UASs no espaço aéreo civil não é tratada neste documento.

De momento existem três edições publicadas do STANAG 4586, sendo que a última foi publicada em Novembro de 2012 (STANAG 4586, 3ª edição). Em qualquer dos casos, devido á complexidade e à abrangência do tema foi desenvolvido um guia de implementação com o objetivo de auxiliar os seus implementadores. Este guia fornece simplesmente sugestões para desenvolvimento e implementação, e não restrições necessárias. Na atualidade, tanto a 3ª como a 2ª edição do STANAG 4586 se encontram implementadas. A segunda edição do referido documento foi amplamente implementada. Devido às alterações para a terceira edição, emergia a necessidade de restruturar todo o sistema de modo a conseguir implementar esta ultima versão, o que muitos fabricantes e entidades não intenderam como primordial, continuando assim a utilizar a segunda edição.

O STANAG 4586 fornece um guia para definição das interfaces internas do sistema, e afeta, como consequência, a arquitetura do sistema, funcionalidades e a interface humano-máquina. Considerando a interface entre a GCS e o UAV (ou melhor, o modulo especifico do sistema de controlo UAS, como é referido adiante), o *standard* fornece uma série de mensagens padrão nas duas direções da ligação (*uplink* e *downlink*) e ainda possibilita a criação de novas mensagens específicas (mensagens privadas). Contudo, a utilização de um protocolo (com estrutura de mensagens e dados), inevitavelmente irá afetar o *design* das funcionalidades dos sistema e a sua

arquitetura especialmente considerando o requisito de reduzir, tanto quanto possível, os elementos específicos do veículo para atingir a máxima interoperabilidade.

1.2.2 Diferenças da 2ª para a 3ª edição

Com base na análise de uma equipa especialista em capacidade conjunta de UAVs (JCGUAV) e relatórios provenientes das indústrias e programas nacionais que suportam o desenvolvimento destes sistemas, as mudanças de uma edição para a subsequente são as seguintes:

- A definição do segundo nível de interoperabilidade foi revista de modo a ser aplicável no caso em que os dados de um sensor são recebidos num formato digital com metadados embebidos de acordo com o STANAG 4609.
- As definições dos últimos três níveis de interoperabilidade foram revistas e alteradas de acordo com a tabela 4.

Tabela 4: Alterações nas definições de níveis

Nível	2ª Edição	3ª Edição
3	"Control and monitoring of the UAV payload in addition to direct receipt of ISR/other data"	"Control and monitoring of the UAV payload unless specified as monitor only"
4	"Control and monitoring of the UAV, less launch and recovery"	"Control and monitoring of the UAV, unless specified as monitor only, less launch and recovery"
5	"Control and monitoring of the UAV (Level 4), plus launch and recovery functions"	"Control and monitoring of UAV launch and recovery unless specified as monitor only"

- Com o objetivo de reduzir os requisitos em termos de largura de banda relativamente à ligação de dados entre o UCS e o veículo, as mensagens foram modificadas nos seguintes termos:

- Adição de um vetor presença aplicável a cada campo da mensagem
- As unidades e tipos de dados das mensagens foram migrados para um formato mais eficiente (e. g. virgula flutuante para valor integral escalado)

- Adicionadas mensagens de modo a suportar um maior nível de autonomia.
- Incremento de mensagens atualizadas para uma ligação compatível com o STANAG 7085 (*NATO Interoperable Data Links for ISR Systems*) em termos de conexão, controlo e monitorização.
- Planeamento de missão interoperável por meio de extensões ao CRD (Common Route Definition).
- Erros técnicos e administrativos foram corrigidos e o editorial alterado de modo a clarificar os requisitos.

De modo a facilitar a interoperabilidade com os sistemas compatíveis com a segunda edição do STANAG 4586, os elementos constituintes dos dados foram estruturados de modo a permitir o mapeamento direto de dados dos diferentes formatos de cada edição.

1.2.3 Premissas e Restrições

Este documento foi desenvolvido seguindo as seguintes premissas e restrições:

- Elementos do sistema (e.g., Sistema de controlo central do UAV (Core UAV Control System- CUCS), Interface de Ligação de Dados (Data Link Interface- DLI), Modulo Específico do Veículo (Vehicle Specific Module- VSM), Interface de Comando e Controlo (Command and Control Interface- CCI) e Modulo Específico de Interface de Comando e Controlo (Command and Control Interface Specific Module- CCISM) não necessitam estar co-localizados.
- Os requisitos STANAG foram desenvolvidos independentemente dos conceitos de operações nacionais. A intenção não é nem definir nem inferir específicos conceitos de operações neste STANAG.

- O conteúdo deste STANAG é independente das características de *hardware* de cada UAS.

1.2.4 Nível de Interoperabilidade

1.2.4.1 Definição

De acordo com o STANAG 4586 (2012), a interoperabilidade pode-se traduzir como: "A habilidade de forças aliadas e, quando apropriado, forças parceiras e outras nações para treinar, exercitar e operar efetivamente em cooperação na execução de missões ou tarefas atribuídas."

Esta é uma definição genérica que não discrimina a possibilidade de situações intermédias.

Neste acordo são definidos cinco níveis de interoperabilidade de modo a acomodar os requisitos operacionais. O respetivo requisito operacional ou CONOPS determina o nível de interoperabilidade requerido que um específico sistema autónomo não tripulado terá de alcançar.

Nível 1: Receção indireta e/ou transmissão de sensor e metadados[12] associados.

Nível 2: Receção direta de dados de sensores e metadados associados a partir do UAV.

Nível 3: Controlo e monitorização da carga útil do UAV, a menos que especificado apenas como monitor.

Nível 4: Controlo e monitorização completos do UAV, a menos que especificado apenas como monitor, com exceção do lançamento e recolha.

Nível 5: Controlo e monitorização do UAV, lançamento e recolha, com exceção se for especificado apenas como monitor.

[12] são marcos ou pontos de referência que permitem circunscrever a informação sob todas as formas, (www.metadados.pt, 14/02/2016)

Tabela 5: Comparação dos níveis de interoperabilidade

Nível	RX/TX de dados dos sensores	RX/TX de dados do UAV	Controlo e monitorização carga útil UAV	Controlo e monitorização UAV	Lançamento e Recolha	Apenas como monitor (opcional)
1	X					
2		X				
3		X	X			X
4		X		X		X
5		X		X	X	X

De referir que os níveis de interoperabilidade mais altos não incluem necessariamente os níveis anteriores como demonstrado na tabela 5. Por exemplo, é possível ter o controlo absoluto do veículo (nível 5), sem ter controlo da carga útil do mesmo (nível 3). O controlo da carga útil (nível 3), por sua vez, compreende a receção direta dos dados associados (nível 2).

1.2.4.2 Aplicação

Os níveis de interoperabilidade acima definidos podem ser aproveitados com a padronização das interfaces entre os elementos do sistema como também entre o UCS e os sistemas externos C4I. Tal pode ser alcançado caso toda a arquitetura do sistema esteja padronizada na medida em que acomoda a implementação destas interfaces padrão. De modo a alcançar a interoperabilidade, a arquitetura do UCS e as interfaces

devem suportar os protocolos de comunicação apropriados bem como o formato de mensagens associado.

Para se localizar no segundo nível de interoperabilidade ou superior por meio de comunicações RF (*radio frequency*), o sistema deverá assegurar-se de que o CDT é interoperável com o VDT. Assim, a conectividade entre o CDT e o VDT é um pré-requisito para o nível 2 ou superiores.

Já existe, na atualidade algum número de STANAGs que são aplicáveis aos UASs. Providenciam *standards* para a interoperabilidade da ligação de dados (STANAG 7085), ligação entre a carga útil e a plataforma (STANAG 7023, 4545, 4607 e 4609) e ainda para os dispositivos de gravação a bordo (STANAG 7024 e 4575).

No entanto, não existe um *standard* que defina as interações entre o UCS e o UAV (incluindo as funções de lançamento e recolha) via CDT. Apesar do STANAG 7149, APP-11, definir um catálogo de mensagens padrão para relatórios de tarefas e de estado, não existe nenhum protocolo que defina especificamente os campos e mensagens que devem ser usadas por um UAS. Em complemento, não existe também nenhum *standard* que especifique o tipo de informação que deva ser apresentada a um operador destes sistemas tal como, a definição do nível de proficiência requerido a um operador do mesmo tipo de sistema.

O STANAG 4586 providencia a padronização destas interações. UASs que sejam compatíveis com este *standard*, incluindo os restantes STANAGs, vão possibilitar a interoperabilidade de nível 2 ou superior. Para sistemas de primeiro ou segundo nível de interoperabilidade que utilizem carga útil de imagem digital e ligações de dados compatíveis com o STANAG 7085, unicamente os *standards* NIIA (NATO ISR Interoperability Architecture) são requeridos, independentemente de a componente de superfície ser uma UCS ou outra estrutura de exploração ISR (Institute for Systems and Robotics).

Sempre que um UAS não utiliza os padrões NIIA (e.g. dados analógicos provenientes da carga útil) existe a necessidade de associar os metadados associados

com os dados analógicos. Para que isto seja possível, é necessário que os metadados apropriados sejam capturados por meio de um método padrão. Com esse mesmo objetivo, o mecanismo é utilizar mensagens padrão constantes no STANAG 4586 que irão suportar o nível de interoperabilidade apropriado.

1.2.5 Arquitetura Funcional

A arquitetura funcional do UCS necessária para suportar a interoperabilidade em UASs é a que se ilustra na figura 17.

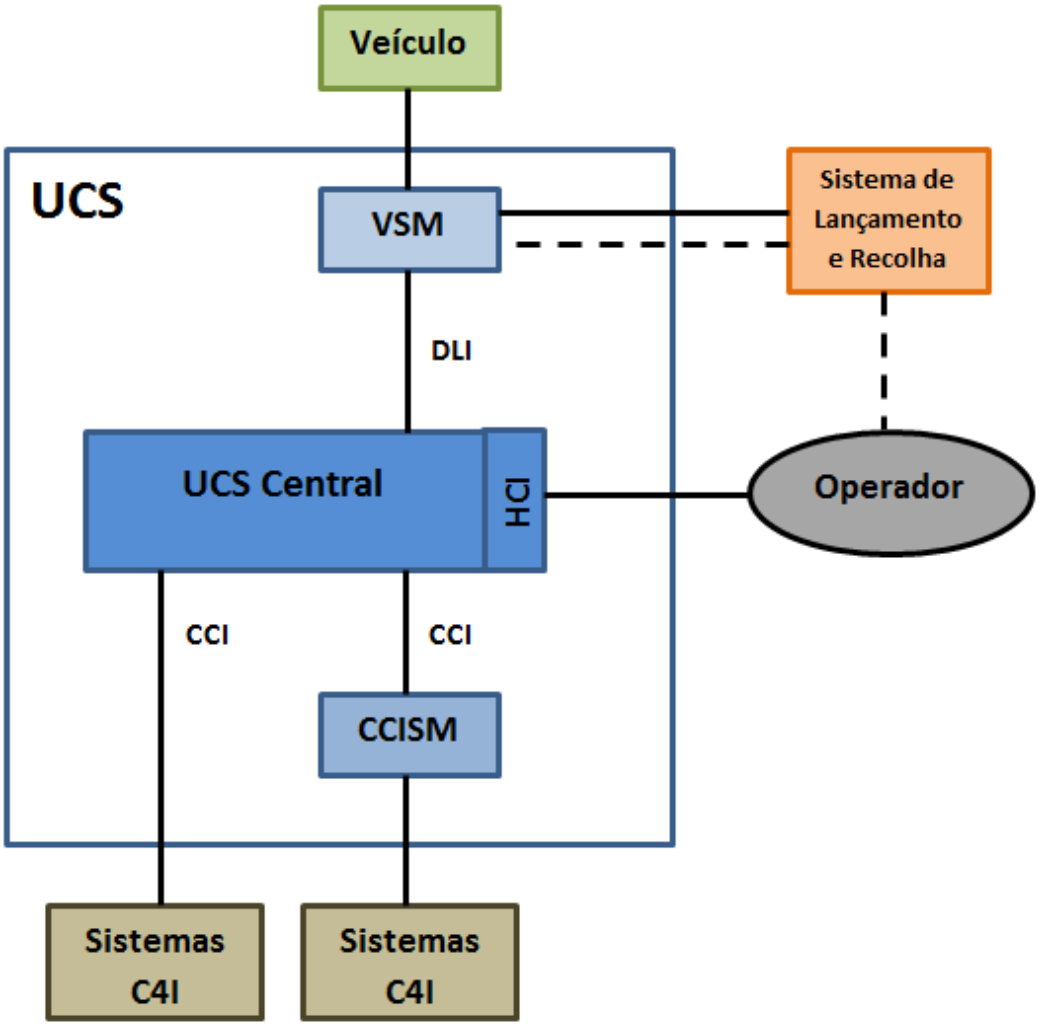

Figura 17: Arquitetura funcional do UCS

Esta arquitetura estabelece os seguintes elementos funcionais e interfaces: CUCS; DLI; CCI; VSM; CCISM; HCI.

1.2.5.1 VSM

A função do VSM é a de providenciar os protocolos de comunicação, temporização de interface e formatos de dados que cada veículo aéreo requere. Uma das principais funções do VSM é a de fornecer qualquer tipo de tradução necessária dos protocolos e formatos de mensagens do DLI para os requisitos do veículo. Caso as ligações de dados utilizados no UAS não sejam compatíveis com o STANAG 7085 (como uma ligação com pouco largura de banda que serve comunicações BLOS, e. g. comunicações satélite), o CDT associado à ligação de dados incompatível tem de ser fornecido e ligado com o UCS via a função DLI do VSM. A gestão de interfaces requerida necessária para controlar e monitorizar as ligações de dados ou a capacidade de receber e processar as mensagens de estado e comando do DLI deverão ser incorporadas no CDT. Funções VSM selecionadas, e. g. tradução de dados apresentados na forma de representação do CUCS (mensagens definidas pelo DLI) para as representações especificas do veiculo e vice-versa, podem ser incorporadas no veículo e/ou na superfície.

Sempre que um UAV é introduzido numa frota de UAS interoperáveis, poderá ser necessária a integração e validação da nova função VSM correspondente a cada UCS. O anterior só será necessário caso o novo veículo introduzido requeira a funcionalidade VSM no segmento de superfície do veículo. Caso o UCS existente inclua um CDT compatível com o STANAG 7985 e incorpore as funções de gestão de ligação de dados definida pelo STANAG 4586, e ainda, se o novo UAV introduzido implemente diretamente mensagens DLI e inclua um VDT compatível com o STANAG 7985, então as funções VSM correspondentes não são necessárias.

A localização do VSM pode ser tanto no veículo como no UCS. Tendo em conta que o CUCS não pode conter nenhum processo em tempo real que seja necessário para suportar a operação entre a plataforma e o CDT, a implementação do VSM e a sua relação com o DLI podem ser efetuadas de 4 formas, ilustradas na figura 18.

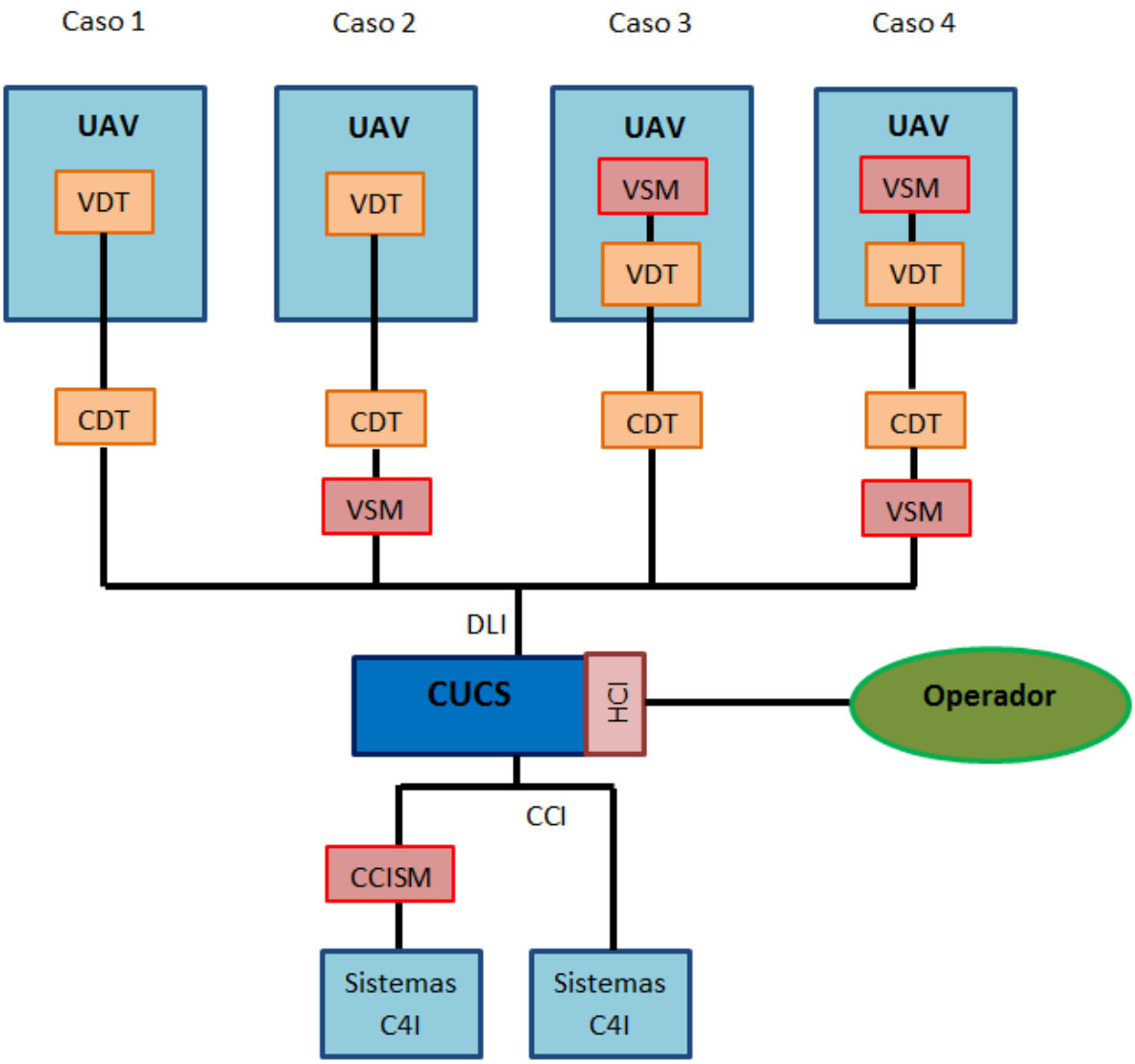

Figura 18: Implementação do VSM e a sua relação com o DLI

A Figura 18 ilustra quatro veículos que suportam diferentes níveis de DLI como a sua linguagem nativa. No caso 1, o veículo e o CDT suportam todas as funcionalidades do VSM. No caso 2, o veículo e CDT não suportam qualquer funcionalidade do VSM e por isso requerem um VSM no seguimento que forneça essas funcionalidades. No caso 3, as funções do VSM estão alocadas no veículo. No caso 4, as funcionalidades do VSM estão distribuídas entre o veículo e os elementos terrestres do sistema.

1.2.5.2 CCISM

O CCISM fornece uma função semelhante ao VSM, isto é, o encapsulamento dos dados CCI e qualquer tradução necessária para que seja compatível/interoperável

com as ligações físicas de comunicação entre o UCS e os sistemas C4I. O CCISM pode estar localizado junto do UCS ou com o terminal dos sistemas C4I. A arquitetura do UCS deve prever a integração do CCISM.

1.2.5.3 Operador

O sistema de operação de um UAV deve possuir um conjunto de parâmetros padrão, com os quais o operador possa operar/monitorizar todos os veículos que possam ser designados a esse mesmo operador. Apesar de não ser necessário que diferentes UCSs compatíveis com o STANAG 4586 possuam displays idênticos, é mandatário que o CUCS esteja de acordo com os requisitos do HCI.

1.2.5.4 CUCS

O UCS central deverá fornecer ao utilizador uma interface que permita a um operador qualificado conduzir todas as fases da missão de um UAV. Deverá também suportar os requisitos do DLI, CCI e HCI. O CUCS permite uma interface gráfica com o utilizador gerada computacionalmente a alta resolução que fornece ao operador a capacidade de controlar diferentes UAVs e *payloads*.

Dependendo do nível de interoperabilidade apropriado e a carga útil suportada no respetivo UAS, o CUCS deve:

- Fornecer a funcionalidade e capacidade de receber, processar e disseminar dados recebidos do UAV e carga útil;
- Permitir o planeamento de missões;
- Monitorizar e controlar a carga útil, o UAV e as ligações de dados;
- Possuir uma arquitetura *open software*[13] para que seja possível no futuro suportar novas capacidades de UAVs e carga útil;
- Fornecer ao operador do sistema as ferramentas computacionais necessárias relativamente a comunicações, planeamento, monitorização e execução da missão e receção, processamento e disseminação de dados;

[13] software que tem o seu código fonte disponível para modificações ou ajustes por qualquer pessoa, www.opensource.com

- Possuir a capacidade de aglomerar as funções VSM e CCISM necessárias.

1.2.5.5 CCI

O conjunto de mensagens padrão relativas á interface de comando e controlo e os protocolos associados foram selecionados para serem independentes dos sistemas C4I e para evitar adicionar requisitos nesses mesmos sistemas. O UCS e respetivo utilizador C4I do UAS deverão identificar conjuntamente a funcionalidade CCISM requerida, caso necessário, para fornecer ao UCS compatibilidade com o sistema C4I específico. As redes e comunicações usadas para suportar o CCI deverão ser compatíveis com NISP (*NATO Interoperability Standards and Profiles*). A intenção do NISP é providenciar um quadro global para as comunicações NATO que procuram a interoperabilidade entre sistemas de comando militar, controlo e comunicações. A estratégia do NISP foi desenvolvida com o objetivo de alcançar a interoperabilidade, maximizar a exploração de COTS[14] (Commercial Of-The Shelf) e ainda reduzir a proliferação de sistemas não padronizados. Todas as comunicações e sistemas de informação utilizados em âmbito NATO, no futuro, irão estar em conformidade com estes padrões.

1.2.5.6 HCI

O HCI estabelece os requisitos de exibição e *input* do utilizador que o CUCS deverá suportar. Estes estão categorizados em: requisitos gerais; configuração UCS; planeamento de missão; controlo do veículo aéreo; monitorização e controlo do operador; monitorização e controlo da carga útil; avisos, precauções e conselhos; gestão das comunicações.

1.2.5.7 Protocolos de Tecnologia de Informação e Comunicação do UCS

Todos os sistemas que englobem o veículo aéreo não tripulado e os sistemas C4I devem ser capazes de interoperar ao longo de uma rede encaminhada com múltiplas sub-redes, em que o UAS é visto como um elemento terminal (ou sub-rede terminal) de toda a rede. Esse mesmo facto possibilita que os componentes físicos do

[14] Produtos padrão fabricados em serie ao invés de produtos por medida

UAS e dos sistemas C4I estejam localizados em qualquer parte da rede. A troca eletrónica de informação por parte do UCS e dos sistemas C4I deverá estar de acordo com o NISP.

O NISP vai também servir como guia e componente técnico para suportar implementações de projeto e transições para a NNEC (*NATO Network Enabled Capability*). O NISP é aplicável a todos os sistemas de informação computorizados (CIS – Computer Information System) NATO e sistemas de gestão de informação (MIS – Management Information System), incluindo as suas interfaces internas e externas, que produzem, utilizam e trocam informação eletronicamente.

1.2.5.8 DLI

Ao estabelecer o conjunto de mensagens DLI foram considerados todos os requisitos do sistema de controlo e a grande variedade de veículos aéreos. O DLI deverá ser então a interface entre UAV/ligação de dados e o CUCS. Fornece ainda o formato e mensagens padrão que possibilita a comunicação entre a grande variedade de veículos e os CUCSs compatíveis com o STANG 4586.

No anexo B, apêndice B1 do STANAG 4586 encontra-se a definição da estrutura genérica de mensagens para a comunicação AV/CDT/CUCS. Este conjunto de mensagens inclui mensagens de controlo e de estado dos seguintes elementos:

- Veiculo Aéreo
- Carga Útil
- Ligação de Dados
- Precauções e avisos

O conjunto de mensagens contém os dados do UAV que são independentes da plataforma e da carga útil, tal que a interação padrão não requer uma mudança de modo a acomodar um veículo aéreo ou carga útil particular. Como complemento, o conjunto de mensagens ainda inclui a capacidade do DLI criar displays específicos do sistema pelo HCI.

O DLI é constituído por dois componentes principais. O primeiro componente é um conjunto genérico de mensagens desenvolvidos para serem independentes do veículo e carga útil e ainda suportar a funcionalidade do CUCS. O segundo componente é um mecanismo para suportar a comunicação da informação específica do veículo, do AV/CDT para o CUCS, de modo a exibir remotamente essa informação.

O CUCS irá gerar e compreender as mensagens comuns do veículo e carga útil usando o DLI. É possível concluir que o desenvolvimento de um conjunto de mensagens padrão e respetivo protocolo de comunicação entre o veículo/ligação de dados e a função CUCS é a chave para estabelecer uma arquitetura do CUCS interoperável.

O método primário de transferência de informação entre estes dois componentes é a comunicação por mensagens. Essa estrutura de mensagens tem como objetivo passar informação de estado e controlo do UAV entre o CUCS e o CDT sem criar dependências entre estes dois componentes. Esta metodologia permite ao CUCS usar os dados fornecidos pelo CDT e transmiti-los para uma localização independente.

O método secundário de comunicação entre estes dois componentes prende-se com a utilização de serviços. Os serviços permitem ao CDT afetar o HCI no CUCS, processo muito semelhante como um explorador web acede a uma página web para localmente apresentar dados que residem num servidor remoto.

1.2.6 Representação dos dados

A definição de uma representação padrão dos dados é de extrema importância para uma padronização de interfaces.

1.2.6.1 Ordenação

A ordenação dos bytes deverá ser em primeiro lugar o byte mais significativo (MSB – *Most Significant Byte*). Isto significa que o byte que se encontra mais á esquerda na sequência de números será o byte que representa o maior valor.

Números de vírgula flutuante deverão ser representados como definido no *standard* IEEE para a aritmética binaria de vírgulas flutuantes (Kahan, 1997).

1.2.6.2 Unidades

Devido à grande variedade possível de sistemas idealizados para o futuro e à natureza internacional da interoperabilidade planeada para o UCS, a filosofia de desenvolvimento das mensagens no DLI passa por usar unidades métricas sempre que possível. O DLI é um sistema interno de representação entre o CUCS e o VSM, e assim, qualquer conversão de representação necessária para a leitura e utilização dos dados pelo utilizador (e. g. metros por segundo para nós) serão realizados na interface do utilizador apropriada.

Todas as referências em terra deverão ser expressas no sistema latitude-longitude em concordância com o elipsoide WGS-84 (*World Geodetic System*) em radianos usando a medida angular binaria (BAM). Representações em outros sistemas, como UTM (*Universal Transverse Mercator coordinated system)*, devem ser convertidas aquando do seu uso. Todas as horas devem ser representadas em UTC (*Universal Time Coordinated*), em segundos desde 1 de Janeiro de 2000 usando 5 bytes em que o bit menos significativo representa 0.001 segundos. Em 2034 e os anos subsequentes, quando o valor máximo do quinto byte for excedido, a representação temporal deverá recomeçar do zero.

Todos os parâmetros angulares devem ser expressos em radianos. Azimutes devem ser contados a partir do norte verdadeiro no sentido dos ponteiros do relógio. A elevação é um parâmetro que deve ser medido tendo o horizonte local como referência sendo que a parte positiva será até ao zénite.

1.2.6.3 Pacotes de Dados

A intenção da utilização de pacotes de dados foi a de atingir o equilíbrio entre minimizar o cabeçalho das mensagens e a maximização da modularidade do conjunto de mensagens. Em complemento, tem ainda como intenção a categorização de dados em combinações logicas de mensagens como dados inerciais, dados de estado e dados

relativos ao vento quando se refere o estado do veículo. Dados de comando e de estado localizam-se em grupos de mensagens diferentes de modo a separar as mensagens *uplink* das mensagens *downlink*. As mensagens que normalmente requerem algum tipo de *acknowledgement*[15] estão também separadas das que não têm esse tipo de especificações.

1.2.6.4 Números de Identificação (ID)

Cada mensagem contém um campo que específica o número de identificação das duas unidades que se encontram em comunicação (e. g. veiculo e CUCS). Algumas delas podem ainda conter a identificação da ligação de dados. A identificação do destinatário pode ser configurada, em antecipação, de forma a ser um ID de um VSM de uma conexão futura de um veículo, carga útil ou ligação de dados. O propósito destes números é identificar de forma única cada entidade num sistema arbitrário em que seja possível combinar vários CUCS, veículos e ligações de dados, que poderão interagir com VSMs que potencialmente estarão a controlar nenhum ou vários veículos ao mesmo tempo.

Os números de identificação devem ser formados como números de 4 bytes. O primeiro (mais significativo) byte identifica o Owning ID desse CUCS ou veiculo. Este número identifica a que país pertence o equipamento e está estipulado no STANAG 4586. Na tabela 6 estão tabelados alguns exemplos.

Tabela 6: *Owning IDs*

Owning ID	**Número**
Bélgica	002
Bulgária	003
Canada	004

[15] Sinal que indica o reconhecimento ou correta receção de uma mensagem

Republica Checa	005
Dinamarca	006
Estónia	007
França	008
Alemanha	009
Grécia	010
Portugal	020
EUA	027

A cada veículo, ligação de dados ou CUCS compatível com o STANAG 4586 deverá ser designado um único número de identificação de sistema dentro da respetiva categoria. Entre estas três categorias, sistemas de diferentes tipos podem ter números de identificação idênticos mas, tal facto, deve ser evitado sempre que possível.

Os números de identificação serão apresentados como bytes individuais em hexadecimal, separados por colunas (e. g. 10:4E:F3:06). O número de Identificação FF:FF:FF:FF deve estar reservado como ID de *broadcast* a todos os veículos e o FF:00:00:00 como nulo. O ID 00:00:00:00 é reservado e indica que este campo não é aplicável.

1.2.6.5 Dados de Informação da Mensagem (Wrapper)

Cada mensagem deverá utilizar a estrutura definida no anexo B do STANAG 4586 e representada na figura 19. O cabeçalho contém informação que permite ao *software* de manuseamento de mensagens gerir a transmissão e distribuição dessas mensagens para as entidades apropriadas. A última informação desta estrutura,

designada por rodapé (i. e. footer) contém a informação de *checksum* (soma de verificação) que é utilizada para identificar erros de transmissão.

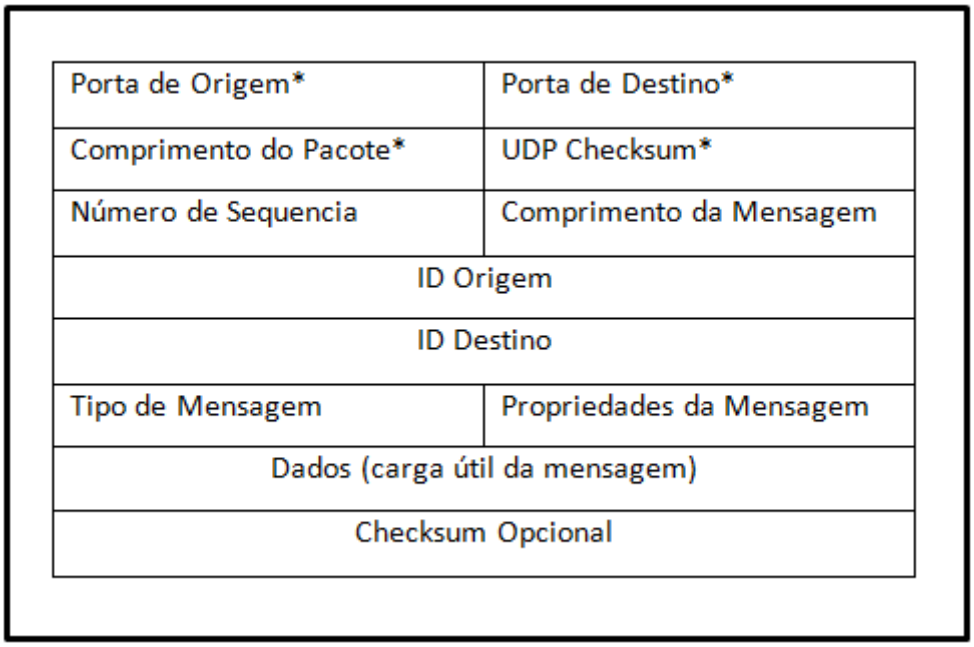

Figura 19: Estrutura dos dados de Informação da Mensagem (Wrapper)

A não ser que seja referido de outra forma, todos os cabeçalhos devem ser representados por 2 bytes, ou seja, 16 bits.

Todos os campos que apresentem um "*" indicam o cabeçalho UDP (User Datagram Protocol) que é ilustrado na figura acima de modo a que todo o cabeçalho da mensagem seja visível incluindo o cabeçalho UDP. Como as mensagens funcionam a nível da camada de aplicação, o wrapper começa no número de sequência da mensagem e termina no *checksum* opcional.

O UDP é um protocolo simples que dentro da pilha de protocolos de um sistema se situa na camada de transporte. Este permite que uma aplicação escreva um datagrama (pacote de dados) embutido seja em IPv4[16] ou IPv6[16] e posteriormente enviado para o destino. Este mesmo protocolo não é fiável pois não garante que o

[16] Protocolo de comunicação utilizado entre todas as máquinas em rede para encaminhamento de dados.

mesmo chegue ao destino. Caso sejam necessárias medidas utilizadas como garantias, é então necessário implementar uma série de estruturas de controlo como as apresentadas a estrutura de mensagens implementadas neste STANAG, e. g. *checksum* (Postel, 1980).

1.2.6.5.1 Número de Sequencia

O propósito do número de sequência é o de providenciar meios de segmentar dados de uma única mensagem em sequências de blocos com um comprimento máximo definido. Este campo não é utilizado atualmente e deve constar "-1".

1.2.6.5.2 Comprimento da Mensagem

O comprimento da mensagem deverá ser um campo com 16 bits (2 bytes) com o número de bytes que constam nos dados da mensagem, entre 1 e 528, sem cabeçalho.

O protocolo UDP relativo ao IPv4 garante um datagrama mínimo de 576 bytes. Subtraindo o cabeçalho do IPv4 que são 20 bytes, o cabeçalho UDP de 8 bytes e o cabeçalho do wrapper de 20 bytes resulta então o comprimento máximo dos dados da mensagem de 526 bytes.

1.2.6.5.3 ID Origem e ID Destino

Tanto o ID de origem como o de destino designam o campo onde se introduz o número de identificação do elemento UAS que enviou ou recebe a mensagem, respetivamente.

Cada mensagem deve conter os campos que especificam os números de identificação para o veículo e CUCS que estão em comunicação. Algumas mensagens também contêm o ID da ligação de dados e do VSM. O ID do VSM é fornecido nas mensagens que poderão necessitar de ser transmitidas entre o CUCS e o VSM á priori em relação à conexão com o veículo, payload, ou ligação de dados. O propósito destes números é identificar qualquer entidade num sistema arbitrário que combine múltiplos CUCS, veículos e ligações que potencialmente poderão interagir com vários

VSMs, estes que por sua vez podem controlar nenhum ou vários veículos em simultâneo.

O ID de origem e destino são formados por 4 bytes (32 bits) distribuídos como indicado na figura 20.

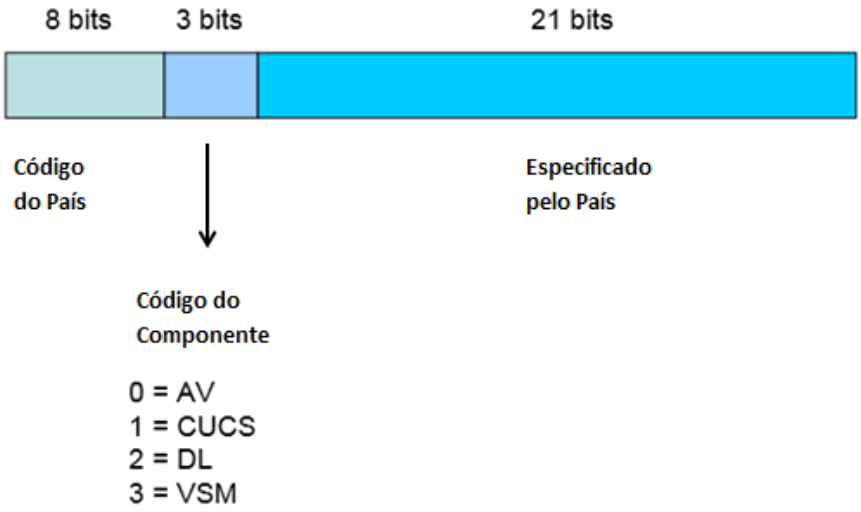

Figura 20: Número de identificação tanto de origem como de destino

O byte mais significativo é o código do país NATO como definido na tabela em anexo B.

Os 3 bits seguintes identificam o componente como demonstrado na figura 20. Os 21 bits seguintes são definidos por cada país consoantes os seus protocolos e procedimentos específicos.

1.2.6.5.4 Tipo de Mensagem

Este campo é utilizado para especificar qual o número de identificação da mensagem ao qual estão alocados 2 bytes.

1.2.6.5.5 Propriedades da Mensagem

Este campo ocupa também ele o espaço de 2 bytes, ou seja 16 bits. Estes 16 bits são divididos em 4 campos (Figura 21).

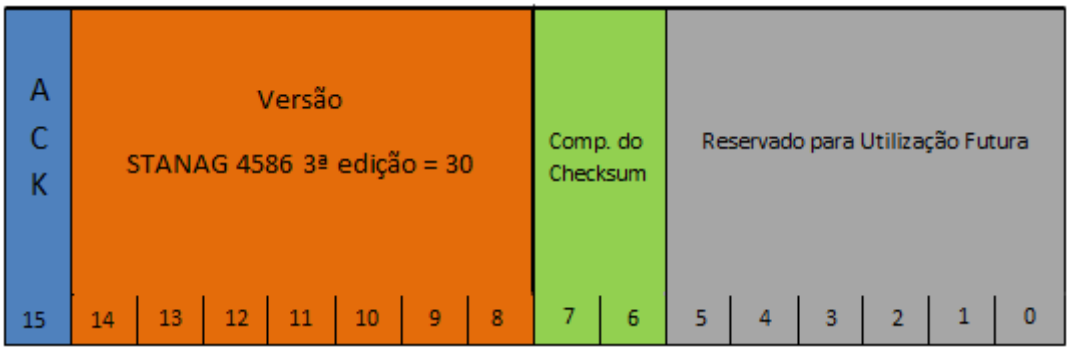

Figura 21: Formatação das propriedades da mensagem

O bit mais significativo indica se aquando a receção da mensagem deverá ser enviado um acknowledgement ("1" se sim, "0" se não). Os sete bits seguintes identificam o número da versão do IDD (Interface Definition Document) do qual a estrutura da mensagem foi definida. No caso do STANAG 4586 3ª edição o ID é o "30". Os dois bits seguintes indicam o comprimento do *checksum* como indicado na tabela 7.

Tabela 7: Comprimento do *Checksum*

Identificação	**Comprimento**
00 Binário = "0"	Sem *checksum*
01 Binário = "1"	2 Bytes
10 Binário = "2"	4 Bytes

1.2.6.5.6 *Checksum* Opcional

Para além do *checksum* UDP, pode ser introduzido um checksum opcional no fim da mensagem de modo a que sejam identificados erros de transmissão. O uso e comprimento deste mesmo *checksum* são especificados nas propriedades da mensagem como descrito anteriormente. O *checksum* é simplesmente calculado como adição binária de todos os dados contidos na mensagem excluindo o mesmo e truncado em 2 ou 4 bytes que é utilizado para verificar a integridade dos dados transmitidos.

1.3 MAVLink

O Micro Aerial Vehicle Communication (MAVLink) é um protocolo de comunicação que engloba uma biblioteca de mensagens especialmente vocacionadas para veículos aéreos não tripulados de pequenas dimensões. O mesmo foi desenvolvido por Lorenz Meier na ETH Zurich e consiste basicamente numa biblioteca de comunicações leves entre veículos aéreos micro não tripulados e GCSs (Heng, 2011).

O formato das mensagens MAVLink codifica as estruturas de dados em pacotes de dados mais eficientes que utilizam codificação binária, em vez de código ASCII (*American Standard Code for Information Interchange*), possibilitando assim uma mais rápida e integra transferência de dados (Coombes, 2012).

O MAVLink serializa estruturas em C[17] para os canais série e pode ser usado em qualquer *modem* de rádio. As definições de mensagens são criadas em XML (*eXtensible Markup Language*) e seguidamente convertidos em estruturas arquivadas de cabeçalhos C. O MAVLink é atualmente utilizado também no sistema operativo Linux, ligações de comunicação terrestre e em vários pacotes de *software* (e. g. ROS e APM Planner) (Lim, 2012, Meier, 2011, Lee, 2013).

1.3.1 Estrutura Funcional

A arquitetura funcional do MAVLink funciona como um mecanismo que age tendo uma vasta transmissão não filtrada de mensagens que cada componente ou subsistema está possibilitado a receber e ler/interpretar. Complementariamente, cada componente que se insira neste âmbito usufrui da possibilidade de fazer broadcast de mensagens (Meier, 2011, Marty, 2013).

Este protocolo é suportado por uma grande variedade de pilotos automáticos e *software* de estações de controlo terrestre (e. g. Ardupilot, Parrot AR, QGroundControl, APM Planner). Com a utilização deste protocolo e tendo em conta o

[17] linguagem de programação estruturada e padronizada pela ISO, *International Standard Organization*

firmware[18] da carga útil, é perfeitamente possível a interação com uma grande variedade de pilotos automáticos existentes (Coombes, 2012, Shilov, 2014).

1.3.2 Formato da Mensagem

Todas as mensagens definidas neste protocolo têm um comprimento máximo de 17 bytes. As mensagens variam com o ID de cada um. Mensagens comuns a todos os pilotos automáticos incluem heartbeat, comando e campos de gestão de waypoints (Coombes, 2012, Heng, 2011).

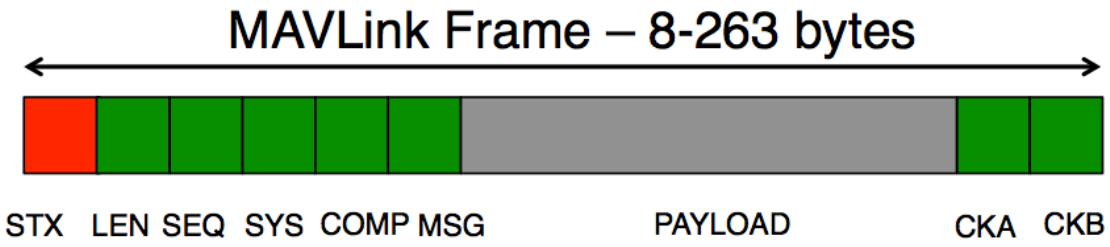

Figura 22: Estrutura mensagem MAVLink

A mensagem pode ser dividida em três partes constituintes: o cabeçalho; *payload*; *checksum,* como pode ser verificado na Figura 22 (Reker, 2015).

1.3.2.1 Cabeçalho

O cabeçalho é constituído por seis bytes, cada um com o seu significado. As seis partes constituintes deste cabeçalho são: *header*; comprimento da mensagem; número de sequência; ID do sistema de origem; ID do componente do sistema de origem; ID da mensagem (Lim, 2012).

1.3.2.1.1 Header

Este primeiro byte coincidente em cada mensagem é o sinal de início de pacote e é utilizado para sincronizar o início da mensagem codificada. Atualmente existem duas versões deste protocolo (versão 1.0 e versão 0.9). A principal diferença destas

[18] conjunto de instruções operacionais programadas diretamente no hardware de um equipamento eletrónico

duas versões é o valor que representa este byte que difere de uma versão para a outra da seguinte forma:

Tabela 8: Inicio de pacote consoante a versão do MAVLink

Índice do Byte	Conteúdo	Versão	Valor
0	Header	0.9	0x55
0	Header	1.0	0xFE

Os valores tabelados acima, representados em hexadecimal, serão os valores a filtrar de modo a localizarmos o início de uma mensagem e possibilitar o recetor a sincronizar a mensagem.

1.3.2.1.2 Comprimento da Mensagem

Este parâmetro é de alguma importância pois indica o comprimento da carga útil da mensagem que no máximo é 9 bytes. Para este parâmetro está reservado um byte que em hexadecimal tem valor máximo 0xFF. Podemos então concluir que o comprimento máximo da mensagem é de 255 em decimal.

1.3.2.1.3 Número de Sequência

O número de sequência é utilizado pelo MAVLink para cada pacote como um mecanismo de segurança para detetar a perda de pacotes. Depois de verificado, caso a taxa de perda de pacotes seja significativa, o componente irá inicializar novamente a tarefa ou missão e reduzir a sua capacidade operacional (Marty, 2013).

1.3.2.1.4 ID do Sistema de Origem

Este parâmetro representa o sistema de origem da mensagem, ou seja, o sistema que enviou a mensagem (e. g. Mission Planner). Possibilita também a diferenciação de todos os MAVs existentes numa rede.

1.3.2.1.5 ID do Componente do Sistema de Origem

Neste caso, representa qualquer subsistema localizado no sistema principal referido no parâmetro anterior no qual a mensagem foi originada.

1.3.2.1.6 ID da Mensagem

O ID da mensagem representa a categoria dos dados que se seguem. Os dados transmitidos na carga útil variam consoante o ID da mensagem pois este indica o assunto da mensagem. Depois do ID ser interpretado, os dados uteis da mensagem vão ser colocados num pacote que será correlacionado com a estrutura apropriada tendo em conta o tipo de dados (o que representam). Um dos exemplos será a atitude que engloba o pitch, roll e o yaw ou então o *heartbeat*. Na tabela disponível no anexo A é possível visualizar algumas das mensagens MAVLink com o seu respetivo ID. Não são apresentadas todas devido a ser um campo com elevada dimensão.

1.3.2.2 Payload

Nesta parte da mensagem é onde estão incutidos os dados, portanto a parte mais importante da mensagem. Estes dados é o que vamos utilizar, tendo em conta o seu ID, depois de ser descodificados e interpretados. O payload ou carga útil é o cerne da mensagem que tem reservado 9 bytes para a sua utilização.

1.3.2.3 Checksum

Cada mensagem utiliza um CRC (Cyclic Redundacy Check) duplo, representados na figura como ChecksumA e ChecksumB, de modo a possibilitar uma probabilidade de erro mais reduzida do que tendo só uma verificação de transmissão. No *checksum* não é incluído o *header*. Caso seja recebida uma mensagem com o *checksum* errado, essa mensagem será imediatamente descartada.

O CRC é uma verificação de redundância cíclica que funciona como um método para identificação de erros. Representa um género de código para cada pacote de dados que ao ser transmitido é verificado no recetor de modo a assegurar a não existência de erros nos dados transmitidos.

2 Metodologia de Investigação

A figura 23 ilustra o esquema de blocos do modo como foi estruturada a metodologia de investigação desta dissertação.

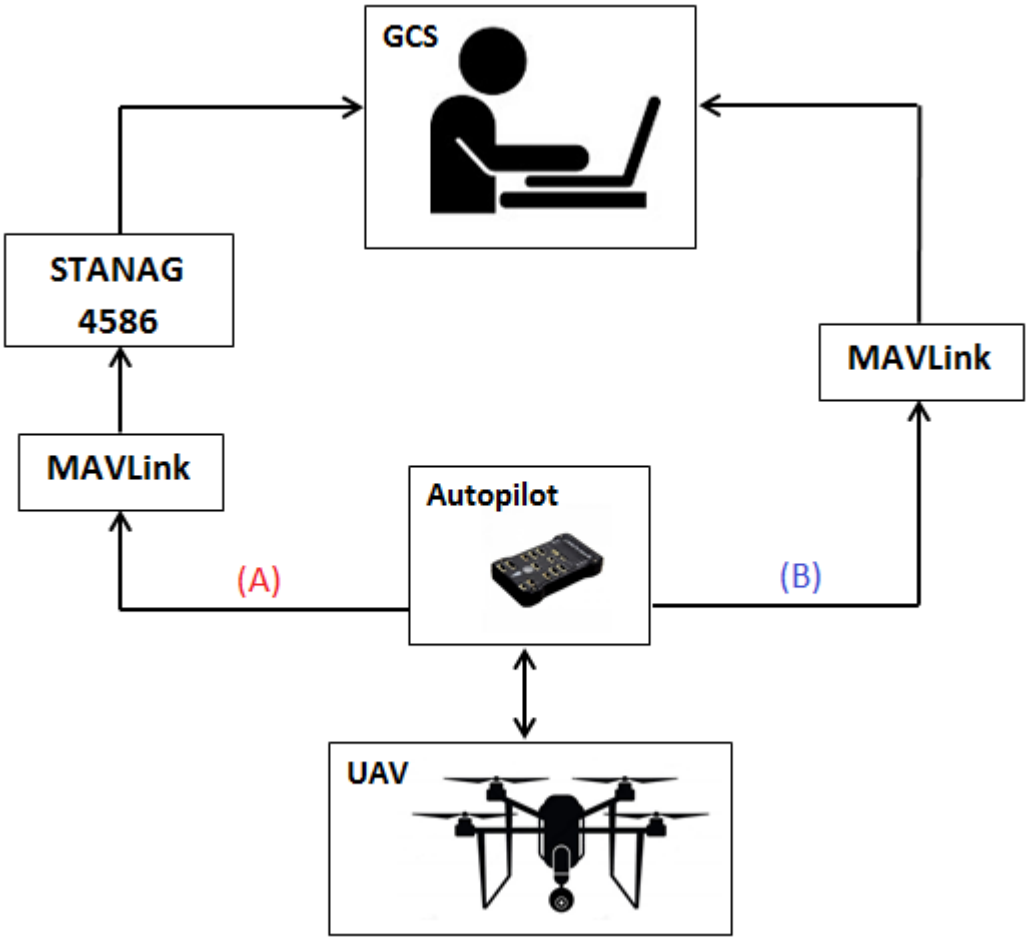

Figura 23: Metodologia de investigação adotada

O bloco UAV apesar de no esquema se encontrar separado do bloco Autopilot, este abrange-o. O dispositivo de piloto automático está inserido no veículo, no entanto, existem interações importantes entre estes. O UAV, como referido anteriormente, engloba toda a plataforma, incluindo os sensores. Estes, por sua vez, são indispensáveis para transmitir informação ao piloto automático. À luz destes factos, adotei a separação destes dois blocos.

O bloco do piloto automático na realidade encontra-se fisicamente conectado e incutido no UAV. No entanto, nesta investigação, o piloto automático foi utilizado de

forma independente pois a sua interação com o UAV não é o objeto de estudo que se pretende.

A partir do bloco do piloto automático existem dois caminhos que podem ser seguidos.

O caminho designado como caminho "A" é ilustrativo da conversão de mensagens que, provenientes do Pixhawk, circulam primeiramente no formato de mensagens MAVLink, sendo seguidamente convertidas no formato de mensagens STANAG 4586. Por fim as mensagens irão chegar a uma GCS que irá monitorizar os dados recebidos.

O caminho designado como caminho "B" simboliza a transmissão natural de um sistema aéreo não tripulado em que as mensagens são transmitidas diretamente do piloto automático com destino a uma GCS que irá monitorizar esses dados.

O bloco GCS simboliza uma estação de controlo terrestre que na realidade será um *software* computorizado em que será feita a conversão de um protocolo de comunicação para o outro. Este *software* irá também monitorizar os dados recebidos do piloto automático nos dois formatos de mensagem distintos e verificar se são correspondentes. Este mesmo programa possibilitará a análise de tempo em relação ao *delay* proveniente da conversão efetuada no caminho "A" comparativamente com o tempo de transmissão do caminho "B".

Sendo a linguagem de programação em que me sentia mais confortável, escolhi a linguagem de programação C#. Por acrescento, existe uma aplicação *open source* que funciona como GCS e o seu código está disponível nesta mesma linguagem (Mission Planner).

3 Arquitetura do Sistema

Neste capítulo será descrito o trabalho desenvolvido e está dividido em 2 subcapítulos, Hardware e Software. No subcapítulo Hardware serão descritos todos os componentes utilizados no desenvolvimento do projeto.

3.1 *Hardware*

O componente principal integrado no desenvolvimento deste projeto é o piloto automático Pixhawk (Figura 24).

Figura 24: Componente do piloto automático Pixhawk

O Pixhawk é um piloto automático de alto desempenho adequado para aeronaves de asa fixa, aeronaves de rotores múltiplos, helicópteros, carros, barcos e qualquer outra plataforma robótica com capacidade de se movimentar. Foi desenvolvido de modo a suplantar as necessidades de pesquisa, amadora ou de indústria e combina as funcionalidades de PX4FMU e PX4IO (Figura 25). PX4FMU é a unidade de gestão de voo onde está inserido o sensor digital de movimento a 3 eixos. O PX4IO é o módulo Input/Output que incorpora 4 relés, um servomotor de redundância e um sistema de substituição manual.

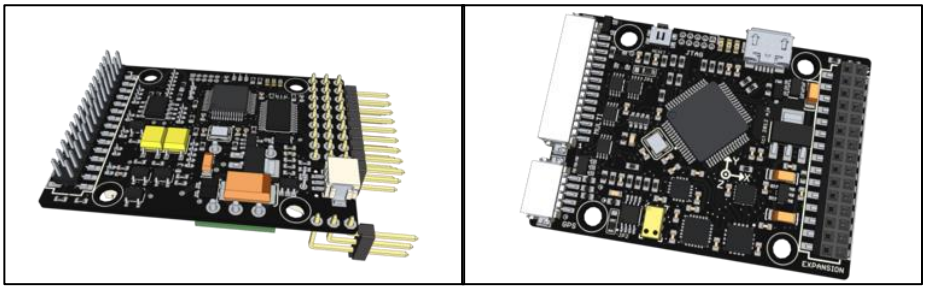

Figura 25: Partes constituintes do PIXHAWK

O Pixhawk é um projeto independente (hardware aberto) com o objetivo de fornecer um *hardware* piloto automático de alto rendimento para propósitos académicos e comunidades industriais em baixos custos e alta disponibilidade.

O processador do Pixhawk é de 32 bits, funciona a 168 MHz e incorpora 256 KB de RAM. Como sensores apresenta um giroscópio, um acelerómetro/magnetómetro, um acelerómetro/giroscópio de 3 eixos e um barómetro. Relativamente a interfaces contém várias sendo que as mais relevantes serão as 5 portas serie UART (transmissor/recetor assíncrono universal) e uma entrada compatível com comunicações satélite.

3.2 Software

O processo de desenvolvimento do *software* envolveu vários processos que serão aqui explicados. Esses mesmos processos estão ilustrados na figura 26.

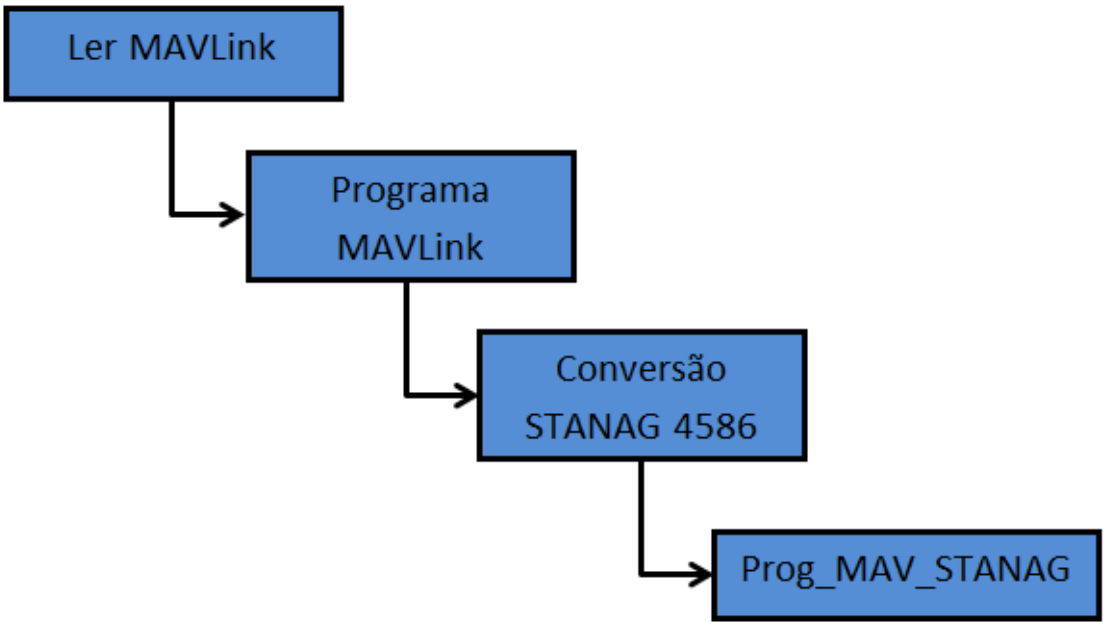

Figura 26: Processos de desenvolvimento do software

O esquema de blocos ilustrado na figura 26 identifica a sequência de *softwares* desenvolvidos no desenvolvimento desta dissertação. Cada bloco irá ser descrito nas páginas seguintes.

3.2.1 Ler MAVLink

Tendo o componente piloto automático conectado ao computador via um cabo micro USB (Universal Serial Bus) foi necessário desenvolver um *software* que lê-se as mensagens transmitidas pelo componente de modo a que fosse percetível. Os dados são transmitidos em hexadecimal e no formato anteriormente referido no capítulo MAVLink. A figura 27 diz respeito às funções inseridas neste programa.

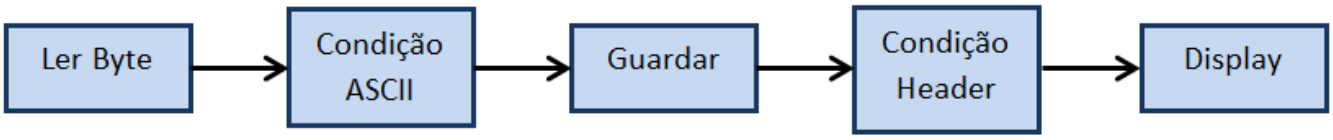

Figura 27: Esquema de blocos do programa "Ler MAVLink"

O primeiro módulo diz respeito à função de ler o byte recebido. Cada byte recebido irá seguidamente passar pela condição ASCII em que o seu valor em hexadecimal terá de ser maior ou igual a 0x20 e menor ou igual que 0x7F. Estas condições foram impostas pois os valores compreendidos neste intervalo seriam os que queria ler, tendo em conta a tabela ASCII (Figura 28[19]). De modo a não cometer erros desnecessários ainda introduzi duas vertentes nessa condição em que o caracter poderia ser igual a '/n' ou a 0x1B. O caracter '/n' simboliza um parágrafo.

[19] http://www.asciitable.com/, Março 2016

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Figura 28: Tabela ASCII

Sempre que o que o valor hexadecimal recebido é autenticado na condição anterior fica guardado num *buffer*. Este ciclo continua até que o valor recebido não cumpra a condição ASCII.

No módulo seguinte verifica-se onde se encontram os valores de início de uma mensagem MAVLink que tanto pode ser 0x55 ou 0xFE tendo em conta a versão, como já referido neste documento.

Por ultimo, o programa mostra a mensagem recebida a partir do valor de cabeçalho da mensagem como se mostra na figura 29.

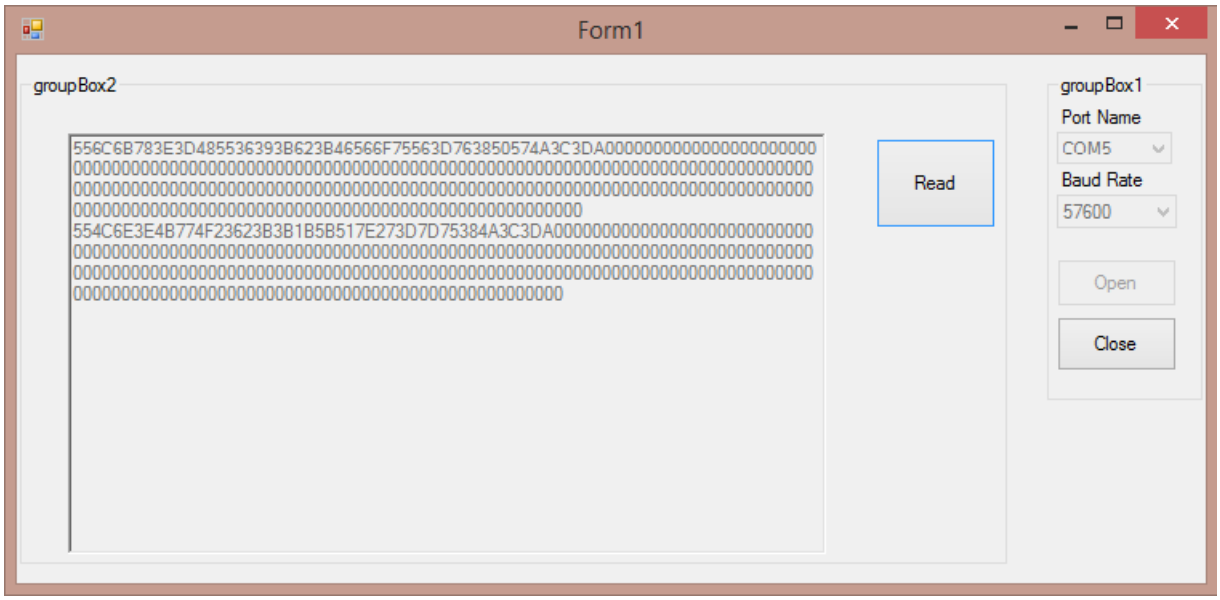

Figura 29: Output do programa

Nesta altura já se notava que algo não estava em conformidade. Tendo eu a informação que a versão MAVLink que o componente utilizava seria a versão mais recente (versão 1.0) o início da mensagem deveria ser 0xFE e não 0x55. Caso o início da mensagem fosse 0x55, o segundo byte seria o comprimento da mensagem, o que no caso da primeira mensagem da figura acima é 0x6C (108 decimal). Os restantes dados da mensagem não estão em conformidade com esse comprimento.

Identifiquei dois possíveis problemas:

- A taxa de transmissão (baud rate) não era a adequada de modo a receber todas as mensagens ou todos os bytes;
- O componente estar a transmitir dados aleatórios;

3.2.2 Correção de Erros

Com recurso a vários testes e experimentação, chegou-se à seguinte conclusão, relativamente aos possíveis erros identificados anteriormente.

3.2.2.1 Taxa de transmissão

A taxa de transmissão utilizada era a correta e por isso fixa-se nos 57600 bps. Poderia também fixar-se nos 115200 bps, o que poderia causar erros, pois quanto mais baixa a taxa de transmissão, menor a probabilidade de erros. O problema identificado consistia na forma como estava a ser utilizada e não o seu valor. Tendo em conta que só se está a utilizar uma CPU (Unidade Central de Processamento), será então necessário utilizar um método do sistema (System.Threading) de modo a que um processo se divida em várias tarefas que irão ser executadas concorrencialmente. Cada thread é processada de forma aparentemente simultânea pois a mudança de uma para outra ocorre de forma tão espontânea que parece realizar-se paralelamente.

3.2.2.2 Transmissão de Dados

Os dados que estavam a ser transmitidos no programa eram mensagens aleatórias. Percebi, a certo ponto, que seria necessário enviar uma mensagem em formato MAVLink para o componente a requerer a transmissão de dados. Seguidamente o programa receberia os dados, no entanto, teria de se desenvolver um processo mais otimizado que o anterior de modo a ler as mensagens.

3.2.3 Programa MAVLink

Com o programa Ler MAVLink concretizado, foi possível aprender com os erros cometidos e resolvê-los. Foi aberta ainda a possibilidade de otimização da aplicação. O esquema da figura 30 reflete o esquema de blocos do programa MAVLink. O objetivo é receber os dados de atitude do dispositivo (*pitch*, *roll* e *yaw*).

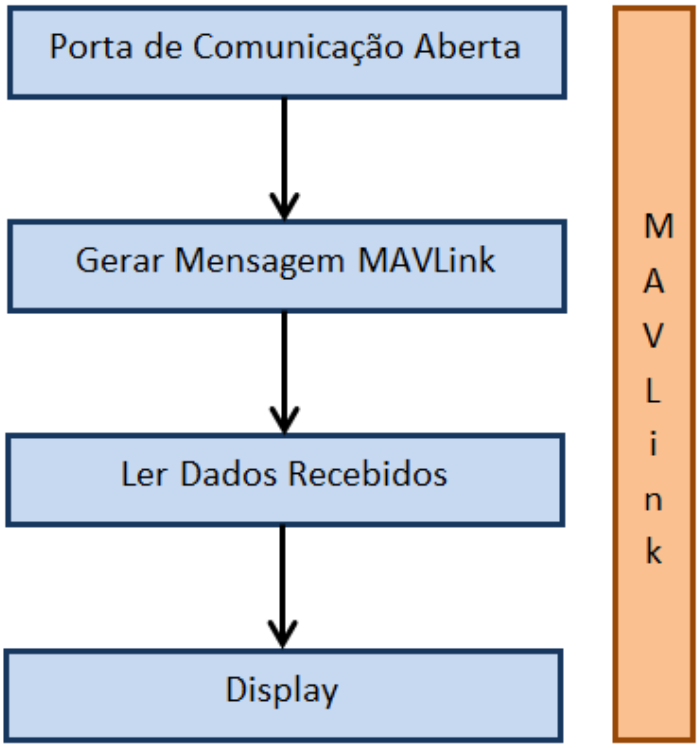

Figura 30: Esquema blocos do programa MAVLink

3.2.3.1 Porta de Comunicação Aberta

O primeiro bloco do esquema anterior é idêntico ao do programa Ler MAVLink. O programa verifica as portas de comunicação (COM PORTS) disponíveis, e abre a porta que o utilizador selecionar com a taxa de transmissão que for também selecionada.

A grande diferença é que neste caso, o programa lê os dados recebidos com a função "SerialPort.ReadTimeout". Esta função define o número de milissegundos em que ocorre o tempo limite enquanto a operação de leitura não terminar. Foram definidos 2000 milissegundos, ou seja, 2 segundos.

3.2.3.2 Biblioteca MAVLink

O bloco que se encontra na vertical no esquema anterior simboliza a biblioteca MAVLink, a qual, foi inserida no programa. Esta biblioteca inclui todos os possíveis comprimentos das mensagens, informação das mensagens tendo em conta o seu ID, ID das mensagens e nomes associados. A tabela 9 demonstra os dados inseridos na

biblioteca da mensagem MAVLink associada ao heartbeat (ID 0x0). O heartbeat é a mensagem mais importante. Como o nome indica, o batimento cardíaco serve para verificar se o sistema continua a funcionar e conectado. A mensagem é enviada a cada segundo para esse mesmo propósito. Esta mensagem, na biblioteca, dá pelo nome de "mavlink_heartbeat_t". De referir que o formato disponibilizado seguidamente apenas diz respeito à carga útil da mensagem e não ao cabeçalho.

Tabela 9: Estruturação da mensagem *heartbeat*

Índice	**Designação**
0	Int 32 custom_mode
1	Byte type
2	Byte autopilot
3	Byte base_mode
4	Byte system_status
5	Byte mavlink_version

O campo de índice 0 é o campo onde é inserido um bit para uso nas "flags" específicas do piloto automático. As "flags" são sinais que indicam o estado específico de uma conexão, por exemplo.

O campo de índice 1 é um byte que indica o tipo de sistema a que o piloto automático está associado como indicado na tabela 10 onde vão ser indicados alguns exemplos.

Tabela 10: Número de identificação dos vários tipos de sistema

Tipo de Sistema	**Número de identificação**
Genérico	0

Asa Fixa	1
Quadrotor	2
Helicóptero	4
Antenna Tracker	5
GCS	6
Free Balloon	8
Hexarotor	13
Octorotor	14
Tricopter	15
Flapping Wing	16
VTOL Duorotor	19
VTOL Quadrotor	20
Gimbal	26

O campo de índice 2 indica o tipo de piloto automático que corresponde a uma tabela do mesmo formato da anterior que inclui os vários tipos de pilotos automáticos existentes. Para o Pixhawk o valor é fixado no número 1.

O campo de índice 3 está reservado para uma “flag” associada ao modo de operação do piloto automático.

O campo de índice 4 indica o estado do sistema e esta diretamente associado ao modo de voo em que o sistema se encontra.

O último campo desta mensagem indica a versão do MAVLink que é automaticamente inserido pela biblioteca MAVLink.

Esta biblioteca inclui todas as mensagens bem como todas possibilidades para cada campo dessa mesma mensagem.

3.2.3.3 Gerar Mensagem MAVLink

É então necessário gerar uma mensagem no formato MAVLink que seja enviada para o piloto automático de modo que este a reconheça. Essa mensagem irá pedir ao piloto automático os dados pretendidos, a atitude do dispositivo.

Para esse efeito foi construída uma função intitulada "mavlink.GenerateMAVLinkPacket" onde é necessário inserir o nome do tipo de mensagem que queremos gerar, o ritmo de transmissão da mensagem que se está a requerer, e o componente e sistema destino. Com esses dados, a função gera a mensagem com os dados providenciados pela biblioteca MAVLink e faz o cálculo do CRC de modo a que a mensagem não seja descartada aquando da sua chegada ao destino. Seguidamente a mensagem é enviada.

A mensagem gerada é designada por "mavlink_request_data_stream_t", que, como o nome indica, pede um *stream* de dados ao componente alvo. Na tabela 11 está demonstrada a esquematização da mensagem.

Tabela 11: Estruturação da mensagem de requisição do stream de dados

Índice	**Designação**	**Valor (Hexadecimal)**
0	*Header*	FE
1	Comprimento dos dados	06
2	Número de Sequência	00
3	ID do sistema	FF
4	ID do Componente	00
5	Tipo de Mensagem	42
6	Ritmo de transmissão	02

7	Ritmo de transmissão	00
8	Sistema Destino	01
9	Componente Destino	01
10	ID do *stream*	00
11	*Start-Stop*	01
12	*ChecksumA*	5A
13	*ChecksumB*	2A

Os primeiros 3 campos já foram explanados neste documento. O ID do sistema é referente à estação de controlo terrestre (“MYGCS”). O ID do componente é nulo e faz referência a qualquer componente. Este campo não é significativo neste caso.

O tipo de mensagem é o valor referente à mensagem que se refere. O ritmo de transmissão é um número de 16 bits e por isso é convertido em 2 bytes. Designa 2 segundos entre cada mensagem requerida. Como o único sistema e componente conectado é o piloto automático, os campos de índice 8 e 9 fixam-se no valor 1. O ID do *stream* localiza-se na biblioteca MAVLink e o valor nulo representa todos os dados. O campo de índice 11 se o valor for “1” significa inicialização da transmissão dos dados, se o valor for “0” significa o término da transmissão. Os *checksum* são calculados tendo em conta os dados da mensagem anterior.

3.2.3.4 Ler Dados Recebidos

O processo de leitura de dados, para além de estar muito associado à biblioteca MAVLink, incorpora alguns ciclos e condições. Esses ciclos e condições mais importantes irão ser descritos tendo em conta o esquema da figura 31.

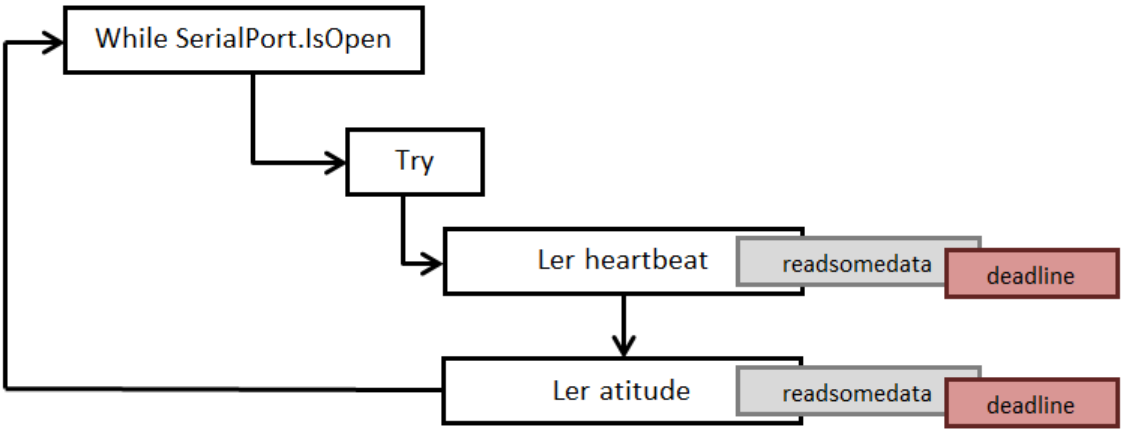

Figura 31: Esquema blocos do processo de leitura da mensagem

O ciclo intitulado "While SerialPort.IsOpen" tem como objetivo verificar se a porta de comunicação se encontra aberta, e em caso afirmativo, irá executar as linhas de código seguintes.

O bloco "Try" incorpora as linhas de código seguintes e é utilizado de modo a que as linhas de código seguintes sejam executadas até que haja uma exceção ou não haja mais linhas de código para serem executadas. Neste caso, o programa irá voltar ao ciclo "While SerialPort.IsOpen".

O bloco "Ler heartbeat" tenta ler a mensagem respetiva utilizando a função "readsomedata". Esta função utiliza outras funções da biblioteca MAVLink de modo a guardar os dados recebidos na mensagem e alocá-los em espaços diferentes num *buffer* de modo a que fique organizado e mais facilmente acessível. Existe ainda uma condição denominada "deadline" que adiciona o tempo "timeout" já referido anteriormente, à hora atual a que o método foi invocado. Esta condição está inserida num ciclo "while", pois enquanto o tempo atual não for superior ao "deadline" o programa continua a executar o "readsomedata" e assim a ler e alocar os dados da mensagem.

O formato da mensagem heartbeat está descrito em 3.2.3.2.

O bloco “Ler atitude” é semelhante ao bloco anterior a nível de processos, funções e ciclos. Difere unicamente no formato da mensagem associada à biblioteca MAVLink.

A mensagem que disponibiliza os dados de atitude do componente denomina-se “mavlink_attitude_t” (ID 0x1E) e tem o formato descrito na tabela 12. De referir que o formato disponibilizado na tabela 12 apenas diz respeito à carga útil da mensagem e não ao cabeçalho.

Tabela 12: Campos da mensagem atitude

Índice	Nome do Campo	Tipo	Descrição
0	Time_boot_ms	Int32	Timestamp (milissegundos desde o inicio do sistema)
1	Roll	Float	Roll angle (rad, -pi…+pi)
2	Pitch	Float	Pitch angle (rad, -pi…+pi)
3	Yaw	Float	Yaw angle (rad, -pi…+pi)
4	Rollspeed	Float	Roll angular speed (rad/s)
5	Pitchspeed	Float	Pitch angular speed (rad/s)
6	Yawspeed	Float	Yaw angular speed (rad/s)

O “time_boot_ms” representa o tempo em milissegundos desde o início do componente. É do tipo 32-bits que significa que pode oscilar entre 2^{32} valores. O *roll*, *pitch* e *yaw* representam respetivamente o ângulo de rotação, cabeceio e azimute. São do tipo *float* já explicado neste documento e estão expressos em radianos (variam de –

π a π). Os três campos seguintes representam as velocidades de rotação de cada variável no seu eixo respetivo e estão expressas em radianos por segundo.

3.2.3.5 Display

Como o objetivo é comparar os dados recebidos em MAVLink e esses mesmos dados depois de convertidos para STANAG 4586, é então necessário poder visualizá-los. A imagem na figura 32 demonstra como os dados são exibidos na aplicação Visual Studio.

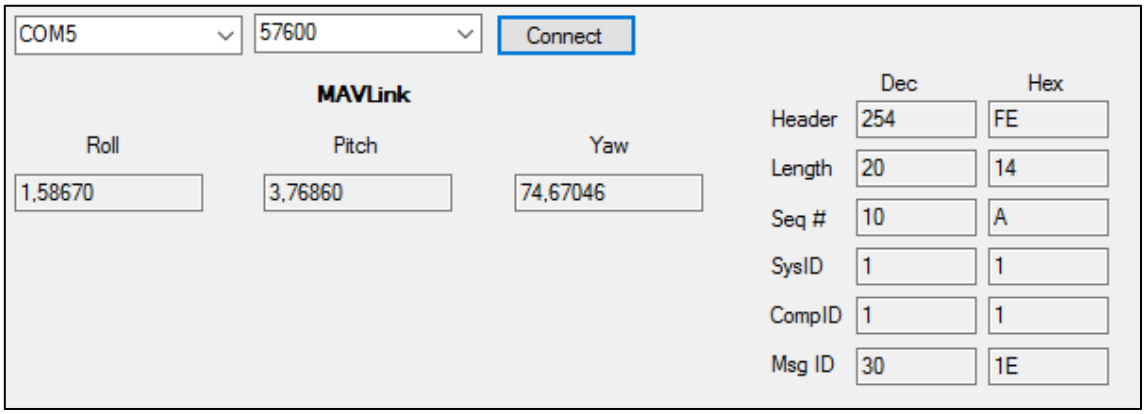

Figura 32: Output da primeira fase do programa

Na figura 32 encontra-se uma visualização facilmente intuitiva e interpretável dos dados. Primeiramente encontram-se duas "comboboxes" onde se seleciona a porta de comunicação em que o dispositivo se encontra (se unicamente estiver uma ligada, as opções disponíveis serão limitadas a uma) e o ritmo de transmissão da mesma. O botão "*Connect*" serve exatamente para estabelecer a comunicação com o dispositivo.

3.2.4 Conversão STANAG 4586

No final do programa MAVLink possuí-a todos os dados necessários relativamente à mensagem MAVlink e dados requeridos necessários à conversão apropriada para o formato STANAG 4586.

Como referido anteriormente, o formato STANAG 4586 apresenta um wrapper dividido em vários blocos. Esses blocos são: número de sequência, comprimento da

mensagem, ID de origem, ID de destino, tipo de mensagem, propriedades da mensagem, dados e checksum opcional.

Todos estes blocos estão explicados neste documento. Assim, é necessário explicar e descrever os dados inseridos em cada bloco no caso da mensagem que iremos enviar.

A mensagem que se adequa e em que nela estão inseridos os dados atitude é a mensagem "#40000: Inertial States".

Esta mensagem é enviada pelo VSM ao CUCS de modo a fornecer os dados inerciais tendo em conta o estado do veículo.

3.2.4.1 Comprimento da Mensagem

Como referido anteriormente, este campo contempla o comprimento dos dados da mensagem, ou seja, a carga útil. Para esta mensagem e como pode ser verificado na tabela 13 o comprimento é de 25 ou 0x19. O campo para o comprimento da mensagem contempla dois bytes e irá aparecer o seguinte: "00 19", tendo em conta o comprimento da mensagem.

3.2.4.2 ID de Origem

Como já referido neste documento, o ID de origem está dividido em três partes. O código do país (8bits), o código do componente (3bits) e os seguintes 21 bits estão reservados para uso específico do país utilizador. Tendo em conta o anexo B onde se encontram os códigos de cada país, o código de Portugal é 20 (decimal). Este código tem de ser convertido em hexadecimal e assim o valor utilizado é 0x14. O componente de onde a mensagem é enviada é o VSM e assim, os 3 bits designados para este propósito fixam-se nos "011". Como os restantes 21 bits são reservados para uso específico do país, utilizou-se o valor 2 arbitrariamente.

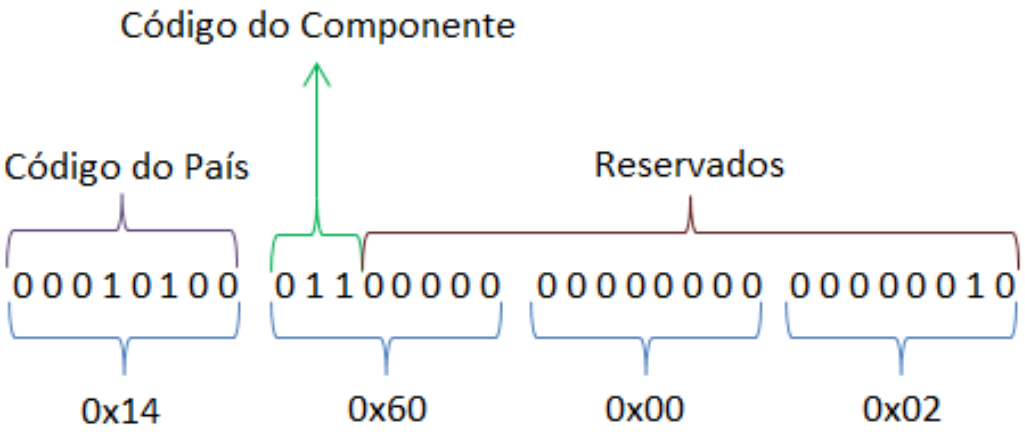

Figura 33: ID de origem do VSM

Concluindo, ID de origem a ser transmitido, divido pelos 4 bytes alocados é “14 60 00 02” (Figura 33).

3.2.4.3 ID de destino

O ID de destino é semelhante ao de origem pois os campos como código do país e os reservados não alteram. O código do componente é que altera, pois a mensagem tem como destino o CUCS que é identificado com o valor “1”.

Assim o código do componente altera para “001” e o segundo byte passa a ser “0x20”.

Com as alterações efetuadas o ID de destino transmitido é “14 20 00 02”.

3.2.4.4 Tipo de Mensagem

Para este campo estão alocados 2 bytes e indicam o tipo de mensagem que está a ser enviada de modo a tratar os dados da carga útil de forma adequada tendo em conta o seu tipo. O tipo de mensagem, como já referido é “Inertial States” e é identificado pelo número 4000. Este valor convertido em hexadecimal e distribuído pelos dois bytes fica “0F A0”.

3.2.4.5 Propriedades da Mensagem

Como já especificado neste documento, os bytes alocados às propriedades são dois e estão divididos da seguinte forma (Figura 34): indicação de *acknowledgement* (1bit); versão de IDD (7bits); comprimento *checksum* (2bits); utilização futura (6bits).

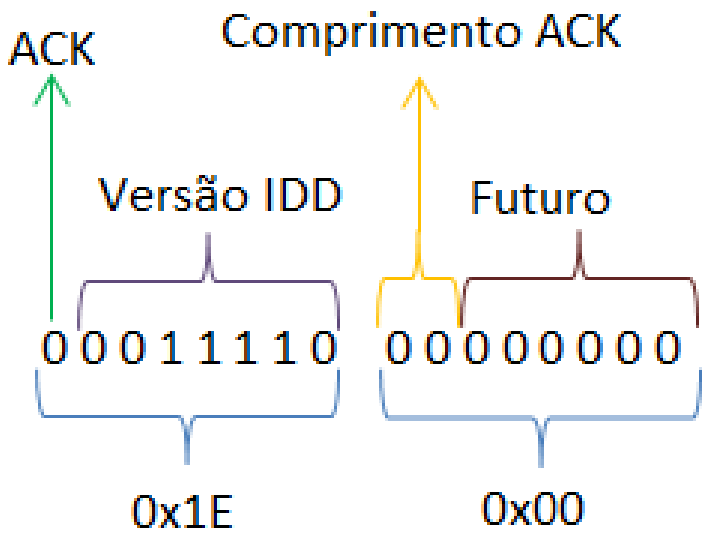

Figura 34: Propriedades da mensagem referida

Existem dois tipos de mensagem: "Push" e "Pull". As mensagens "Push" são mensagens enviadas periodicamente ou enviadas com base num evento ou acontecimento. Estas mensagens não necessitam de acknowledgment. As mensagens "Pull" são mensagens que são transmitidas em resposta a um pedido. Esta divisão ou categorização das mensagens é feita de modo a minimizar a utilização da largura de banda necessária.

A mensagem referente a este caso, mensagem "*Inertial States*", é uma mensagem "*Push"*, pois é enviada periodicamente de modo a informar e manter o CUCS atualizado em relação aos dados de estado do veículo.

Assim, o primeiro campo que indica se a mensagem requere acknowledgment fixa-se no valor "0".

A versão do IDD, como já referido anteriormente, em referência à 3ª edição do STANAG 4586 utiliza o valor 30 (decimal) que em hexadecimal é representado por 0x1E.

O comprimento ACK é um campo não utilizado pois não é necessário e é representado pelo valor nulo. Os últimos 6 bits nulos estão reservados para utilização futura e não são assim utilizados.

3.2.4.6 Carga Útil

Os dados referentes a esta mensagem estão especificados na 3ª edição do STANAG 4586 e foram extrapolados para a tabela 13.

Tabela 13: Campos dos dados da mensagem referida

Índice	Designação	Tipo	Unidades
0	Vetor Presença	3 bytes	Nil
1	Time Stamp	5 bytes	…
2	Latitude	4 bytes	BAM
3	Longitude	4 bytes	BAM
4	Altitude	3 bytes	0.02 m
5	Tipo de altitude	Byte	…
6	U_speed	2 bytes	0.05m/s
7	V_speed	2 bytes	0.05m/s
8	W_speed	2 bytes	0.05m/s
9	U_accel	2 bytes	$0.05m/s^2$
10	V_accel	2 bytes	$0.05m/s^2$
11	W_accel	2 bytes	$0.05m/s^2$
12	Phi	2 bytes	BAM
13	Theta	2 bytes	BAM

14	Psi	2 bytes	BAM
15	Phi_dot	2 bytes	0.0005 rad/s
16	Theta_dot	2 bytes	0.0005 rad/s
17	Psi_dot	2 bytes	0.0005 rad/s
18	Variação Magnética	2 bytes	BAM

Todas as mensagens começam com um vetor presença. O vetor presença fornece a quem transmite um método de modo a não enviar dados que não sejam aplicáveis ao contexto da mensagem e assim minimizar a largura de banda da transmissão e/ou tempo de processamento. Os campos que não são utilizados ou não têm relevância na presente mensagem são eliminados tendo em conta o vetor presença.

O vetor presença é um campo mapeado bit a bit que indica os campos presentes (a serem considerados) nos restantes dados da mensagem. Cada bit representa um campo específico. O bit "1" numa certa localização indica que o respetivo campo está presente e o bit "0" indica que o campo está ausente ou não deve ser considerado.

O bit menos significativo indica a presença do "time stamp" (também presente em todas as mensagens) e que se encontra no campo de índice 1.

De referir ainda que o STANAG só envia dados caso tenham existido alterações aos dados que foram enviados anteriormente. Caso um certo dado não esteja presente na mensagem (tendo em conta o vetor presença) deve ser considerado como válido o último dado recebido.

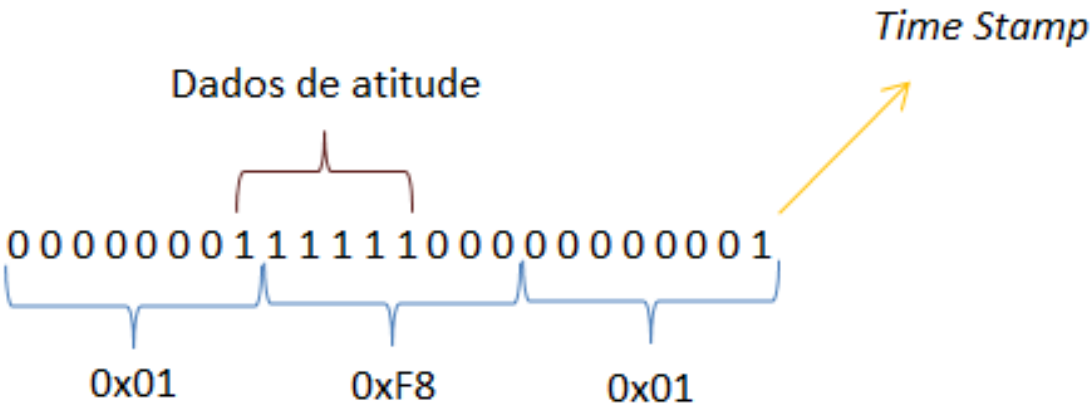

Figura 35: Campos do vetor presença

O vetor presença referente a esta mensagem está representado na figura seguinte sendo que o número final referente aos 3 bytes é “01 F8 01”.

Os 5 bytes alocados ao *Time Stamp* indicam a data em que a mensagem foi criada. Para os objetivos desta dissertação, esses valores não são relevantes e devido a esse facto foram descartados apresentando valores nulos.

Os dez campos seguintes não estão presentes na mensagem pois o seu conteúdo não tem importância no estudo desta dissertação.

O campo designado por “Phi” (ϕ) representa o ângulo do azimute (yaw) alocado no espaço de dois bytes convertido em unidades BAM (Binary Angular Measurement). Estas unidades são utilizadas principalmente por programadores C de modo a representar um número de vírgula flutuante em bytes aritméticos.

Neste programa foi feita a conversão de radianos para ângulos e de ângulos para BAMs.

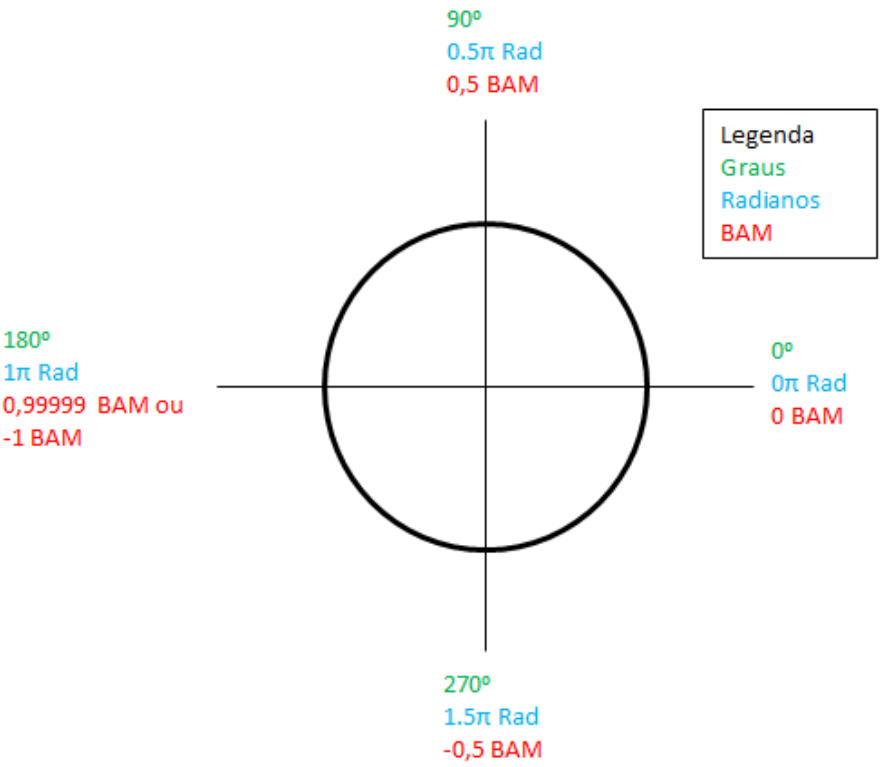

Figura 36: Conversão de dados em graus, radianos e BAM

O campo designado por “Theta” (Θ) representa o angulo do cabeceio (*Pitch*) e o campo ”Psi” (Ψ) o ângulo de rotação (*Roll*). Estes dois dados, similarmente com o anterior estão alocados em 2 bytes com a unidade de BAM. A cada um foi feita a devida conversão no seu eixo referencial.

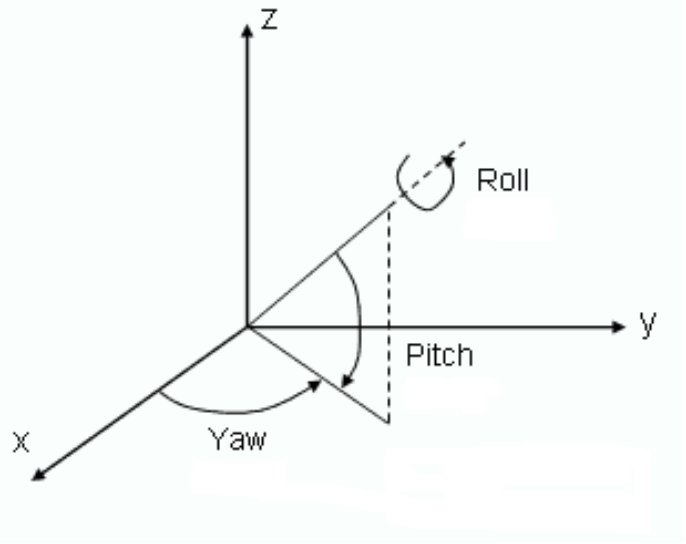

Figura 37: Ângulos de atitude recebidos

O ϕ roda em torno do eixo ZZ, o Θ em torno do eixo YY e o Ψ em torno do eixo XX, como é possível constatar na figura 37.

Os 3 campos seguintes representam a taxa de rotação de cada um dos três ângulos descritos anteriormente. O "Phi_dot", "Theta_dot" e o "Psi_dot" encontram-se em unidades que rondam o meio milionésimo de radiano por segundo (0,0005 rad/s) e representam, respetivamente, a taxa de rotação angular do "Phi", "Theta" e "Psi".

O último campo não é aplicável ao conceito da dissertação devido representar a variação magnética, dispensável neste caso.

3.2.5 Prog_MAV_STANAG

O programa "Prog_MAV_STANAG", como o nome indica, é o programa principal que engloba todas as funções, métodos, conversões e programas anteriormente descritos e explicados neste documento.

Este programa, englobando todos os sub capítulos de software descritos neste documento apresenta numa primeira fase um display como o ilustrado na figura 38.

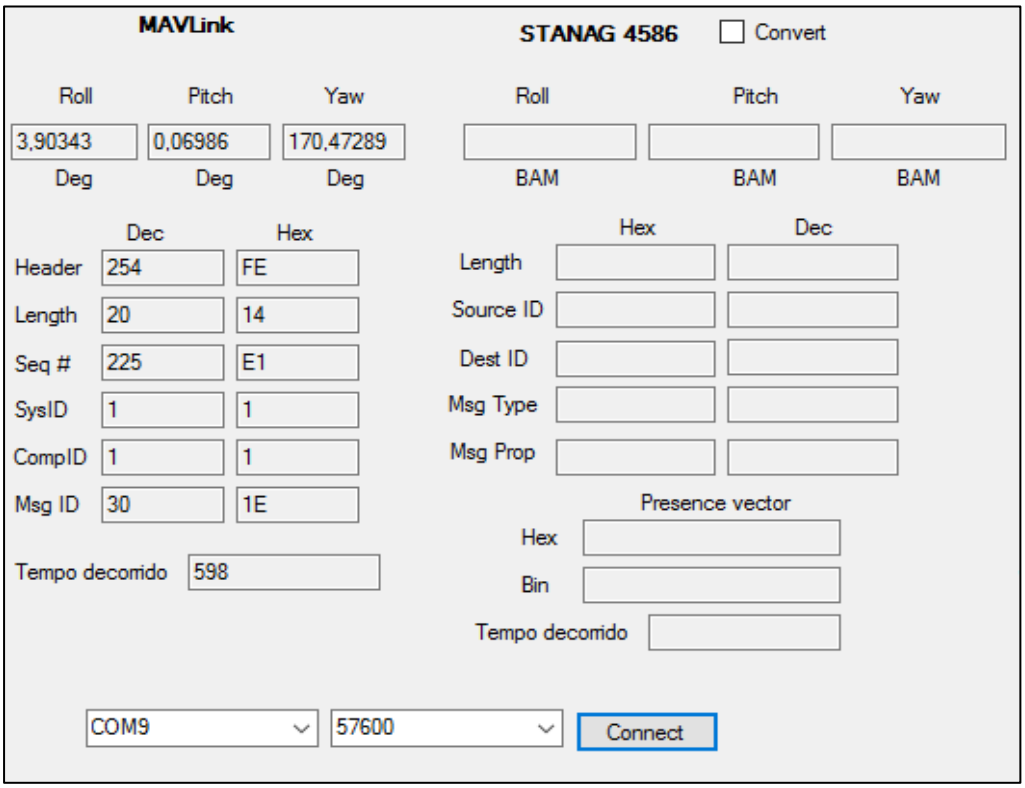

Figura 38: Output final do programa

O programa, ao ser selecionada a porta do dispositivo, a taxa de transmissão e pressionado o botão “Connect” executa todo o código descrito anteriormente até ao programa MAVLink. Assim são apresentados os dados requeridos de atitude do dispositivo em formato MAVLink (graus/degrees). Na minha opinião, como a conversão foi feita em termos de unidades e formato de mensagem, tomei a iniciativa de apresentar também os dados principais de cabeçalho da mensagem que contém os dados de atitude, como se pode ver na figura 38. Toda a parte direita da imagem não apresenta dados devido ao facto de não ter sido ainda executado o código relativo a esses blocos. Na imagem é ainda possível visualizar um campo de descreve o tempo decorrido desde a conexão até à apresentação da mensagem, que irá ser analisado mais adiante neste documento.

O lado STANAG é ativado quando a “checkbox” correspondente for selecionada. Assim que a anterior é selecionada, o programa corre todo o código desenvolvido fazendo a conversão da mensagem MAVLink para a mensagem STANAG 4586. A figura 39 demonstra o visor do programa ao ativar a “checkbox” designada

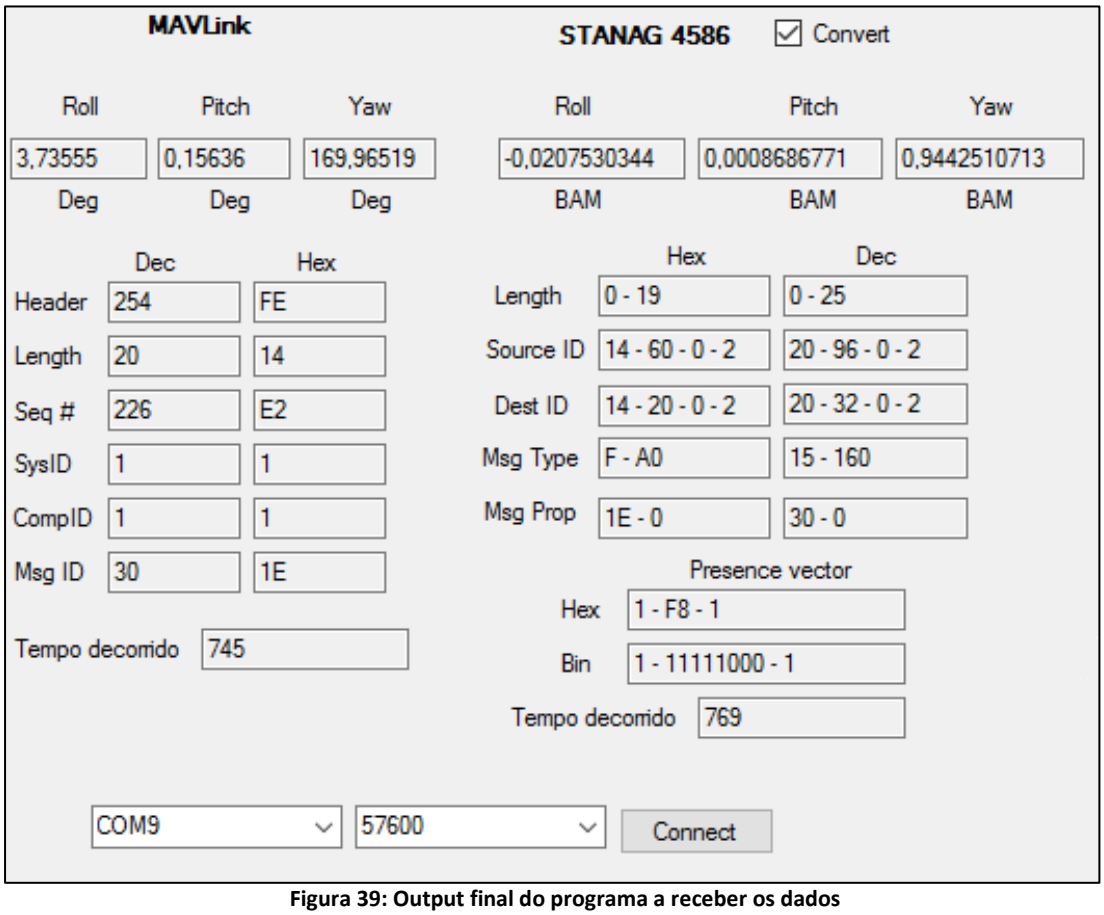

Figura 39: Output final do programa a receber os dados

C*onvert.*

Ao ativar a "*checkbox*" o programa faz a conversão para STANAG 4586 dos dados de atitude do componente em formato STANAG 4586, ou seja, em BAMs. Seguidamente são apresentados os dados da mensagem que transmite esses mesmos dados em formato STANAG 4586 em que se inclui o vetor presença. É também equacionado o tempo decorrido desde o início do programa até à apresentação dos dados neste formato. Os dados do cabeçalho da mensagem apresentados tanto em hexadecimal como em decimal estão separados por bytes de modo a possibilitar uma leitura mais intuitiva dos mesmos.

Conclusão

Neste capítulo pretende-se resumir toda a investigação desenvolvida ao longo do período que foi designado para o desenvolvimento desta dissertação de mestrado. Segue-se uma análise dos resultados obtidos como algumas conclusões detalhadas acerca do projeto. Assim como algumas considerações para o futuro com o intuito de desenvolver e melhorar este projeto em dissertações de mestrados em anos seguintes.

Síntese do Trabalho Efetuado

Como em todos os projetos, primeiramente é necessário um estudo aprimorado de bibliografia relativa ao tema que se pretende abordar bem como um estudo detalhado da mesma. Esse trabalho foi realizado e está explanado no primeiro capítulo. No início desse capítulo faz-se uma abordagem ao Sistema Aéreo Não Tripulado onde está inserido o dispositivo utilizado no trabalho experimental alusivo a esta dissertação. Seguidamente abordam-se os temas relativos aos protocolos e linguagens de comunicação utilizados pelos dispositivos dos UAVs, mais especificamente o STANAG 4586 e o MAVLink. De referir a importância que a estruturação de cada mensagem irá ter no trabalho desenvolvido no decorrer deste documento.

Necessariamente, é detalhada a Metodologia de Investigação utilizada nesta dissertação dando a ideia de como se pretende desenvolver a investigação e que agora se pode afirmar que foi cumprida e se entende como a melhorar abordagem a este tema.

Posteriormente apresenta-se a arquitetura do sistema utilizado na investigação passando pelo *hardware* e terminando com o software. O *hardware* baseia-se no piloto automático Pixhawk e estão descritas as especificações e características do mesmo.

O *software* refere-se à programação desenvolvida e pode ser resumida pela figura 40.

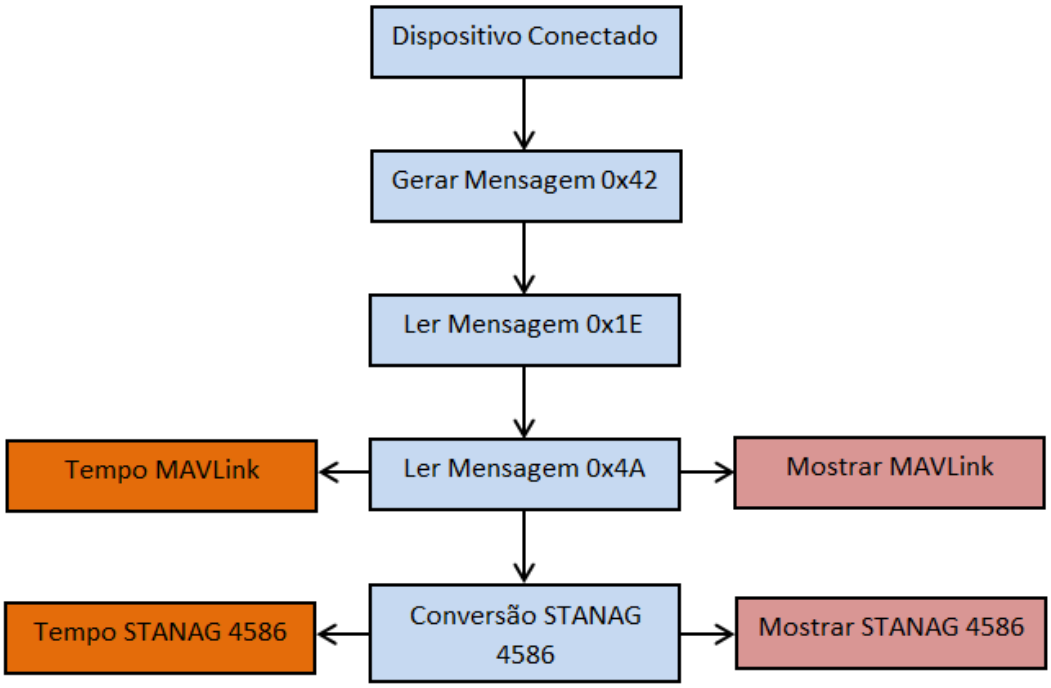

Figura 40: Esquema blocos resumo do programa final

Primeiramente verifica se o dispositivo está conectado. Seguidamente gera a mensagem que requer os dados pretendidos e envia-a para o dispositivo. O dispositivo ao receber a mensagem e reconhecê-la, transmite a mensagem que funciona como batimento cardíaco e a mensagem onde são transmitidos os dados requeridos. O programa lê as duas mensagens e recolhendo os dados da segunda, mostra-os ao utilizador ao mesmo tempo que calcula o tempo decorrido até este instante. Após estes eventos, o programa converte esses mesmos dados numa mensagem em formato STANAG 4586 e mostra-os ao utilizador ao mesmo tempo que calcula o tempo decorrido desde o início do processo.

Conclusões e Análise de Resultados

Como especificado neste documento, foi realizada uma medição temporal de cada processo (referência à figura 23). O processo A refere-se à receção da mensagem em formato MAVLink e posterior leitura e apresentação. O processo B refere-se ao

processo A complementado com a conversão dos dados para uma mensagem em formato STANAG 4586 e posterior apresentação dos mesmos.

Foram realizados várias medições ao longo dos referidos processos. As primeiras medições foram efetuadas utilizando os segundos decorrentes de um processo e de outro. O *delay* que se pretende analisar será o valor temporal do processo B subtraindo o valor temporal do processo A. Logo de início se percebeu que o tempo decorrido em segundos não poderia ser a unidade utilizada pois em termos de segundos os processos decorrem no mesmo intervalo temporal.

Realizou-se a medição dos mesmos períodos temporais em milissegundos e por fim foi percebido que estas deveriam ser as unidades utilizadas nesta análise temporal de processos.

O gráfico 1 mostra um gráfico onde é visível o período temporal decorrente tanto de um processo como de outro.

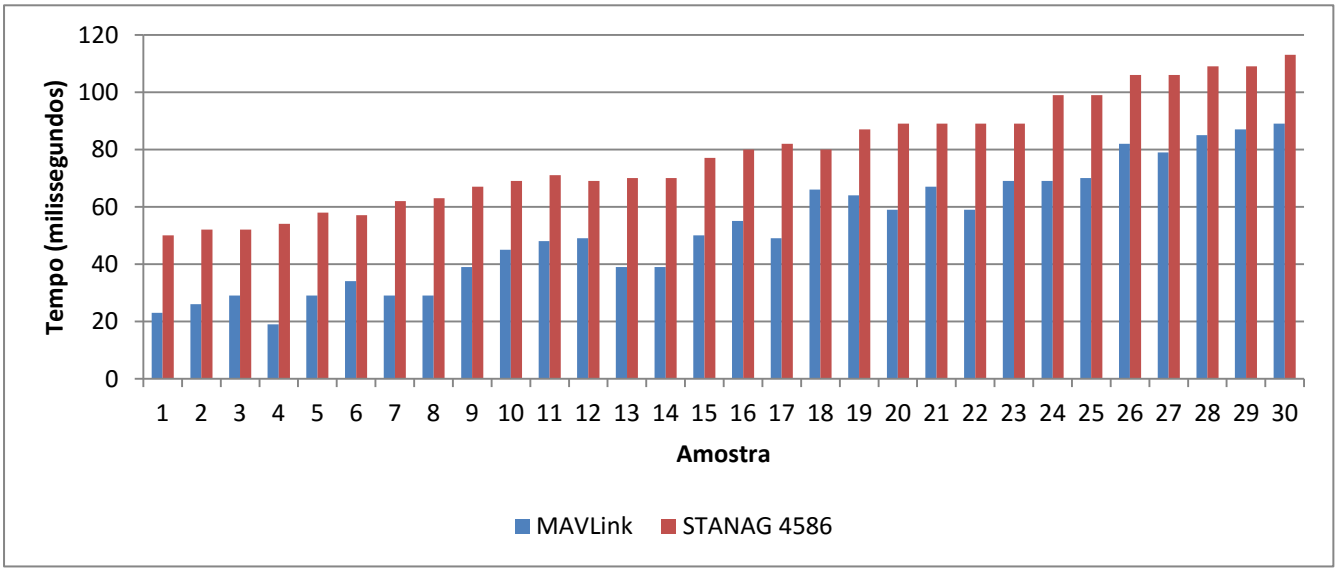

Gráfico 1: Tempo decorrente de cada processo

Foi selecionada uma amostra de trinta mensagens e respetivas conversões arbitrariamente. O resultado gráfico é o apresentado acima onde é possível verificar o tempo que cada processo demora em milissegundos.

O objetivo é analisar se a distância temporal entre os processos não viabiliza a junção da conversão STANAG 4586 ao processo A. Em termos de requisitos operacionais foi definido pela empresa o valor de 500 milissegundos como tempo máximo de distância temporal entre os processos.

Fazendo uma análise da distância temporal entre os processos do gráfico acima obtemos o gráfico 2.

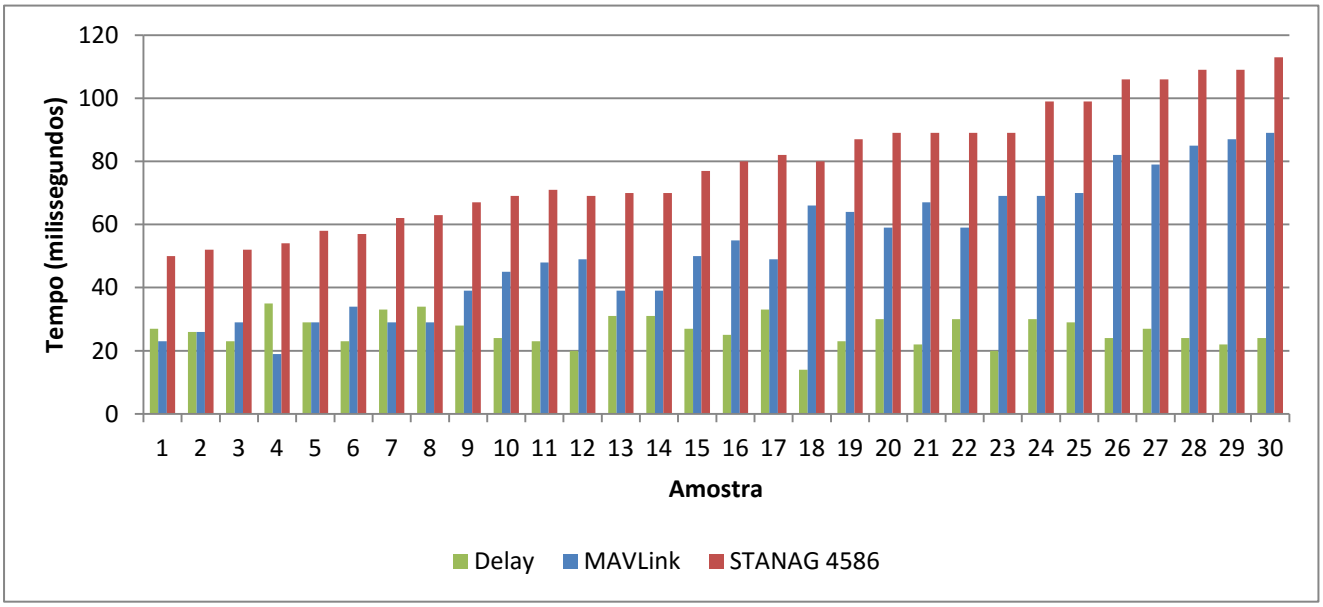

Gráfico 2: Tempo decorrente de cada processo e respetivo delay

Neste segundo gráfico é possível analisar a distância temporal em cada uma das amostras e é correto afirmar que em todas as amostras a distância temporal (delay) se situa no intervalo entre os 15 milissegundos e os 40 milissegundos. De modo a obter um valor aproximado da média do *delay* obtido nesta amostra calculou-se a tendência linear da mesma.

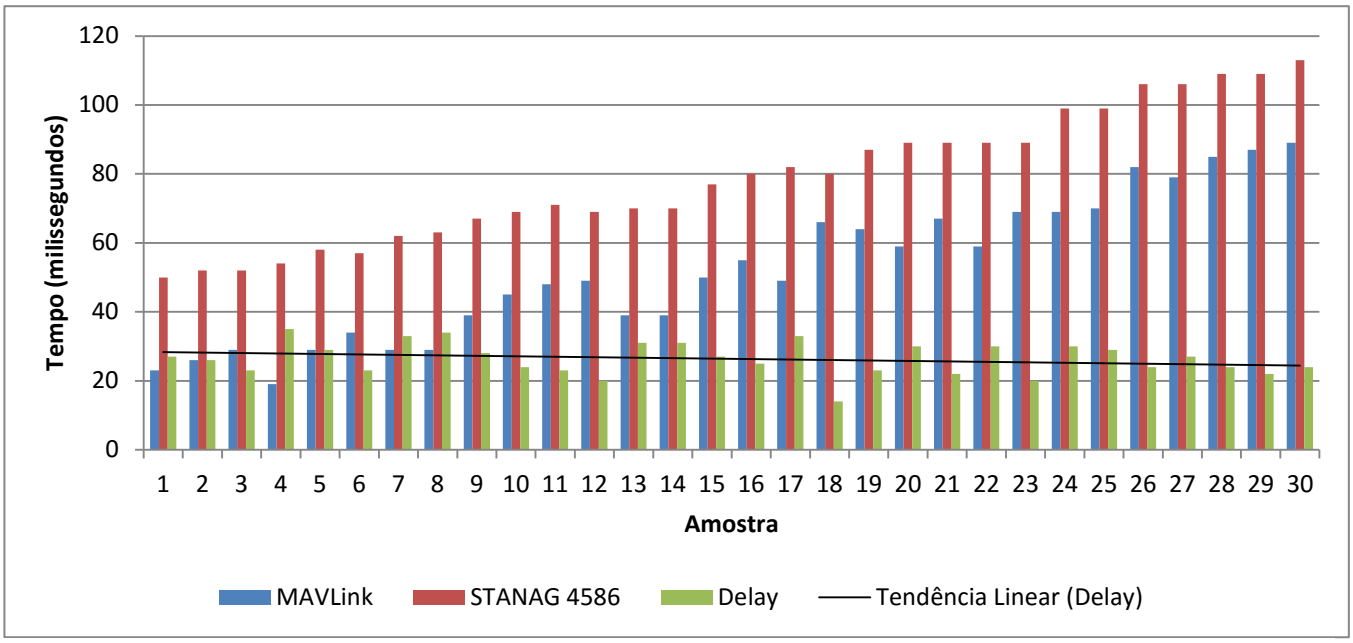

Gráfico 3: Tempo decorrente de cada processo e tendência linear do delay associado

Com a visualização da linha de tendência linear dos dados de *delay* no gráfico 3 é possível verificar que a média se encontra arredondada nos valores de 25 milissegundos. Este valor valida a premissa de que a conversão STANAG é viável em termos operacionais tendo em conta os requisitos para o dispositivo e sistemas da empresa UAVision.

No sistema que se quis simular, o piloto automático comunica com a estação de controlo terrestre através de uma ligação de 2.4 GHz em que o seu protocolo de telemetria (MAVLink) é encapsulado sobre IP (*Internet Protocol*). O IP aumenta a robustez, a padronização e a conetividade de sistemas díspares, o que por si só beneficia em muito o sistema.

Trabalho futuro

Para a comunicação com uma GCS funcionando em STANAG 4586, todas as comunicações requerem uma variedade de mensagens iniciais de modo a certificar a ligação para um certo nível de interoperabilidade. Essas certificações e mensagens associadas não são objeto de estudo nesta dissertação pois iria ser necessário adquirir uma GCS STANAG 4586 o que engloba valores demasiado elevados para esta investigação. Por isso, como trabalho futuro deixo a proposta de se adquirir essa

mesma GCS por forma a efetuar os mesmos testes com essas mensagens e certificações de modo a validar o tempo de resposta.

Como segunda proposta proponho a implementação de um sistema que predisponha esta conversão. Sistema que poderá ser incluído no veículo sendo implementado através de um dispositivo (e.g. Rapsberry Pi). Este tema é o alvo da dissertação de mestrado do CAD EN-AEL Valério Rodrigues, que irá por sua vez, dar continuidade ao trabalho desenvolvido nesta dissertação de mestrado.

Referências Bibliográficas

ADKINS, C. e Liebeck, R. (1994), *Design of optimum propellers*, Journal of Propulsion and Power

ALEXANDROV, B. (2004), *Three-component variometer based on a scalar potassium sensor*, Measurement Science and Technology

AMIDI, O., KANADE, T. e FUJITA, K. (1998), *A visual odometer for autonomous helicopter flight*, Robotics and Autonomous Systems, vol. 28

BAILLIEUL, J. e ANTSAKLIS, P. (2007), *Control and communication challenges in networked real-time systems*, Proceedings of the IEEE

BARBER, D. et al. (2007), *Autonomous landing of miniature aerial vehicles*, Journal of Aerospace Computing, Information and Communication

BENDEA H. et al. (2008), Low cost UAV for post-disaster assessment, Int. Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Beijing, China

BERTRAM, V. (2008), *Unmanned surface vehicles - A survey*, In Skibsteknisk Selskab, Copenhagen, Denmark

BETTNER, J.L. e BLANDFORD, C.S. (1995), *Propulsion System Assessment for Very High Altitude UAV Under ERAST*, Final Report Draft 2, Allison Engine Company

BLONDEL, P. and MURTON, B. J. (1997), *Handbook of seafloor sonar imagery,* Chichester, UK

BOARD, N. S. (2005), Autonomous vehicles in support of naval operations, National Academies Press

BONE, E. e BOLKCOM, C. (2003), *Unmanned aerial vehicles: Background and issues for congress*, LIBRARY OF CONGRESS WASHINGTON DC CONGRESSIONAL RESEARCH SERVICE.

BRELSTAFF, G. J. et al. (1995), *Hyperspectral camera system: acquisition and analysis,* In Satellite Remote Sensing II, International Society for Optics and Photonics

Brickner, W. (2005), *An Analysis of the Kill Chain for Time-Critical Strike*, NAVAL POSTGRADUATE SCHOOL MONTEREY CA

BRYSON, M. e SUKKARICH, S. (2004), *Vehicle Model Aided Inertial Navigation for a UAV using Low-cost Sensors*, In Proceedings of the Australasian Conference on Robotics and Automation, Australian Robotics and Automation Association

BUTTON, R. et al. (2015), *A Survey of Missions for Unmanned Undersea Vehicles*, Santa Monica, Calif.: RAND Corporation, MG-808-NAVY

CACCIA, M. INDIVERI, G. e VERUGGIO, G. (2000), *Modeling and Identification of Open-Frame Variable Configuration Unmanned Underwater Vehicles*, IEEE Journal of Oceanic Engineering

CAI, G. et al. (2010), *An overview on development of miniature unmanned rotorcraft systems.* Frontiers of Electrical and Electronic Engineering in China

CASBEER, D. (2006), *Cooperative forest fire surveillance using a team of small unmanned air vehicles*, International Journal of Systems Sciences

CHAN, C. (2002), *The state of the art of electric and hybrid vehicles,* Proceedings of the IEEE

CHRISTENSEN, H. e HEDSTROM, A. (2004), *STANAG - JAUS Study*, Royal Institute of Technology, Stockholm, Sweden

COOMBES, M. et al. (2012), *Development of an autopilot system for rapid prototyping of high level control algorithms*, Proceedings of the 2012 UKACC International Conference on Control, CONTROL 2012, Cardiff

ĆOSIĆ, J. et al. (2013), *Interpreting Development of Unmanned Aerial Vehicles Using Systems Thinking,* Interdisciplinary Description of Complex Systems 11

CUADRADO, R. et al. (2013), *Architecture issues and challenges for the integration of*

RPAS in non-segregated airspace

CUTRONA, L. J. (1990), *Synthetic aperture radar*, Radar handbook

DALAMAGKIDIS, K. et al. (2008), *Current status and future perspectives for unmanned aircraft system operations in the US*, Journal of Intelligent and Robotic Systems

DEGARMO, M. e NELSON, G. (2004), *Prospective Unmanned Aerial Vehicle Operations in the Future National Airspace System,* AIAA-2004-6243, AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum, Chicago, Illinois

DIXON, S. (2005), *Mission control of multiple unmanned aerial vehicles: A workload analysis,* Human Factors

DRAPER, M. et al. (2003), *Manual versus speech input for unmanned aerial vehicle control station operations*, in Proc. Hum. Factors Ergonom. Soc. 47th Annu. Meet

DUDEK, G. et al. (1993*), A taxonomy for swarm robots,* In Intelligent Robots and Systems, Proceedings of the 1993 IEEE/RSJ International Conference

DUDEK, G. et al. (1996), *A taxonomy for multi-agent robotics*, Autonomous Robots

DUELLEY, R. (2010), *Autonomous Underwater Vehicle Propulsion Design*, Masters thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia

DUNBABIN, M. (2009), *An autonomous surface vehicle for water quality monitoring,* in Proc. Australasian Conf. Robotics and Automation, Sydney, Australia

FEITSHANS, G. L. et al. (2008), *Vigilant spirit control station (VSCS)—'The face of COUNTER,* In Proceedings of AIAA Guidance, Navigation and Control Conference Exhibition

FLETCHER, B. (2000), *UUV Master Plan: A Vision for Navy UUV Development*, Proceedings of the Oceans Conference

FREITAG, L. (2005), *The WHOI Micro-Modem: An Acoustic Communications and Navigation System for Multiple Platforms*, In Proc. IEEE OCEANS'05 Conf., September

GAGE, D. (1995) *UGV history 101: A brief history of Unmanned Ground Vehicle (UGV) development efforts*, Unmanned Systems Magazine

GARCIA-PARDO, P. et al. (2002), *Towards vision-based safe landing for an autonomous helicopter*, Robotics and Autonomous Systems, vol. 38

GRASMEYER, J. e KEENON, M. (2001), *Development of the Black Widow Micro Air Vehicle,* AIAA Paper 2001-0127, JAN

GREGOR, R. (2002), *EMS-Vision: A perceptual system for autonomous vehicles,* IEEE Transactions on Intelligent Transportation Systems

GROCHOLSKY, B. et al. (2006), *Cooperative air and ground surveillance*, IEEE Robot. Automat. Mag., vol. 13

GUR, O. e ROSEN, A. (2009), *Optimizing electric propulsion systems for unmanned aerial vehicles*, Journal of aircraft

HADDON, D. & Whittaker, C. (2003), *Aircraft airworthiness certification standards for civil UAVs*, Aeronautical Journal, *107*(1068 Spec)

HEISENBEISS, H. (2004), *A mini unmanned aerial vehicle (UAV): system overview and image acquisition*, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences

HENDERSON, I. e CAVANAGH, B. (2014*), Unmanned Aerial Vehicles: Do They Pose Legal Challenges?*, In *New Technologies and the Law of Armed Conflict,* TMC Asser Press.

HENG, L. et al. (2011), *Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing,* In Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Shanghai. New York: IEEE Press.

HUDSON, E. T. LICHT, S. C. e EICKSTEDT, D. P. (2012), *U.S. Patent Application 13/415*

HUANG, HM. (2004), *Autonomy levels for unmanned systems (alfus) framework, volume i: Terminology*, National Institute of Standards and Technology

IZADI-ZAMANABADI, R. e BLANKE, M. (1999), *A ship propultion system as a benchmark for fault-tolerant control,* Control Engineering Practice

JOHNSON, E. et al. (2001), *Adaptive Guidance and Control for Autonomous Launch Vehicles*, IEEE Aerospace Conference, Big Sky, MT

JOVANOVIC, M. e STARCEVIC, D. (2008), *Software Architecture for Ground Control Station for Unmanned Aerial Vehicle*, In Proc. 10th IEEE Int. Conf. on Computer Modeling and Simulation, Cambridge, UK

KAHAN, W. (1997), *IEEE standard 754 for binary floating-point arithmetic*, Lecture Notes on the Status of IEEE

KAMRANI, F. (2007), *Using on-line simulation in UAV path planning*

KEANE, J. F. e CARR, S. S. (2013). A brief history of early unmanned aircraft, *Johns Hopkins APL Technical Digest*

KIM, J. (2003), *Real-time Navigation, Guidance and Control of a UAV using Low-cost Sensors*, in International Conference of Field and Service Robotics, Yamanashi, Japan

KOLOMEITSEV, L. et al. (2008), *Linear switched reluctance motor as high efficiency propulsion system for railway vehicles,* in Proc. SPEEDAM

LEE, D., & CHONG, K. (2013), *Remote Control of Machine Vision Processing System*

LEVINSON, J. (2011), *Towards fully autonomous driving: Systems and algorithms,* in Proc. IEEE IV, JUN

LIM, J. P. et al. (2012), *Build your own quadrotor (open-source projects on unmanned aerial vehicles)*, IEEE Robotics & Automation Magazine

LUSTOSA, L. et al. (2011), *Sighting device-aided inertial navigation: fusion with Adaptative kalman filtering techniques,* 21st International Congress of Mechanical Engineering, Natal, Brazil

MADHAVAN, R. et al. (2009), *Benchmarking and standardization of intelligent robotic systems*, in Advanced Robotics, ICAR 2009, International Conference

MANLEY, J. E. (2008), *Unmanned surface vehicles, 15 years of development*, In OCEANS 2008, IEEE

MARTY, J. A. (2013), *Vulnerability analysis of the mavlink protocol for command and control of unmanned aircraft*

MCGREW, T. M. (2009), *Army aviation addressing battlefield anomalies in real time with the teaming and collaboration of manned and unmanned aircraft*, NAVAL POSTGRADUATE SCHOOL MONTEREY CA

MEIER, L. et al. (2011), *The PIXHAWK opensource computer vision framework for MAVs*, ISPRS – Int. Arch. Photogramm, Remote Sens. Spatial Inform. Sci

MEIER, L. et al (2011a), *Pixhawk: A system for autonomous flight using onboard computer vision*, In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)

MEROLA, J. M. (1996), *Development of onboard data acquisition for unmanned air vehicle flight testing* (Doctoral dissertation, Monterey, California, Naval Postgraduate School)

MIRA, J. C. C. (2014), NÃO-PROLIFERAÇÃO DE ARMAMENTOS: O CASO DOS MÍSSEIS DE CRUZEIRO, *Revista de Ciências Militares*

MOAT, B. I. et al. (2005), *An overview of the airflow distortion at anemometer sites on ships*, International Journal of Climatology

NAGEL, A. LEVY, D. e SHEPSHELOVICH, M. (2006), *Conceptual Aerodynamic Evaluation of Mini/Micro UAV*, AIAA Paper

Nanda H. and Davis L. (2002), *Probabilistic Template Based Pedestrian Detection in Infrared Videos,* In Procs. IEEE Intelligent Vehicles Symposium, Versailles, France

NAVY, U. S. (2007*), The Navy Unmanned Surface Vehicle (USV) Master Plan*, URL:< http://www. navy. mil/navydata/technology/usvmppr

Nehme C. (2006), *UAV Mission Hierarchy* Boston: Humans and Automation Laboratory, Massachusetts Institute of Technology

NSA (2006), Standardization Agreement (STANAG), Standard Interfaces of UAV Control Systems (UCS) for NATO UAV Interoperability

PASCOAL, A. et al. (2000), *Robotic ocean vehicles for marine science applications: The European ASIMOV project*, in Proceedings of the OCEANS MTS/IEEE Conference, Providence, RI

POSTEL, J. (1980), Search RFCs

RAITHEL Jr, A. (1994), *Patrol Aviation in the Atlantic in World War II*, Naval Aviation News

REKER, N., TROY Jr, D., & TROXELL, D. (2015), *Universal UAV Payload Interface*

ROSA, *G.,* Marques, M. and Lobo, V. (2016), *Unmanned Aerial Vehicles in the Navy: its benefits,* Scientific Bulletin Of Naval Academy, Constanta, Romania

ROSS, J. A. (2008), *Computer Vision and Target Localization Algorithms for Autonomous Unmanned Aerial Vehicles* (Doctoral dissertation, The Pennsylvania State University)

ROWE, S. e WAGNER, C. (2008), *An Introduction to the Joint Architecture for Unmanned Systems (JAUS)*, Technical Report from Cybernet Systems Corporation, http://www.cybernet.com

SCATTOLINI, R. (2009), *Architectures for distributed and hierarchical model predictive control: a review,* J Process Control

SEBASTIAN, R. (2011), *Modelling and simulation of a high penetration wind diesel system with battery energy storage*, International Journal of Electrical Power and Energy Systems

SEGOR, F. et al. (2010), *Mobile Ground Control Station for Local Surveillance*, presented at the ICONS 2010, French Alps, França

SHERMAN, J. et al. (2001), *The autonomous underwater glider 'Spray',* IEEE Journal of Oceanic Engineering, 26(4):437–446, Special Issue on Autonomous Ocean-Sampling Networks

SHILOV, K. (2014), *The Next Generation Design of Autonomous MAV Flight Control System SmartAP // IMAV 2014: International Micro Air Vehicle Conference and Competition*

SKOLNIK, M. I. (1962), *Introduction to radar*, Radar Handbook

SOJOURNER, R. J. e ANTIN, J. F. (1990), *The effects of a simulated head-up display speedometer on perceptual task performance*, Human Factors: The Journal of the Human Factors and Ergonomics Society

"STANAG 4586 Third Edition, Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability, NATO Standardization," November 2012.

STANSBURY, S. et al. (2008), *A Survey of UAS Technologies for Command, Control, and Communication (C3)*, Journal of Intelligent and Robotic Systems, Volume 54, Springer Science + Business Media B.V.

STELZER, R. e JAFARMADAR, K. (2010), *History and Recent Developments in Robotic Sailing,* Robotic Sailing - Proceedings of the 4th International Robotic Sailing Conference

STOREY, M. et al. (2006), *Shared waypoints and social tagging to support collaboration in software development*, In Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work

URABE, M. (2000), *U.S. Patent No. 6,137,434*, Washington, DC: U.S. Patent and Trademark Office

VALDES, R. (2009*), How the predator UAV works*, *HowStuffWorks, Inc., Atlanta, Ga, USA*

VALAVANIS, K. (Ed.) (2008), *Advances in unmanned aerial vehicles: state of the art and the road to autonomy* (Vol. 33), Springer Science & Business Media

WANG, N. et al. (2006), *Wireless sensors in agriculture and food industry—recent development and future perspective,* Comp. Electron. Agric. 50 (1), 1–14

WATTS, A. AMBROSIA, V. e HINKLEY, E. (2012*), Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use*, Remote Sensing

Weatherington D., & Deputy U. (2005*), Unmanned aircraft systems roadmap*, 2005-2030, Deputy, UAV Planning Task Force, OUSD (AT&L)

WZOREK, M. et al. (2006), *GSM technology as a communication media for an autonomous unmanned aerial vehicle,* In Proceedings of the 21st Bristol International Conference on UAV Systems

Apendice 1

Código de Prog_MAV_STANAG

```
-------------------------------------Prog_MAV_STANAG-----------------------------------------------------------

namespace Prog_MAV_STANAG
{
   public partial class Prog_MAV_STANAG : Form
   {
      // inicialization of variables
      int inicio = DateTime.Now.Millisecond;
      int[] delaysmav = new int[30];
      int[] delaysstanag = new int[30];

      MAVLink.MavlinkParse mavlink = new MAVLink.MavlinkParse();
      int initial_time = DateTime.Now.Millisecond;

      public simpleexample()
      {
         InitializeComponent();
      }

      private void but_connect_Click(object sender, EventArgs e)
      {

         // temporal variables
         int countmav = 0;
         int countstanag = 0;


         // if the port is open close it
         if (serialPort1.IsOpen)
         {
            serialPort1.Close();
            return;
         }

         // set the comport options
         serialPort1.PortName = CMB_comport.Text;
         serialPort1.BaudRate = int.Parse(cmb_baudrate.Text);

         // open the comport
         serialPort1.Open();

         // set timeout to 2 seconds
         serialPort1.ReadTimeout = 2000;

         // request streams - assume target is at 1,1
mavlink.GenerateMAVLinkPacket(MAVLink.MAVLINK_MSG_ID.REQUEST_DATA_STREAM,
            new MAVLink.mavlink_request_data_stream_t()
            {
               req_message_rate = 2,
               req_stream_id = (byte)MAVLink.MAV_DATA_STREAM.ALL,
               start_stop = 1,
               target_component = 1,
```

```
      target_system = 1
    });

while (serialPort1.IsOpen)
{
    try
    {
      // try read a hb packet from the comport
      var hb = readsomedata<MAVLink.mavlink_heartbeat_t>();

      var att = readsomedata<MAVLink.mavlink_attitude_t>();

      // test
      Console.WriteLine(att.pitch * 57.2958 + " " + att.roll * 57.2958);


      int a, b, c;
      if (att.roll < 0)
      {
        a = -1;
      }
      else { a = 1; } // sinalize if roll is negative

      if (att.pitch < 0)
      {
        b = -1;
      }
      else { b = 1; } // sinalize if pitch is negative


      if (att.yaw < 0)
      {
        c = -1;
      }
      else { c = 1; } // sinalize if yaw is negative


      // convert absolute value into degrees
      double navroll = Math.Abs(att.roll) * 57.2958;
      double navpitch = Math.Abs(att.pitch) * 57.2958;
      double navyaw = Math.Abs(att.yaw) * 57.2958;

      // display
      textroll.Text = string.Format("{0:N5}", navroll);
      textyaw.Text = string.Format("{0:N5}", navyaw);
      textpitch.Text = string.Format("{0:N5}", navpitch);

      // temporal count
      int delaymav = Math.Abs((DateTime.Now.Millisecond - inicio));
      textBox2.Text = Convert.ToString(delaymav);
      delaysmav[countmav] = delaymav;
```

```
if (convertbutton.Checked)
{
   STANAG_4586 msgSTANAG = new STANAG_4586();
   byte[] msg = msgSTANAG.GenerateMSG("#4000",
   initial_time,
   navyaw,
   navpitch,
   navroll,
   att.yawspeed,
   att.pitchspeed,
   att.rollspeed);

   lengthstanagbox.Text = (msgSTANAG.length1.ToString("X") +
      " - " + msgSTANAG.length2.ToString("X"));
   lengthstanagdec.Text = (msgSTANAG.length1 + " - " + msgSTANAG.length2);

   sourcestanagbox.Text = (msgSTANAG.sourceid1.ToString("X") +
      " - " + msgSTANAG.sourceid2.ToString("X") + " - "
                     + msgSTANAG.sourceid3.ToString("X") + " - " +
  msgSTANAG.sourceid4.ToString("X"));
   sourcestanagdec.Text = (msgSTANAG.sourceid1 + " - " + msgSTANAG.sourceid2 +
   " - "   + msgSTANAG.sourceid3 + " - " + msgSTANAG.sourceid4);
   deststanagbox.Text = (msgSTANAG.destid1.ToString("X") +
      " - " + msgSTANAG.destid2.ToString("X") + " - "
               + msgSTANAG.destid3.ToString("X") + " - " +
           msgSTANAG.destid4.ToString("X"));
   deststanagdec.Text = (msgSTANAG.destid1 + " - " + msgSTANAG.destid2 + " - "
      + msgSTANAG.destid3 + " - " + msgSTANAG.destid4);

   typestanagbox.Text = (msgSTANAG.msgtype1.ToString("X") +
      " - " + msgSTANAG.msgtype2.ToString("X"));
   typestanagdec.Text = (msgSTANAG.msgtype1 + " - " + msgSTANAG.msgtype2);

   propstanagbox.Text = (msgSTANAG.msgprop1.ToString("X") +
      " - " + msgSTANAG.msgprop2.ToString("X"));
   propstanagdec.Text = (msgSTANAG.msgprop1 + " - " + msgSTANAG.msgprop2);

   presencevectorbox.Text = (msgSTANAG.presence_vector[0].ToString("X") + " - "
      + msgSTANAG.presence_vector[1].ToString("X")
      + " - " + msgSTANAG.presence_vector[2].ToString("X"));

  presencevectorbin.Text = ((Convert.ToString(msgSTANAG.presence_vector[0], 2))
  + " - " + (Convert.ToString(msgSTANAG.presence_vector[1], 2)) + " - "
      + (Convert.ToString(msgSTANAG.presence_vector[2], 2)));


   // display
   rollbox.Text = string.Format("{0:N10}",(a * navroll) / 180);
   pitchbox.Text = string.Format("{0:N10}", (b * navpitch) / 180);
   yawbox.Text = string.Format("{0:N10}", (c * navyaw) / 180);
```

```
            // delay
            int delaystanag = Math.Abs(DateTime.Now.Millisecond - inicio);
            delaysstanag[countstanag] = delaystanag;


            if (delaysstanag[countstanag]<delaysmav[countmav])
            {
               delaysstanag[countstanag] = delaysstanag[countstanag] + 1000;
            }
            textBox3.Text = Convert.ToString(delaystanag);

            countstanag++;
            countmav++;

          }
        }
        catch
        {
        }

        System.Threading.Thread.Sleep(1);
        Application.DoEvents();

    }
}

T readsomedata<T>(int timeout = 2000)
{
    DateTime deadline = DateTime.Now.AddMilliseconds(timeout);

    // read the current buffered bytes
    while (DateTime.Now < deadline)
    {

        var packet = mavlink.ReadPacketObj(serialPort1.BaseStream);

        if (packet == null)
            continue;

        Console.WriteLine(mavlink.buffer1[5]);
        Console.WriteLine(packet);

        if (packet.GetType() == typeof(T))
        {


            textheader.Text = mavlink.buffer1[0]+"";
            textlength.Text = mavlink.buffer1[1] + "";
            textseq.Text = mavlink.buffer1[2] + "";
            textsysid.Text = mavlink.buffer1[3] + "";
```

```
            textcompid.Text = mavlink.buffer1[4] + "";
            textmsgid.Text = 30 + "";
            textlength0x.Text = mavlink.buffer1[1].ToString("X");
            textheader0x.Text = mavlink.buffer1[0].ToString("X");
            textseq0x.Text = mavlink.buffer1[2].ToString("X");
            textsys0x.Text = mavlink.buffer1[3].ToString("X");
            textcomp0x.Text = mavlink.buffer1[4].ToString("X");
            textID0x.Text = "1E";

            return (T)packet;
        }
    }

    throw new Exception("No packet match found");
}



private void CMB_comport_Click(object sender, EventArgs e)
{
    CMB_comport.DataSource = SerialPort.GetPortNames();
}

private void simpleexample_Load(object sender, EventArgs e)
{

}

private void label2_Click(object sender, EventArgs e)
{

}

private void textyaw_TextChanged(object sender, EventArgs e)
{

}

private void label11_Click(object sender, EventArgs e)
{

}

private void button1_Click(object sender, EventArgs e)
{

}

private void label35_Click(object sender, EventArgs e)
{

}
```

```
  }
}
```

----------Class MAVLink----------

```
public partial class MAVLink
{

public static readonly byte[] MAVLINK_MESSAGE_LENGTHS = new byte[] {9, 31, 12, 0,
14, 28, 3, 32, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 20, 2, 25, 23, 30, 101, 22,
26, 16, 14, 28, 32, 28, 28, 22, 22, 21, 6, 6, 37, 4, 4, 2, 2, 4, 2, 2, 3, 13, 12,
37, 0, 0, 0, 27, 25, 0, 0, 0, 0, 0, 68, 26, 185, 181, 42, 6, 4, 0, 11, 18, 0, 0,
37, 20, 35, 33, 3, 0, 0, 0, 22, 39, 37, 53, 51, 53, 51, 0, 28, 56, 42, 33, 0, 0,
0, 0, 0, 0, 0, 26, 32, 32, 20, 32, 62, 44, 64, 84, 9, 254, 16, 0, 36, 44, 64, 22,
6, 14, 12, 97, 2, 2, 113, 35, 6, 79, 35, 35, 22, 13, 255, 14, 18, 43, 8, 22, 14,
36, 43, 41, 0, 0, 14, 0, 0, 0, 36, 60, 30, 42, 8, 4, 12, 15, 13, 6, 15, 14, 0,
12, 3, 8, 28, 44, 3, 9, 22, 12, 18, 34, 66, 98, 8, 48, 19, 3, 20, 24, 29, 45, 4,
40, 2, 206, 7, 29, 0, 0, 0, 0, 27, 44, 22, 25, 0, 0, 0, 0, 0, 42, 14, 2, 3, 2, 1,
33, 1, 6, 2, 4, 2, 3, 2, 0, 1, 3, 2, 4, 2, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 32, 52, 53, 0, 0, 38, 0, 254, 36, 30, 18, 18, 51, 9, 0};

        public static readonly byte[] MAVLINK_MESSAGE_CRCS = new byte[] {50, 124,
137, 0, 237, 217, 104, 119, 0, 0, 0, 89, 0, 0, 0, 0, 0, 0, 0, 0, 214, 159, 220,
168, 24, 23, 170, 144, 67, 115, 39, 246, 185, 104, 237, 244, 222, 212, 9, 254,
230, 28, 28, 132, 221, 232, 11, 153, 41, 39, 78, 0, 0, 0, 15, 3, 0, 0, 0, 0, 0,
153, 183, 51, 82, 118, 148, 21, 0, 243, 124, 0, 0, 38, 20, 158, 152, 143, 0, 0,
0, 106, 49, 22, 143, 140, 5, 150, 0, 231, 183, 63, 54, 0, 0, 0, 0, 0, 0, 0, 175,
102, 158, 208, 56, 93, 138, 108, 32, 185, 84, 34, 0, 124, 237, 4, 76, 128, 56,
116, 134, 237, 203, 250, 87, 203, 220, 25, 226, 46, 29, 223, 85, 6, 229, 203, 1,
195, 109, 168, 181, 0, 0, 131, 0, 0, 0, 154, 178, 200, 134, 219, 208, 188, 84,
22, 19, 21, 134, 0, 78, 68, 189, 127, 154, 21, 21, 144, 1, 234, 73, 181, 22, 83,
167, 138, 234, 240, 47, 189, 52, 174, 229, 85, 159, 186, 72, 0, 0, 0, 0, 92, 36,
71, 98, 0, 0, 0, 0, 0, 134, 205, 94, 128, 54, 63, 112, 201, 221, 226, 238, 103,
235, 14, 0, 77, 50, 163, 115, 47, 0, 0, 0, 0, 0, 0, 207, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 90, 104, 85, 0, 0, 158, 0, 8, 204, 49, 170, 44, 83, 46, 0};

        public static readonly Type[] MAVLINK_MESSAGE_INFO = new Type[] {typeof(
mavlink_heartbeat_t ), typeof( mavlink_sys_status_t ), typeof(
mavlink_system_time_t ), null, typeof( mavlink_ping_t ), typeof(
mavlink_change_operator_control_t ), typeof(
mavlink_change_operator_control_ack_t ), typeof( mavlink_auth_key_t ), null,
null, null, typeof( mavlink_set_mode_t ), null, null, null, null, null, null,
null, null, typeof( mavlink_param_request_read_t ), typeof(
mavlink_param_request_list_t ), typeof( mavlink_param_value_t ), typeof(
mavlink_param_set_t ), typeof( mavlink_gps_raw_int_t ), typeof(
mavlink_gps_status_t ), typeof( mavlink_scaled_imu_t ), typeof( mavlink_raw_imu_t
), typeof( mavlink_raw_pressure_t ), typeof( mavlink_scaled_pressure_t ), typeof(
mavlink_attitude_t ), typeof( mavlink_attitude_quaternion_t ), typeof(
mavlink_local_position_ned_t ), typeof( mavlink_global_position_int_t ), typeof(
mavlink_rc_channels_scaled_t ), typeof( mavlink_rc_channels_raw_t ), typeof(
mavlink_servo_output_raw_t ), typeof( mavlink_mission_request_partial_list_t ),
typeof( mavlink_mission_write_partial_list_t ), typeof( mavlink_mission_item_t ),
typeof( mavlink_mission_request_t ), typeof( mavlink_mission_set_current_t ),
typeof( mavlink_mission_current_t ), typeof( mavlink_mission_request_list_t ),
typeof( mavlink_mission_count_t ), typeof( mavlink_mission_clear_all_t ), typeof(
mavlink_mission_item_reached_t ), typeof( mavlink_mission_ack_t ), typeof(
mavlink_set_gps_global_origin_t ), typeof( mavlink_gps_global_origin_t ), typeof(
mavlink_param_map_rc_t ), null, null, null, typeof(
mavlink_safety_set_allowed_area_t ), typeof( mavlink_safety_allowed_area_t ),
```

```
null, null, null, null, null, typeof( mavlink_attitude_quaternion_cov_t ),
typeof( mavlink_nav_controller_output_t ), typeof(
mavlink_global_position_int_cov_t ), typeof( mavlink_local_position_ned_cov_t ),
typeof( mavlink_rc_channels_t ), typeof( mavlink_request_data_stream_t ), typeof(
mavlink_data_stream_t ), null, typeof( mavlink_manual_control_t ), typeof(
mavlink_rc_channels_override_t ), null, null, typeof( mavlink_mission_item_int_t
), typeof( mavlink_vfr_hud_t ), typeof( mavlink_command_int_t ), typeof(
mavlink_command_long_t ), typeof( mavlink_command_ack_t ), null, null, null,
typeof( mavlink_manual_setpoint_t ), typeof( mavlink_set_attitude_target_t ),
typeof( mavlink_attitude_target_t ), typeof(
mavlink_set_position_target_local_ned_t ), typeof(
mavlink_position_target_local_ned_t ), typeof(
mavlink_set_position_target_global_int_t ), typeof(
mavlink_position_target_global_int_t ), null, typeof(
mavlink_local_position_ned_system_global_offset_t ), typeof( mavlink_hil_state_t
), typeof( mavlink_hil_controls_t ), typeof( mavlink_hil_rc_inputs_raw_t ), null,
null, null, null, null, null, null, typeof( mavlink_optical_flow_t ), typeof(
mavlink_global_vision_position_estimate_t ), typeof(
mavlink_vision_position_estimate_t ), typeof( mavlink_vision_speed_estimate_t ),
typeof( mavlink_vicon_position_estimate_t ), typeof( mavlink_highres_imu_t ),
typeof( mavlink_optical_flow_rad_t ), typeof( mavlink_hil_sensor_t ), typeof(
mavlink_sim_state_t ), typeof( mavlink_radio_status_t ), typeof(
mavlink_file_transfer_protocol_t ), typeof( mavlink_timesync_t ), null, typeof(
mavlink_hil_gps_t ), typeof( mavlink_hil_optical_flow_t ), typeof(
mavlink_hil_state_quaternion_t ), typeof( mavlink_scaled_imu2_t ), typeof(
mavlink_log_request_list_t ), typeof( mavlink_log_entry_t ), typeof(
mavlink_log_request_data_t ), typeof( mavlink_log_data_t ), typeof(
mavlink_log_erase_t ), typeof( mavlink_log_request_end_t ), typeof(
mavlink_gps_inject_data_t ), typeof( mavlink_gps2_raw_t ), typeof(
mavlink_power_status_t ), typeof( mavlink_serial_control_t ), typeof(
mavlink_gps_rtk_t ), typeof( mavlink_gps2_rtk_t ), typeof( mavlink_scaled_imu3_t
), typeof( mavlink_data_transmission_handshake_t ), typeof(
mavlink_encapsulated_data_t ), typeof( mavlink_distance_sensor_t ), typeof(
mavlink_terrain_request_t ), typeof( mavlink_terrain_data_t ), typeof(
mavlink_terrain_check_t ), typeof( mavlink_terrain_report_t ), typeof(
mavlink_scaled_pressure2_t ), typeof( mavlink_att_pos_mocap_t ), typeof(
mavlink_set_actuator_control_target_t ), typeof(
mavlink_actuator_control_target_t ), null, null, typeof(
mavlink_scaled_pressure3_t ), null, null, null, typeof( mavlink_battery_status_t
), typeof( mavlink_autopilot_version_t ), typeof( mavlink_landing_target_t ),
typeof( mavlink_sensor_offsets_t ), typeof( mavlink_set_mag_offsets_t ), typeof(
mavlink_meminfo_t ), typeof( mavlink_ap_adc_t ), typeof(
mavlink_digicam_configure_t ), typeof( mavlink_digicam_control_t ), typeof(
mavlink_mount_configure_t ), typeof( mavlink_mount_control_t ), typeof(
mavlink_mount_status_t ), null, typeof( mavlink_fence_point_t ), typeof(
mavlink_fence_fetch_point_t ), typeof( mavlink_fence_status_t ), typeof(
mavlink_ahrs_t ), typeof( mavlink_simstate_t ), typeof( mavlink_hwstatus_t ),
typeof( mavlink_radio_t ), typeof( mavlink_limits_status_t ), typeof(
mavlink_wind_t ), typeof( mavlink_data16_t ), typeof( mavlink_data32_t ), typeof(
mavlink_data64_t ), typeof( mavlink_data96_t ), typeof( mavlink_rangefinder_t ),
typeof( mavlink_airspeed_autocal_t ), typeof( mavlink_rally_point_t ), typeof(
mavlink_rally_fetch_point_t ), typeof( mavlink_compassmot_status_t ), typeof(
mavlink_ahrs2_t ), typeof( mavlink_camera_status_t ), typeof(
mavlink_camera_feedback_t ), typeof( mavlink_battery2_t ), typeof(
mavlink_ahrs3_t ), typeof( mavlink_autopilot_version_request_t ), typeof(
mavlink_remote_log_data_block_t ), typeof( mavlink_remote_log_block_status_t ),
typeof( mavlink_led_control_t ), null, null, null, null, typeof(
mavlink_mag_cal_progress_t ), typeof( mavlink_mag_cal_report_t ), typeof(
mavlink_ekf_status_report_t ), typeof( mavlink_pid_tuning_t ), null, null, null,
null, null, typeof( mavlink_gimbal_report_t ), typeof( mavlink_gimbal_control_t
), typeof( mavlink_gimbal_reset_t ), typeof(
mavlink_gimbal_axis_calibration_progress_t ), typeof(
```

```
mavlink_gimbal_set_home_offsets_t ), typeof(
mavlink_gimbal_home_offset_calibration_result_t ), typeof(
mavlink_gimbal_set_factory_parameters_t ), typeof(
mavlink_gimbal_factory_parameters_loaded_t ), typeof(
mavlink_gimbal_erase_firmware_and_config_t ), typeof(
mavlink_gimbal_perform_factory_tests_t ), typeof(
mavlink_gimbal_report_factory_tests_progress_t ), typeof(
mavlink_gimbal_request_axis_calibration_status_t ), typeof(
mavlink_gimbal_report_axis_calibration_status_t ), typeof(
mavlink_gimbal_request_axis_calibration_t ), null, typeof(
mavlink_gopro_heartbeat_t ), typeof( mavlink_gopro_get_request_t ), typeof(
mavlink_gopro_get_response_t ), typeof( mavlink_gopro_set_request_t ), typeof(
mavlink_gopro_set_response_t ), null, null, null, null, null, null, typeof(
mavlink_rpm_t ), null, null, null, null, null, null, null, null, null, null,
null, null, null, null, typeof( mavlink_vibration_t ), typeof(
mavlink_home_position_t ), typeof( mavlink_set_home_position_t ), null, null,
typeof( mavlink_adsb_vehicle_t ), null, typeof( mavlink_v2_extension_t ), typeof(
mavlink_memory_vect_t ), typeof( mavlink_debug_vect_t ), typeof(
mavlink_named_value_float_t ), typeof( mavlink_named_value_int_t ), typeof(
mavlink_statustext_t ), typeof( mavlink_debug_t ), null};

            public static readonly string[] MAVLINK_NAMES = new string[]
{"HEARTBEAT", "SYS_STATUS", "SYSTEM_TIME", null, "PING",
"CHANGE_OPERATOR_CONTROL", "CHANGE_OPERATOR_CONTROL_ACK", "AUTH_KEY", null, null,
null, "SET_MODE", null, null, null, null, null, null, null, null,
"PARAM_REQUEST_READ", "PARAM_REQUEST_LIST", "PARAM_VALUE", "PARAM_SET",
"GPS_RAW_INT", "GPS_STATUS", "SCALED_IMU", "RAW_IMU", "RAW_PRESSURE",
"SCALED_PRESSURE", "ATTITUDE", "ATTITUDE_QUATERNION", "LOCAL_POSITION_NED",
"GLOBAL_POSITION_INT", "RC_CHANNELS_SCALED", "RC_CHANNELS_RAW",
"SERVO_OUTPUT_RAW", "MISSION_REQUEST_PARTIAL_LIST", "MISSION_WRITE_PARTIAL_LIST",
"MISSION_ITEM", "MISSION_REQUEST", "MISSION_SET_CURRENT", "MISSION_CURRENT",
"MISSION_REQUEST_LIST", "MISSION_COUNT", "MISSION_CLEAR_ALL",
"MISSION_ITEM_REACHED", "MISSION_ACK", "SET_GPS_GLOBAL_ORIGIN",
"GPS_GLOBAL_ORIGIN", "PARAM_MAP_RC", null, null, null, "SAFETY_SET_ALLOWED_AREA",
"SAFETY_ALLOWED_AREA", null, null, null, null, null, "ATTITUDE_QUATERNION_COV",
"NAV_CONTROLLER_OUTPUT", "GLOBAL_POSITION_INT_COV", "LOCAL_POSITION_NED_COV",
"RC_CHANNELS", "REQUEST_DATA_STREAM", "DATA_STREAM", null, "MANUAL_CONTROL",
"RC_CHANNELS_OVERRIDE", null, null, "MISSION_ITEM_INT", "VFR_HUD", "COMMAND_INT",
"COMMAND_LONG", "COMMAND_ACK", null, null, null, "MANUAL_SETPOINT",
"SET_ATTITUDE_TARGET", "ATTITUDE_TARGET", "SET_POSITION_TARGET_LOCAL_NED",
"POSITION_TARGET_LOCAL_NED", "SET_POSITION_TARGET_GLOBAL_INT",
"POSITION_TARGET_GLOBAL_INT", null, "LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET",
"HIL_STATE", "HIL_CONTROLS", "HIL_RC_INPUTS_RAW", null, null, null, null, null,
null, null, "OPTICAL_FLOW", "GLOBAL_VISION_POSITION_ESTIMATE",
"VISION_POSITION_ESTIMATE", "VISION_SPEED_ESTIMATE", "VICON_POSITION_ESTIMATE",
"HIGHRES_IMU", "OPTICAL_FLOW_RAD", "HIL_SENSOR", "SIM_STATE", "RADIO_STATUS",
"FILE_TRANSFER_PROTOCOL", "TIMESYNC", null, "HIL_GPS", "HIL_OPTICAL_FLOW",
"HIL_STATE_QUATERNION", "SCALED_IMU2", "LOG_REQUEST_LIST", "LOG_ENTRY",
"LOG_REQUEST_DATA", "LOG_DATA", "LOG_ERASE", "LOG_REQUEST_END",
"GPS_INJECT_DATA", "GPS2_RAW", "POWER_STATUS", "SERIAL_CONTROL", "GPS_RTK",
"GPS2_RTK", "SCALED_IMU3", "DATA_TRANSMISSION_HANDSHAKE", "ENCAPSULATED_DATA",
"DISTANCE_SENSOR", "TERRAIN_REQUEST", "TERRAIN_DATA", "TERRAIN_CHECK",
"TERRAIN_REPORT", "SCALED_PRESSURE2", "ATT_POS_MOCAP",
"SET_ACTUATOR_CONTROL_TARGET", "ACTUATOR_CONTROL_TARGET", null, null,
"SCALED_PRESSURE3", null, null, null, "BATTERY_STATUS", "AUTOPILOT_VERSION",
"LANDING_TARGET", "SENSOR_OFFSETS", "SET_MAG_OFFSETS", "MEMINFO", "AP_ADC",
"DIGICAM_CONFIGURE", "DIGICAM_CONTROL", "MOUNT_CONFIGURE", "MOUNT_CONTROL",
"MOUNT_STATUS", null, "FENCE_POINT", "FENCE_FETCH_POINT", "FENCE_STATUS", "AHRS",
"SIMSTATE", "HWSTATUS", "RADIO", "LIMITS_STATUS", "WIND", "DATA16", "DATA32",
"DATA64", "DATA96", "RANGEFINDER", "AIRSPEED_AUTOCAL", "RALLY_POINT",
"RALLY_FETCH_POINT", "COMPASSMOT_STATUS", "AHRS2", "CAMERA_STATUS",
"CAMERA_FEEDBACK", "BATTERY2", "AHRS3", "AUTOPILOT_VERSION_REQUEST",
```

```
"REMOTE_LOG_DATA_BLOCK", "REMOTE_LOG_BLOCK_STATUS", "LED_CONTROL", null, null,
null, null, "MAG_CAL_PROGRESS", "MAG_CAL_REPORT", "EKF_STATUS_REPORT",
"PID_TUNING", null, null, null, null, null, "GIMBAL_REPORT", "GIMBAL_CONTROL",
"GIMBAL_RESET", "GIMBAL_AXIS_CALIBRATION_PROGRESS", "GIMBAL_SET_HOME_OFFSETS",
"GIMBAL_HOME_OFFSET_CALIBRATION_RESULT", "GIMBAL_SET_FACTORY_PARAMETERS",
"GIMBAL_FACTORY_PARAMETERS_LOADED", "GIMBAL_ERASE_FIRMWARE_AND_CONFIG",
"GIMBAL_PERFORM_FACTORY_TESTS", "GIMBAL_REPORT_FACTORY_TESTS_PROGRESS",
"GIMBAL_REQUEST_AXIS_CALIBRATION_STATUS",
"GIMBAL_REPORT_AXIS_CALIBRATION_STATUS", "GIMBAL_REQUEST_AXIS_CALIBRATION", null,
"GOPRO_HEARTBEAT", "GOPRO_GET_REQUEST", "GOPRO_GET_RESPONSE",
"GOPRO_SET_REQUEST", "GOPRO_SET_RESPONSE", null, null, null, null, null, null,
"RPM", null, null, null, null, null, null, null, null, null, null, null, null,
null, null, "VIBRATION", "HOME_POSITION", "SET_HOME_POSITION", null, null,
"ADSB_VEHICLE", null, "V2_EXTENSION", "MEMORY_VECT", "DEBUG_VECT",
"NAMED_VALUE_FLOAT", "NAMED_VALUE_INT", "STATUSTEXT", "DEBUG", null};

        public const byte MAVLINK_VERSION = 2;

             public enum MAVLINK_MSG_ID
             {
                   HEARTBEAT = 0,
SYS_STATUS = 1,
SYSTEM_TIME = 2,
PING = 4,
CHANGE_OPERATOR_CONTROL = 5,
CHANGE_OPERATOR_CONTROL_ACK = 6,
AUTH_KEY = 7,
SET_MODE = 11,
PARAM_REQUEST_READ = 20,
PARAM_REQUEST_LIST = 21,
PARAM_VALUE = 22,
PARAM_SET = 23,
GPS_RAW_INT = 24,
GPS_STATUS = 25,
SCALED_IMU = 26,
RAW_IMU = 27,
RAW_PRESSURE = 28,
SCALED_PRESSURE = 29,
ATTITUDE = 30,
ATTITUDE_QUATERNION = 31,
LOCAL_POSITION_NED = 32,
GLOBAL_POSITION_INT = 33,
RC_CHANNELS_SCALED = 34,
RC_CHANNELS_RAW = 35,
SERVO_OUTPUT_RAW = 36,
MISSION_REQUEST_PARTIAL_LIST = 37,
MISSION_WRITE_PARTIAL_LIST = 38,
MISSION_ITEM = 39,
MISSION_REQUEST = 40,
MISSION_SET_CURRENT = 41,
MISSION_CURRENT = 42,
MISSION_REQUEST_LIST = 43,
MISSION_COUNT = 44,
MISSION_CLEAR_ALL = 45,
MISSION_ITEM_REACHED = 46,
MISSION_ACK = 47,
SET_GPS_GLOBAL_ORIGIN = 48,
GPS_GLOBAL_ORIGIN = 49,
PARAM_MAP_RC = 50,
SAFETY_SET_ALLOWED_AREA = 54,
SAFETY_ALLOWED_AREA = 55,
ATTITUDE_QUATERNION_COV = 61,
```

```
NAV_CONTROLLER_OUTPUT = 62,
GLOBAL_POSITION_INT_COV = 63,
LOCAL_POSITION_NED_COV = 64,
RC_CHANNELS = 65,
REQUEST_DATA_STREAM = 66,
DATA_STREAM = 67,
MANUAL_CONTROL = 69,
RC_CHANNELS_OVERRIDE = 70,
MISSION_ITEM_INT = 73,
VFR_HUD = 74,
COMMAND_INT = 75,
COMMAND_LONG = 76,
COMMAND_ACK = 77,
MANUAL_SETPOINT = 81,
SET_ATTITUDE_TARGET = 82,
ATTITUDE_TARGET = 83,
SET_POSITION_TARGET_LOCAL_NED = 84,
POSITION_TARGET_LOCAL_NED = 85,
SET_POSITION_TARGET_GLOBAL_INT = 86,
POSITION_TARGET_GLOBAL_INT = 87,
LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET = 89,
HIL_STATE = 90,
HIL_CONTROLS = 91,
HIL_RC_INPUTS_RAW = 92,
OPTICAL_FLOW = 100,
GLOBAL_VISION_POSITION_ESTIMATE = 101,
VISION_POSITION_ESTIMATE = 102,
VISION_SPEED_ESTIMATE = 103,
VICON_POSITION_ESTIMATE = 104,
HIGHRES_IMU = 105,
OPTICAL_FLOW_RAD = 106,
HIL_SENSOR = 107,
SIM_STATE = 108,
RADIO_STATUS = 109,
FILE_TRANSFER_PROTOCOL = 110,
TIMESYNC = 111,
HIL_GPS = 113,
HIL_OPTICAL_FLOW = 114,
HIL_STATE_QUATERNION = 115,
SCALED_IMU2 = 116,
LOG_REQUEST_LIST = 117,
LOG_ENTRY = 118,
LOG_REQUEST_DATA = 119,
LOG_DATA = 120,
LOG_ERASE = 121,
LOG_REQUEST_END = 122,
GPS_INJECT_DATA = 123,
GPS2_RAW = 124,
POWER_STATUS = 125,
SERIAL_CONTROL = 126,
GPS_RTK = 127,
GPS2_RTK = 128,
SCALED_IMU3 = 129,
DATA_TRANSMISSION_HANDSHAKE = 130,
ENCAPSULATED_DATA = 131,
DISTANCE_SENSOR = 132,
TERRAIN_REQUEST = 133,
TERRAIN_DATA = 134,
TERRAIN_CHECK = 135,
TERRAIN_REPORT = 136,
SCALED_PRESSURE2 = 137,
ATT_POS_MOCAP = 138,
```

```
SET_ACTUATOR_CONTROL_TARGET = 139,
ACTUATOR_CONTROL_TARGET = 140,
SCALED_PRESSURE3 = 143,
BATTERY_STATUS = 147,
AUTOPILOT_VERSION = 148,
LANDING_TARGET = 149,
SENSOR_OFFSETS = 150,
SET_MAG_OFFSETS = 151,
MEMINFO = 152,
AP_ADC = 153,
DIGICAM_CONFIGURE = 154,
DIGICAM_CONTROL = 155,
MOUNT_CONFIGURE = 156,
MOUNT_CONTROL = 157,
MOUNT_STATUS = 158,
FENCE_POINT = 160,
FENCE_FETCH_POINT = 161,
FENCE_STATUS = 162,
AHRS = 163,
SIMSTATE = 164,
HWSTATUS = 165,
RADIO = 166,
LIMITS_STATUS = 167,
WIND = 168,
DATA16 = 169,
DATA32 = 170,
DATA64 = 171,
DATA96 = 172,
RANGEFINDER = 173,
AIRSPEED_AUTOCAL = 174,
RALLY_POINT = 175,
RALLY_FETCH_POINT = 176,
COMPASSMOT_STATUS = 177,
AHRS2 = 178,
CAMERA_STATUS = 179,
CAMERA_FEEDBACK = 180,
BATTERY2 = 181,
AHRS3 = 182,
AUTOPILOT_VERSION_REQUEST = 183,
REMOTE_LOG_DATA_BLOCK = 184,
REMOTE_LOG_BLOCK_STATUS = 185,
LED_CONTROL = 186,
MAG_CAL_PROGRESS = 191,
MAG_CAL_REPORT = 192,
EKF_STATUS_REPORT = 193,
PID_TUNING = 194,
GIMBAL_REPORT = 200,
GIMBAL_CONTROL = 201,
GIMBAL_RESET = 202,
GIMBAL_AXIS_CALIBRATION_PROGRESS = 203,
GIMBAL_SET_HOME_OFFSETS = 204,
GIMBAL_HOME_OFFSET_CALIBRATION_RESULT = 205,
GIMBAL_SET_FACTORY_PARAMETERS = 206,
GIMBAL_FACTORY_PARAMETERS_LOADED = 207,
GIMBAL_ERASE_FIRMWARE_AND_CONFIG = 208,
GIMBAL_PERFORM_FACTORY_TESTS = 209,
GIMBAL_REPORT_FACTORY_TESTS_PROGRESS = 210,
GIMBAL_REQUEST_AXIS_CALIBRATION_STATUS = 211,
GIMBAL_REPORT_AXIS_CALIBRATION_STATUS = 212,
GIMBAL_REQUEST_AXIS_CALIBRATION = 213,
GOPRO_HEARTBEAT = 215,
GOPRO_GET_REQUEST = 216,
```

```
GOPRO_GET_RESPONSE = 217,
GOPRO_SET_REQUEST = 218,
GOPRO_SET_RESPONSE = 219,
RPM = 226,
VIBRATION = 241,
HOME_POSITION = 242,
SET_HOME_POSITION = 243,
ADSB_VEHICLE = 246,
V2_EXTENSION = 248,
MEMORY_VECT = 249,
DEBUG_VECT = 250,
NAMED_VALUE_FLOAT = 251,
NAMED_VALUE_INT = 252,
STATUSTEXT = 253,
DEBUG = 254,

             }



        ///<summary>  </summary>
        public enum MAV_CMD
        {
       ///<summary> Navigate to MISSION. |Hold time in decimal seconds. (ignored
by fixed wing, time to stay at MISSION for rotary wing)| Acceptance radius in
meters (if the sphere with this radius is hit, the MISSION counts as reached)| 0
to pass through the WP, if > 0 radius in meters to pass by WP. Positive value for
clockwise orbit, negative value for counter-clockwise orbit. Allows trajectory
control.| Desired yaw angle at MISSION (rotary wing)| Latitude| Longitude|
Altitude|  </summary>
            WAYPOINT=16,
              ///<summary> Loiter around this MISSION an unlimited amount of time
|Empty| Empty| Radius around MISSION, in meters. If positive loiter clockwise,
else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|
</summary>
            LOITER_UNLIM=17,
              ///<summary> Loiter around this MISSION for X turns |Turns| Empty|
Radius around MISSION, in meters. If positive loiter clockwise, else counter-
clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|  </summary>
            LOITER_TURNS=18,
              ///<summary> Loiter around this MISSION for X seconds |Seconds
(decimal)| Empty| Radius around MISSION, in meters. If positive loiter clockwise,
else counter-clockwise| Desired yaw angle.| Latitude| Longitude| Altitude|
</summary>
            LOITER_TIME=19,
              ///<summary> Return to launch location |Empty| Empty| Empty| Empty|
Empty| Empty| Empty|  </summary>
            RETURN_TO_LAUNCH=20,
              ///<summary> Land at location |Abort Alt| Empty| Empty| Desired yaw
angle.| Latitude| Longitude| Altitude|  </summary>
            LAND=21,
              ///<summary> Takeoff from ground / hand |Minimum pitch (if airspeed
sensor present), desired pitch without sensor| Empty| Empty| Yaw angle (if
magnetometer present), ignored without magnetometer| Latitude| Longitude|
Altitude|  </summary>
            TAKEOFF=22,
              ///<summary> Continue on the current course and climb/descend to
specified altitude.  When the altitude is reached continue to the next command
(i.e., don't proceed to the next command until the desired altitude is reached.
|Empty| Empty| Empty| Empty| Empty| Empty| Desired altitude in meters|
</summary>
            CONTINUE_AND_CHANGE_ALT=30,
```

```
            ///<summary> Begin loiter at the specified Latitude and Longitude.
If Lat=Lon=0, then loiter at the current position.  Don't consider the navigation
command complete (don't leave loiter) until the altitude has been reached.
Additionally, if the Heading Required parameter is non-zero the  aircraft will
not leave the loiter until heading toward the next waypoint.  |Heading Required
(0 = False)| Radius in meters. If positive loiter clockwise, negative counter-
clockwise, 0 means no change to standard loiter.| Empty| Empty| Latitude|
Longitude| Altitude|  </summary>
            LOITER_TO_ALT=31,
            ///<summary> Sets the region of interest (ROI) for a sensor set or
the vehicle itself. This can then be used by the vehicles control system to
control the vehicle attitude and the attitude of various sensors such as cameras.
|Region of intereset mode. (see MAV_ROI enum)| MISSION index/ target ID. (see
MAV_ROI enum)| ROI index (allows a vehicle to manage multiple ROI's)| Empty| x
the location of the fixed ROI (see MAV_FRAME)| y| z|  </summary>
            ROI=80,
            ///<summary> Control autonomous path planning on the MAV. |0:
Disable local obstacle avoidance / local path planning (without resetting map),
1: Enable local path planning, 2: Enable and reset local path planning| 0:
Disable full path planning (without resetting map), 1: Enable, 2: Enable and
reset map/occupancy grid, 3: Enable and reset planned route, but not occupancy
grid| Empty| Yaw angle at goal, in compass degrees, [0..360]| Latitude/X of goal|
Longitude/Y of goal| Altitude/Z of goal|  </summary>
            PATHPLANNING=81,
            ///<summary> Navigate to MISSION using a spline path. |Hold time in
decimal seconds. (ignored by fixed wing, time to stay at MISSION for rotary
wing)| Empty| Empty| Empty| Latitude/X of goal| Longitude/Y of goal| Altitude/Z
of goal|  </summary>
            SPLINE_WAYPOINT=82,
            ///<summary> Mission command to wait for an altitude or downwards
vertical speed. This is meant for high altitude balloon launches, allowing the
aircraft to be idle until either an altitude is reached or a negative vertical
speed is reached (indicating early balloon burst). The wiggle time is how often
to wiggle the control surfaces to prevent them seizing up. |altitude (m)| descent
speed (m/s)| Wiggle Time (s)| Empty| Empty| Empty| Empty|  </summary>
            ALTITUDE_WAIT=83,
            ///<summary> hand control over to an external controller |On / Off
(> 0.5f on)| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            GUIDED_ENABLE=92,
            ///<summary> NOP - This command is only used to mark the upper
limit of the NAV/ACTION commands in the enumeration |Empty| Empty| Empty| Empty|
Empty| Empty| Empty|  </summary>
            LAST=95,
            ///<summary> Delay mission state machine. |Delay in seconds
(decimal)| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            CONDITION_DELAY=112,
            ///<summary> Ascend/descend at rate.  Delay mission state machine
until desired altitude reached. |Descent / Ascend rate (m/s)| Empty| Empty|
Empty| Empty| Empty| Finish Altitude|  </summary>
            CONDITION_CHANGE_ALT=113,
            ///<summary> Delay mission state machine until within desired
distance of next NAV point. |Distance (meters)| Empty| Empty| Empty| Empty|
Empty| Empty|  </summary>
            CONDITION_DISTANCE=114,
            ///<summary> Reach a certain target angle. |target angle: [0-360],
0 is north| speed during yaw change:[deg per second]| direction: negative:
counter clockwise, positive: clockwise [-1,1]| relative offset or absolute angle:
[ 1,0]| Empty| Empty| Empty|  </summary>
            CONDITION_YAW=115,
            ///<summary> NOP - This command is only used to mark the upper
limit of the CONDITION commands in the enumeration |Empty| Empty| Empty| Empty|
Empty| Empty| Empty|  </summary>
```

```
            CONDITION_LAST=159,
              ///<summary> Set system mode. |Mode, as defined by ENUM MAV_MODE|
Custom mode - this is system specific, please refer to the individual autopilot
specifications for details.| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_SET_MODE=176,
              ///<summary> Jump to the desired command in the mission list.
Repeat this action only the specified number of times |Sequence number| Repeat
count| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_JUMP=177,
              ///<summary> Change speed and/or throttle set points. |Speed type
(0=Airspeed, 1=Ground Speed)| Speed  (m/s, -1 indicates no change)| Throttle  (
Percent, -1 indicates no change)| Empty| Empty| Empty| Empty|  </summary>
            DO_CHANGE_SPEED=178,
              ///<summary> Changes the home location either to the current
location or a specified location. |Use current (1=use current location, 0=use
specified location)| Empty| Empty| Empty| Latitude| Longitude| Altitude|
</summary>
            DO_SET_HOME=179,
              ///<summary> Set a system parameter.  Caution!  Use of this command
requires knowledge of the numeric enumeration value of the parameter. |Parameter
number| Parameter value| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_SET_PARAMETER=180,
              ///<summary> Set a relay to a condition. |Relay number| Setting
(1=on, 0=off, others possible depending on system hardware)| Empty| Empty| Empty|
Empty| Empty|  </summary>
            DO_SET_RELAY=181,
              ///<summary> Cycle a relay on and off for a desired number of cyles
with a desired period. |Relay number| Cycle count| Cycle time (seconds, decimal)|
Empty| Empty| Empty| Empty|  </summary>
            DO_REPEAT_RELAY=182,
              ///<summary> Set a servo to a desired PWM value. |Servo number| PWM
(microseconds, 1000 to 2000 typical)| Empty| Empty| Empty| Empty| Empty|
</summary>
            DO_SET_SERVO=183,
              ///<summary> Cycle a between its nominal setting and a desired PWM
for a desired number of cycles with a desired period. |Servo number| PWM
(microseconds, 1000 to 2000 typical)| Cycle count| Cycle time (seconds)| Empty|
Empty| Empty|  </summary>
            DO_REPEAT_SERVO=184,
              ///<summary> Terminate flight immediately |Flight termination
activated if > 0.5| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_FLIGHTTERMINATION=185,
              ///<summary> Mission command to perform a landing. This is used as
a marker in a mission to tell the autopilot where a sequence of mission items
that represents a landing starts. It may also be sent via a COMMAND_LONG to
trigger a landing, in which case the nearest (geographically) landing sequence in
the mission will be used. The Latitude/Longitude is optional, and may be set to
0/0 if not needed. If specified then it will be used to help find the closest
landing sequence. |Empty| Empty| Empty| Empty| Latitude| Longitude| Empty|
</summary>
            DO_LAND_START=189,
              ///<summary> Mission command to perform a landing from a rally
point. |Break altitude (meters)| Landing speed (m/s)| Empty| Empty| Empty| Empty|
Empty|  </summary>
            DO_RALLY_LAND=190,
              ///<summary> Mission command to safely abort an autonmous landing.
|Altitude (meters)| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_GO_AROUND=191,
              ///<summary> Control onboard camera system. |Camera ID (-1 for
all)| Transmission: 0: disabled, 1: enabled compressed, 2: enabled raw|
Transmission mode: 0: video stream, >0: single images every n seconds (decimal)|
```

```
Recording: 0: disabled, 1: enabled compressed, 2: enabled raw| Empty| Empty|
Empty|  </summary>
            DO_CONTROL_VIDEO=200,
             ///<summary> Sets the region of interest (ROI) for a sensor set or
the vehicle itself. This can then be used by the vehicles control system to
control the vehicle attitude and the attitude of various sensors such as cameras.
|Region of intereset mode. (see MAV_ROI enum)| MISSION index/ target ID. (see
MAV_ROI enum)| ROI index (allows a vehicle to manage multiple ROI's)| Empty| x
the location of the fixed ROI (see MAV_FRAME)| y| z|  </summary>
            DO_SET_ROI=201,
             ///<summary> Mission command to configure an on-board camera
controller system. |Modes: P, TV, AV, M, Etc| Shutter speed: Divisor number for
one second| Aperture: F stop number| ISO number e.g. 80, 100, 200, Etc| Exposure
type enumerator| Command Identity| Main engine cut-off time before camera trigger
in seconds/10 (0 means no cut-off)|  </summary>
            DO_DIGICAM_CONFIGURE=202,
             ///<summary> Mission command to control an on-board camera
controller system. |Session control e.g. show/hide lens| Zoom's absolute
position| Zooming step value to offset zoom from the current position| Focus
Locking, Unlocking or Re-locking| Shooting Command| Command Identity| Empty|
</summary>
            DO_DIGICAM_CONTROL=203,
             ///<summary> Mission command to configure a camera or antenna mount
|Mount operation mode (see MAV_MOUNT_MODE enum)| stabilize roll? (1 = yes, 0 =
no)| stabilize pitch? (1 = yes, 0 = no)| stabilize yaw? (1 = yes, 0 = no)| Empty|
Empty| Empty|  </summary>
            DO_MOUNT_CONFIGURE=204,
             ///<summary> Mission command to control a camera or antenna mount
|pitch or lat in degrees, depending on mount mode.| roll or lon in degrees
depending on mount mode| yaw or alt (in meters) depending on mount mode|
reserved| reserved| reserved| MAV_MOUNT_MODE enum value|  </summary>
            DO_MOUNT_CONTROL=205,
             ///<summary> Mission command to set CAM_TRIGG_DIST for this flight
|Camera trigger distance (meters)| Empty| Empty| Empty| Empty| Empty| Empty|
</summary>
            DO_SET_CAM_TRIGG_DIST=206,
             ///<summary> Mission command to enable the geofence |enable?
(0=disable, 1=enable)| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_FENCE_ENABLE=207,
             ///<summary> Mission command to trigger a parachute |action
(0=disable, 1=enable, 2=release, for some systems see PARACHUTE_ACTION enum, not
in general message set.)| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_PARACHUTE=208,
             ///<summary> Mission command to perform motor test |motor sequence
number (a number from 1 to max number of motors on the vehicle)| throttle type
(0=throttle percentage, 1=PWM, 2=pilot throttle channel pass-through. See
MOTOR_TEST_THROTTLE_TYPE enum)| throttle| timeout (in seconds)| Empty| Empty|
Empty|  </summary>
            DO_MOTOR_TEST=209,
             ///<summary> Change to/from inverted flight |inverted (0=normal,
1=inverted)| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_INVERTED_FLIGHT=210,
             ///<summary> Mission command to operate EPM gripper |gripper number
(a number from 1 to max number of grippers on the vehicle)| gripper action
(0=release, 1=grab. See GRIPPER_ACTIONS enum)| Empty| Empty| Empty| Empty| Empty|
</summary>
            DO_GRIPPER=211,
             ///<summary> Enable/disable autotune |enable (1: enable,
0:disable)| Empty| Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_AUTOTUNE_ENABLE=212,
             ///<summary> Mission command to control a camera or antenna mount,
using a quaternion as reference. |q1 - quaternion param #1, w (1 in null-
```

```
rotation)| q2 - quaternion param #2, x (0 in null-rotation)| q3 - quaternion
param #3, y (0 in null-rotation)| q4 - quaternion param #4, z (0 in null-
rotation)| Empty| Empty| Empty|  </summary>
            DO_MOUNT_CONTROL_QUAT=220,
              ///<summary> set id of master controller |System ID| Component ID|
Empty| Empty| Empty| Empty| Empty|  </summary>
            DO_GUIDED_MASTER=221,
              ///<summary> set limits for external control |timeout - maximum
time (in seconds) that external controller will be allowed to control vehicle. 0
means no timeout| absolute altitude min (in meters, AMSL) - if vehicle moves
below this alt, the command will be aborted and the mission will continue.  0
means no lower altitude limit| absolute altitude max (in meters)- if vehicle
moves above this alt, the command will be aborted and the mission will continue.
0 means no upper altitude limit| horizontal move limit (in meters, AMSL) - if
vehicle moves more than this distance from it's location at the moment the
command was executed, the command will be aborted and the mission will continue.
0 means no horizontal altitude limit| Empty| Empty| Empty|  </summary>
            DO_GUIDED_LIMITS=222,
              ///<summary> NOP - This command is only used to mark the upper
limit of the DO commands in the enumeration |Empty| Empty| Empty| Empty| Empty|
Empty| Empty|  </summary>
            DO_LAST=240,
              ///<summary> Trigger calibration. This command will be only
accepted if in pre-flight mode. |Gyro calibration: 0: no, 1: yes| Magnetometer
calibration: 0: no, 1: yes| Ground pressure: 0: no, 1: yes| Radio calibration: 0:
no, 1: yes| Accelerometer calibration: 0: no, 1: yes| Compass/Motor interference
calibration: 0: no, 1: yes| Empty|  </summary>
            PREFLIGHT_CALIBRATION=241,
              ///<summary> Set sensor offsets. This command will be only accepted
if in pre-flight mode. |Sensor to adjust the offsets for: 0: gyros, 1:
accelerometer, 2: magnetometer, 3: barometer, 4: optical flow, 5: second
magnetometer| X axis offset (or generic dimension 1), in the sensor's raw units|
Y axis offset (or generic dimension 2), in the sensor's raw units| Z axis offset
(or generic dimension 3), in the sensor's raw units| Generic dimension 4, in the
sensor's raw units| Generic dimension 5, in the sensor's raw units| Generic
dimension 6, in the sensor's raw units|  </summary>
            PREFLIGHT_SET_SENSOR_OFFSETS=242,
              ///<summary> Request storage of different parameter values and
logs. This command will be only accepted if in pre-flight mode. |Parameter
storage: 0: READ FROM FLASH/EEPROM, 1: WRITE CURRENT TO FLASH/EEPROM| Mission
storage: 0: READ FROM FLASH/EEPROM, 1: WRITE CURRENT TO FLASH/EEPROM| Reserved|
Reserved| Empty| Empty| Empty|  </summary>
            PREFLIGHT_STORAGE=245,
              ///<summary> Request the reboot or shutdown of system components.
|0: Do nothing for autopilot, 1: Reboot autopilot, 2: Shutdown autopilot.| 0: Do
nothing for onboard computer, 1: Reboot onboard computer, 2: Shutdown onboard
computer.| Reserved| Reserved| Empty| Empty| Empty|  </summary>
            PREFLIGHT_REBOOT_SHUTDOWN=246,
              ///<summary> Hold / continue the current action |MAV_GOTO_DO_HOLD:
hold MAV_GOTO_DO_CONTINUE: continue with next item in mission plan|
MAV_GOTO_HOLD_AT_CURRENT_POSITION: Hold at current position
MAV_GOTO_HOLD_AT_SPECIFIED_POSITION: hold at specified position| MAV_FRAME
coordinate frame of hold point| Desired yaw angle in degrees| Latitude / X
position| Longitude / Y position| Altitude / Z position|  </summary>
            OVERRIDE_GOTO=252,
              ///<summary> start running a mission |first_item: the first mission
item to run| last_item:  the last mission item to run (after this item is run,
the mission ends)|  </summary>
            MISSION_START=300,
              ///<summary> Arms / Disarms a component |1 to arm, 0 to disarm| 0
to disarm if landed, 21196 to force disarm any time|  </summary>
            COMPONENT_ARM_DISARM=400,
```

```
            ///<summary> Request the home position from the vehicle. |Reserved|
Reserved| Reserved| Reserved| Reserved| Reserved| Reserved|  </summary>
            GET_HOME_POSITION=410,
            ///<summary> Starts receiver pairing |0:Spektrum| 0:Spektrum DSM2,
1:Spektrum DSMX|  </summary>
            START_RX_PAIR=500,
            ///<summary> Request autopilot capabilities |1: Request autopilot
version| Reserved (all remaining params)|  </summary>
            REQUEST_AUTOPILOT_CAPABILITIES=520,
            ///<summary> Start image capture sequence |Duration between two
consecutive pictures (in seconds)| Number of images to capture total - 0 for
unlimited capture| Resolution in megapixels (0.3 for 640x480, 1.3 for 1280x720,
etc)|  </summary>
            IMAGE_START_CAPTURE=2000,
            ///<summary> Stop image capture sequence |Reserved| Reserved|
</summary>
            IMAGE_STOP_CAPTURE=2001,
            ///<summary> Starts video capture |Camera ID (0 for all cameras), 1
for first, 2 for second, etc.| Frames per second| Resolution in megapixels (0.3
for 640x480, 1.3 for 1280x720, etc)|  </summary>
            VIDEO_START_CAPTURE=2500,
            ///<summary> Stop the current video capture |Reserved| Reserved|
</summary>
            VIDEO_STOP_CAPTURE=2501,
            ///<summary> Create a panorama at the current position |Viewing
angle horizontal of the panorama (in degrees, +- 0.5 the total angle)| Viewing
angle vertical of panorama (in degrees)| Speed of the horizontal rotation (in
degrees per second)| Speed of the vertical rotation (in degrees per second)|
</summary>
            PANORAMA_CREATE=2800,
            ///<summary> Deploy payload on a Lat / Lon / Alt position. This
includes the navigation to reach the required release position and velocity.
|Operation mode. 0: prepare single payload deploy (overwriting previous
requests), but do not execute it. 1: execute payload deploy immediately
(rejecting further deploy commands during execution, but allowing abort). 2: add
payload deploy to existing deployment list.| Desired approach vector in degrees
compass heading (0..360). A negative value indicates the system can define the
approach vector at will.| Desired ground speed at release time. This can be
overriden by the airframe in case it needs to meet minimum airspeed. A negative
value indicates the system can define the ground speed at will.| Minimum altitude
clearance to the release position in meters. A negative value indicates the
system can define the clearance at will.| Latitude unscaled for MISSION_ITEM or
in 1e7 degrees for MISSION_ITEM_INT| Longitude unscaled for MISSION_ITEM or in
1e7 degrees for MISSION_ITEM_INT| Altitude, in meters AMSL|  </summary>
            PAYLOAD_PREPARE_DEPLOY=30001,
            ///<summary> Control the payload deployment. |Operation mode. 0:
Abort deployment, continue normal mission. 1: switch to payload deploment mode.
100: delete first payload deployment request. 101: delete all payload deployment
requests.| Reserved| Reserved| Reserved| Reserved| Reserved| Reserved|
</summary>
            PAYLOAD_CONTROL_DEPLOY=30002,
            ///<summary>  | </summary>
            ENUM_END=30003,
            ///<summary> Initiate a magnetometer calibration |uint8_t bitmask
of magnetometers (0 means all)| Automatically retry on failure (0=no retry,
1=retry).| Save without user input (0=require input, 1=autosave).| Delay
(seconds)| Autoreboot (0=user reboot, 1=autoreboot)| Empty| Empty|  </summary>
            DO_START_MAG_CAL=42424,
            ///<summary> Initiate a magnetometer calibration |uint8_t bitmask
of magnetometers (0 means all)| Empty| Empty| Empty| Empty| Empty| Empty|
</summary>
            DO_ACCEPT_MAG_CAL=42425,
```

```
            ///<summary> Cancel a running magnetometer calibration |uint8_t
bitmask of magnetometers (0 means all)| Empty| Empty| Empty| Empty| Empty| Empty|
</summary>
            DO_CANCEL_MAG_CAL=42426,
            ///<summary> Reply with the version banner |Empty| Empty| Empty|
Empty| Empty| Empty| Empty|  </summary>
            DO_SEND_BANNER=42428,

        };

        ///<summary>  </summary>
        public enum LIMITS_STATE
        {
       ///<summary>  pre-initialization | </summary>
            LIMITS_INIT=0,
             ///<summary>  disabled | </summary>
            LIMITS_DISABLED=1,
             ///<summary>  checking limits | </summary>
            LIMITS_ENABLED=2,
             ///<summary>  a limit has been breached | </summary>
            LIMITS_TRIGGERED=3,
             ///<summary>  taking action eg. RTL | </summary>
            LIMITS_RECOVERING=4,
             ///<summary>  we're no longer in breach of a limit | </summary>
            LIMITS_RECOVERED=5,
             ///<summary>  | </summary>
            ENUM_END=6,

        };

        ///<summary>  </summary>
        public enum LIMIT_MODULE
        {
       ///<summary>  pre-initialization | </summary>
            LIMIT_GPSLOCK=1,
             ///<summary>  disabled | </summary>
            LIMIT_GEOFENCE=2,
             ///<summary>  checking limits | </summary>
            LIMIT_ALTITUDE=4,
             ///<summary>  | </summary>
            ENUM_END=5,

        };

        ///<summary> Flags in RALLY_POINT message </summary>
        public enum RALLY_FLAGS
        {
       ///<summary> Flag set when requiring favorable winds for landing.  |
</summary>
            FAVORABLE_WIND=1,
             ///<summary> Flag set when plane is to immediately descend to break
altitude and land without GCS intervention.  Flag not set when plane is to loiter
at Rally point until commanded to land. | </summary>
            LAND_IMMEDIATELY=2,
             ///<summary>  | </summary>
            ENUM_END=3,

        };

        ///<summary>  </summary>
        public enum PARACHUTE_ACTION
        {
```

```
        ///<summary> Disable parachute release | </summary>
            PARACHUTE_DISABLE=0,
              ///<summary> Enable parachute release | </summary>
            PARACHUTE_ENABLE=1,
              ///<summary> Release parachute | </summary>
            PARACHUTE_RELEASE=2,
              ///<summary>  | </summary>
            ENUM_END=3,

         };

         ///<summary>  </summary>
         public enum MOTOR_TEST_THROTTLE_TYPE
         {
        ///<summary> throttle as a percentage from 0 ~ 100 | </summary>
            MOTOR_TEST_THROTTLE_PERCENT=0,
              ///<summary> throttle as an absolute PWM value (normally in range
of 1000~2000) | </summary>
            MOTOR_TEST_THROTTLE_PWM=1,
              ///<summary> throttle pass-through from pilot's transmitter |
</summary>
            MOTOR_TEST_THROTTLE_PILOT=2,
              ///<summary>  | </summary>
            ENUM_END=3,

         };

         ///<summary> Gripper actions. </summary>
         public enum GRIPPER_ACTIONS
         {
        ///<summary> gripper release of cargo | </summary>
            GRIPPER_ACTION_RELEASE=0,
              ///<summary> gripper grabs onto cargo | </summary>
            GRIPPER_ACTION_GRAB=1,
              ///<summary>  | </summary>
            ENUM_END=2,

         };

         ///<summary>  </summary>
         public enum CAMERA_STATUS_TYPES
         {
        ///<summary> Camera heartbeat, announce camera component ID at 1hz |
</summary>
            CAMERA_STATUS_TYPE_HEARTBEAT=0,
              ///<summary> Camera image triggered | </summary>
            CAMERA_STATUS_TYPE_TRIGGER=1,
              ///<summary> Camera connection lost | </summary>
            CAMERA_STATUS_TYPE_DISCONNECT=2,
              ///<summary> Camera unknown error | </summary>
            CAMERA_STATUS_TYPE_ERROR=3,
              ///<summary> Camera battery low. Parameter p1 shows reported
voltage | </summary>
            CAMERA_STATUS_TYPE_LOWBATT=4,
              ///<summary> Camera storage low. Parameter p1 shows reported shots
remaining | </summary>
            CAMERA_STATUS_TYPE_LOWSTORE=5,
              ///<summary> Camera storage low. Parameter p1 shows reported video
minutes remaining | </summary>
            CAMERA_STATUS_TYPE_LOWSTOREV=6,
              ///<summary>  | </summary>
            ENUM_END=7,
```

```
        };

        ///<summary>  </summary>
        public enum CAMERA_FEEDBACK_FLAGS
        {
       ///<summary> Shooting photos, not video | </summary>
            CAMERA_FEEDBACK_PHOTO=0,
              ///<summary> Shooting video, not stills | </summary>
            CAMERA_FEEDBACK_VIDEO=1,
              ///<summary> Unable to achieve requested exposure (e.g. shutter
speed too low) | </summary>
            CAMERA_FEEDBACK_BADEXPOSURE=2,
              ///<summary> Closed loop feedback from camera, we know for sure it
has successfully taken a picture | </summary>
            CAMERA_FEEDBACK_CLOSEDLOOP=3,
              ///<summary> Open loop camera, an image trigger has been requested
but we can't know for sure it has successfully taken a picture | </summary>
            CAMERA_FEEDBACK_OPENLOOP=4,
              ///<summary>  | </summary>
            ENUM_END=5,

        };

        ///<summary>  </summary>
        public enum MAV_MODE_GIMBAL
        {
       ///<summary> Gimbal is powered on but has not started initializing yet |
</summary>
            UNINITIALIZED=0,
              ///<summary> Gimbal is currently running calibration on the pitch
axis | </summary>
            CALIBRATING_PITCH=1,
              ///<summary> Gimbal is currently running calibration on the roll
axis | </summary>
            CALIBRATING_ROLL=2,
              ///<summary> Gimbal is currently running calibration on the yaw
axis | </summary>
            CALIBRATING_YAW=3,
              ///<summary> Gimbal has finished calibrating and initializing, but
is relaxed pending reception of first rate command from copter | </summary>
            INITIALIZED=4,
              ///<summary> Gimbal is actively stabilizing | </summary>
            ACTIVE=5,
              ///<summary> Gimbal is relaxed because it missed more than 10
expected rate command messages in a row.  Gimbal will move back to active mode
when it receives a new rate command | </summary>
            RATE_CMD_TIMEOUT=6,
              ///<summary>  | </summary>
            ENUM_END=7,

        };

        ///<summary>  </summary>
        public enum GIMBAL_AXIS
        {
       ///<summary> Gimbal yaw axis | </summary>
            YAW=0,
              ///<summary> Gimbal pitch axis | </summary>
            PITCH=1,
              ///<summary> Gimbal roll axis | </summary>
            ROLL=2,
```

```
            ///<summary>  | </summary>
            ENUM_END=3,

        };

        ///<summary>  </summary>
        public enum GIMBAL_AXIS_CALIBRATION_STATUS
        {
       ///<summary> Axis calibration is in progress | </summary>
            IN_PROGRESS=0,
             ///<summary> Axis calibration succeeded | </summary>
            SUCCEEDED=1,
             ///<summary> Axis calibration failed | </summary>
            FAILED=2,
             ///<summary>  | </summary>
            ENUM_END=3,

        };

        ///<summary>  </summary>
        public enum FACTORY_TEST
        {
       ///<summary> Tests to make sure each axis can move to its mechanical
limits | </summary>
            AXIS_RANGE_LIMITS=0,
             ///<summary>  | </summary>
            ENUM_END=1,

        };

        ///<summary>  </summary>
        public enum GIMBAL_AXIS_CALIBRATION_REQUIRED
        {
       ///<summary> Whether or not this axis requires calibration is unknown at
this time | </summary>
            UNKNOWN=0,
             ///<summary> This axis requires calibration | </summary>
            TRUE=1,
             ///<summary> This axis does not require calibration | </summary>
            FALSE=2,
             ///<summary>  | </summary>
            ENUM_END=3,

        };

        ///<summary>  </summary>
        public enum GOPRO_HEARTBEAT_STATUS
        {
       ///<summary> No GoPro connected | </summary>
            DISCONNECTED=0,
             ///<summary> The detected GoPro is not HeroBus compatible |
</summary>
            INCOMPATIBLE=1,
             ///<summary> A HeroBus compatible GoPro is connected | </summary>
            CONNECTED_POWER_OFF=2,
             ///<summary> A HeroBus compatible GoPro is connected | </summary>
            CONNECTED_POWER_ON=3,
             ///<summary> A HeroBus compatible GoPro is connected and recording
| </summary>
            RECORDING=4,
             ///<summary> A HeroBus compatible GoPro is connected and
overtemperature | </summary>
```

```
            ERR_OVERTEMP=5,
              ///<summary> A HeroBus compatible GoPro is connected and storage is
missing or full | </summary>
            ERR_STORAGE=6,
              ///<summary>  | </summary>
            ENUM_END=7,

        };

        ///<summary>  </summary>
        public enum GOPRO_SET_RESPONSE_RESULT
        {
       ///<summary> The write message with ID indicated failed | </summary>
            FAILURE=0,
              ///<summary> The write message with ID indicated succeeded |
</summary>
            SUCCESS=1,
              ///<summary>  | </summary>
            ENUM_END=2,

        };

        ///<summary>  </summary>
        public enum GOPRO_COMMAND
        {
       ///<summary> (Get/Set) | </summary>
            POWER=0,
              ///<summary> (Get/Set) | </summary>
            CAPTURE_MODE=1,
              ///<summary> (___/Set) | </summary>
            SHUTTER=2,
              ///<summary> (Get/___) | </summary>
            BATTERY=3,
              ///<summary> (Get/___) | </summary>
            MODEL=4,
              ///<summary> (Get/___) | </summary>
            REQUEST_FAILED=254,
              ///<summary>  | </summary>
            ENUM_END=255,

        };

        ///<summary>  </summary>
        public enum LED_CONTROL_PATTERN
        {
       ///<summary> LED patterns off (return control to regular vehicle control)
| </summary>
            OFF=0,
              ///<summary> LEDs show pattern during firmware update | </summary>
            FIRMWAREUPDATE=1,
              ///<summary> Custom Pattern using custom bytes fields | </summary>
            CUSTOM=255,
              ///<summary>  | </summary>
            ENUM_END=256,

        };

        ///<summary> Flags in EKF_STATUS message </summary>
        public enum EKF_STATUS_FLAGS
        {
       ///<summary> set if EKF's attitude estimate is good | </summary>
            EKF_ATTITUDE=1,
```

```
            ///<summary> set if EKF's horizontal velocity estimate is good |
</summary>
            EKF_VELOCITY_HORIZ=2,
            ///<summary> set if EKF's vertical velocity estimate is good |
</summary>
            EKF_VELOCITY_VERT=4,
            ///<summary> set if EKF's horizontal position (relative) estimate
is good | </summary>
            EKF_POS_HORIZ_REL=8,
            ///<summary> set if EKF's horizontal position (absolute) estimate
is good | </summary>
            EKF_POS_HORIZ_ABS=16,
            ///<summary> set if EKF's vertical position (absolute) estimate is
good | </summary>
            EKF_POS_VERT_ABS=32,
            ///<summary> set if EKF's vertical position (above ground) estimate
is good | </summary>
            EKF_POS_VERT_AGL=64,
            ///<summary> EKF is in constant position mode and does not know
it's absolute or relative position | </summary>
            EKF_CONST_POS_MODE=128,
            ///<summary> set if EKF's predicted horizontal position (relative)
estimate is good | </summary>
            EKF_PRED_POS_HORIZ_REL=256,
            ///<summary> set if EKF's predicted horizontal position (absolute)
estimate is good | </summary>
            EKF_PRED_POS_HORIZ_ABS=512,
            ///<summary>  | </summary>
            ENUM_END=513,

        };

        ///<summary>  </summary>
        public enum PID_TUNING_AXIS
        {
      ///<summary>  | </summary>
            PID_TUNING_ROLL=1,
            ///<summary>  | </summary>
            PID_TUNING_PITCH=2,
            ///<summary>  | </summary>
            PID_TUNING_YAW=3,
            ///<summary>  | </summary>
            PID_TUNING_ACCZ=4,
            ///<summary>  | </summary>
            PID_TUNING_STEER=5,
            ///<summary>  | </summary>
            ENUM_END=6,

        };

        ///<summary>  </summary>
        public enum MAG_CAL_STATUS
        {
      ///<summary>  | </summary>
            MAG_CAL_NOT_STARTED=0,
            ///<summary>  | </summary>
            MAG_CAL_WAITING_TO_START=1,
            ///<summary>  | </summary>
            MAG_CAL_RUNNING_STEP_ONE=2,
            ///<summary>  | </summary>
            MAG_CAL_RUNNING_STEP_TWO=3,
            ///<summary>  | </summary>
```

```
            MAG_CAL_SUCCESS=4,
              ///<summary>  | </summary>
            MAG_CAL_FAILED=5,
              ///<summary>  | </summary>
            ENUM_END=6,

        };

        ///<summary> Special ACK block numbers control activation of dataflash
log streaming </summary>
        public enum MAV_REMOTE_LOG_DATA_BLOCK_COMMANDS
        {
       ///<summary> UAV to stop sending DataFlash blocks | </summary>
            MAV_REMOTE_LOG_DATA_BLOCK_STOP=2147483645,
              ///<summary> UAV to start sending DataFlash blocks | </summary>
            MAV_REMOTE_LOG_DATA_BLOCK_START=2147483646,
              ///<summary>  | </summary>
            ENUM_END=2147483647,

        };

        ///<summary> Possible remote log data block statuses </summary>
        public enum MAV_REMOTE_LOG_DATA_BLOCK_STATUSES
        {
       ///<summary> This block has NOT been received | </summary>
            MAV_REMOTE_LOG_DATA_BLOCK_NACK=0,
              ///<summary> This block has been received | </summary>
            MAV_REMOTE_LOG_DATA_BLOCK_ACK=1,
              ///<summary>  | </summary>
            ENUM_END=2,

        };



        ///<summary> Micro air vehicle / autopilot classes. This identifies the
individual model. </summary>
        public enum MAV_AUTOPILOT
        {
       ///<summary> Generic autopilot, full support for everything | </summary>
            GENERIC=0,
              ///<summary> PIXHAWK autopilot, http://pixhawk.ethz.ch | </summary>
            PIXHAWK=1,
              ///<summary> SLUGS autopilot, http://slugsuav.soe.ucsc.edu |
</summary>
            SLUGS=2,
              ///<summary> ArduPilotMega / ArduCopter, http://diydrones.com |
</summary>
            ARDUPILOTMEGA=3,
              ///<summary> OpenPilot, http://openpilot.org | </summary>
            OPENPILOT=4,
              ///<summary> Generic autopilot only supporting simple waypoints |
</summary>
            GENERIC_WAYPOINTS_ONLY=5,
              ///<summary> Generic autopilot supporting waypoints and other
simple navigation commands | </summary>
            GENERIC_WAYPOINTS_AND_SIMPLE_NAVIGATION_ONLY=6,
              ///<summary> Generic autopilot supporting the full mission command
set | </summary>
            GENERIC_MISSION_FULL=7,
              ///<summary> No valid autopilot, e.g. a GCS or other MAVLink
component | </summary>
```

```
            INVALID=8,
              ///<summary> PPZ UAV - http://nongnu.org/paparazzi | </summary>
            PPZ=9,
              ///<summary> UAV Dev Board | </summary>
            UDB=10,
              ///<summary> FlexiPilot | </summary>
            FP=11,
              ///<summary> PX4 Autopilot - http://pixhawk.ethz.ch/px4/ |
</summary>
            PX4=12,
              ///<summary> SMACCMPilot - http://smaccmpilot.org | </summary>
            SMACCMPILOT=13,
              ///<summary> AutoQuad -- http://autoquad.org | </summary>
            AUTOQUAD=14,
              ///<summary> Armazila -- http://armazila.com | </summary>
            ARMAZILA=15,
              ///<summary> Aerob -- http://aerob.ru | </summary>
            AEROB=16,
              ///<summary> ASLUAV autopilot -- http://www.asl.ethz.ch |
</summary>
            ASLUAV=17,
              ///<summary>  | </summary>
            ENUM_END=18,

        };

        ///<summary>  </summary>
        public enum MAV_TYPE
        {
       ///<summary> Generic micro air vehicle. | </summary>
            GENERIC=0,
              ///<summary> Fixed wing aircraft. | </summary>
            FIXED_WING=1,
              ///<summary> Quadrotor | </summary>
            QUADROTOR=2,
              ///<summary> Coaxial helicopter | </summary>
            COAXIAL=3,
              ///<summary> Normal helicopter with tail rotor. | </summary>
            HELICOPTER=4,
              ///<summary> Ground installation | </summary>
            ANTENNA_TRACKER=5,
              ///<summary> Operator control unit / ground control station |
</summary>
            GCS=6,
              ///<summary> Airship, controlled | </summary>
            AIRSHIP=7,
              ///<summary> Free balloon, uncontrolled | </summary>
            FREE_BALLOON=8,
              ///<summary> Rocket | </summary>
            ROCKET=9,
              ///<summary> Ground rover | </summary>
            GROUND_ROVER=10,
              ///<summary> Surface vessel, boat, ship | </summary>
            SURFACE_BOAT=11,
              ///<summary> Submarine | </summary>
            SUBMARINE=12,
              ///<summary> Hexarotor | </summary>
            HEXAROTOR=13,
              ///<summary> Octorotor | </summary>
            OCTOROTOR=14,
              ///<summary> Octorotor | </summary>
            TRICOPTER=15,
```

```
            ///<summary> Flapping wing | </summary>
            FLAPPING_WING=16,
            ///<summary> Flapping wing | </summary>
            KITE=17,
            ///<summary> Onboard companion controller | </summary>
            ONBOARD_CONTROLLER=18,
            ///<summary> Two-rotor VTOL using control surfaces in vertical
operation in addition. Tailsitter. | </summary>
            VTOL_DUOROTOR=19,
            ///<summary> Quad-rotor VTOL using a V-shaped quad config in
vertical operation. Tailsitter. | </summary>
            VTOL_QUADROTOR=20,
            ///<summary> VTOL reserved 1 | </summary>
            VTOL_RESERVED1=21,
            ///<summary> VTOL reserved 2 | </summary>
            VTOL_RESERVED2=22,
            ///<summary> VTOL reserved 3 | </summary>
            VTOL_RESERVED3=23,
            ///<summary> VTOL reserved 4 | </summary>
            VTOL_RESERVED4=24,
            ///<summary> VTOL reserved 5 | </summary>
            VTOL_RESERVED5=25,
            ///<summary> Onboard gimbal | </summary>
            GIMBAL=26,
            ///<summary> Onboard ADSB peripheral | </summary>
            ADSB=27,
            ///<summary>  | </summary>
            ENUM_END=28,

        };

        ///<summary> These values define the type of firmware release.  These
values indicate the first version or release of this type.  For example the first
alpha release would be 64, the second would be 65. </summary>
        public enum FIRMWARE_VERSION_TYPE
        {
       ///<summary> development release | </summary>
            DEV=0,
            ///<summary> alpha release | </summary>
            ALPHA=64,
            ///<summary> beta release | </summary>
            BETA=128,
            ///<summary> release candidate | </summary>
            RC=192,
            ///<summary> official stable release | </summary>
            OFFICIAL=255,
            ///<summary>  | </summary>
            ENUM_END=256,

        };

        ///<summary> These flags encode the MAV mode. </summary>
        public enum MAV_MODE_FLAG
        {
       ///<summary> 0b00000001 Reserved for future use. | </summary>
            CUSTOM_MODE_ENABLED=1,
            ///<summary> 0b00000010 system has a test mode enabled. This flag
is intended for temporary system tests and should not be used for stable
implementations. | </summary>
            TEST_ENABLED=2,
```

```
            ///<summary> 0b00000100 autonomous mode enabled, system finds its
own goal positions. Guided flag can be set or not, depends on the actual
implementation. | </summary>
            AUTO_ENABLED=4,
             ///<summary> 0b00001000 guided mode enabled, system flies MISSIONs
/ mission items. | </summary>
            GUIDED_ENABLED=8,
             ///<summary> 0b00010000 system stabilizes electronically its
attitude (and optionally position). It needs however further control inputs to
move around. | </summary>
            STABILIZE_ENABLED=16,
             ///<summary> 0b00100000 hardware in the loop simulation. All motors
/ actuators are blocked, but internal software is full operational. | </summary>
            HIL_ENABLED=32,
             ///<summary> 0b01000000 remote control input is enabled. |
</summary>
            MANUAL_INPUT_ENABLED=64,
             ///<summary> 0b10000000 MAV safety set to armed. Motors are enabled
/ running / can start. Ready to fly. | </summary>
            SAFETY_ARMED=128,
             ///<summary>  | </summary>
            ENUM_END=129,

        };

        ///<summary> These values encode the bit positions of the decode
position. These values can be used to read the value of a flag bit by combining
the base_mode variable with AND with the flag position value. The result will be
either 0 or 1, depending on if the flag is set or not. </summary>
        public enum MAV_MODE_FLAG_DECODE_POSITION
        {
       ///<summary> Eighth bit: 00000001 | </summary>
            CUSTOM_MODE=1,
             ///<summary> Seventh bit: 00000010 | </summary>
            TEST=2,
             ///<summary> Sixt bit:   00000100 | </summary>
            AUTO=4,
             ///<summary> Fifth bit:  00001000 | </summary>
            GUIDED=8,
             ///<summary> Fourth bit: 00010000 | </summary>
            STABILIZE=16,
             ///<summary> Third bit:  00100000 | </summary>
            HIL=32,
             ///<summary> Second bit: 01000000 | </summary>
            MANUAL=64,
             ///<summary> First bit:  10000000 | </summary>
            SAFETY=128,
             ///<summary>  | </summary>
            ENUM_END=129,

        };

        ///<summary> Override command, pauses current mission execution and moves
immediately to a position </summary>
        public enum MAV_GOTO
        {
       ///<summary> Hold at the current position. | </summary>
            DO_HOLD=0,
             ///<summary> Continue with the next item in mission execution. |
</summary>
            DO_CONTINUE=1,
```

```
            ///<summary> Hold at the current position of the system |
</summary>
            HOLD_AT_CURRENT_POSITION=2,
             ///<summary> Hold at the position specified in the parameters of
the DO_HOLD action | </summary>
            HOLD_AT_SPECIFIED_POSITION=3,
             ///<summary>  | </summary>
            ENUM_END=4,

        };

        ///<summary> These defines are predefined OR-combined mode flags. There
is no need to use values from this enum, but it               simplifies the use
of the mode flags. Note that manual input is enabled in all modes as a safety
override. </summary>
        public enum MAV_MODE
        {
       ///<summary> System is not ready to fly, booting, calibrating, etc. No
flag is set. | </summary>
            PREFLIGHT=0,
             ///<summary> System is allowed to be active, under manual (RC)
control, no stabilization | </summary>
            MANUAL_DISARMED=64,
             ///<summary> UNDEFINED mode. This solely depends on the autopilot -
use with caution, intended for developers only. | </summary>
            TEST_DISARMED=66,
             ///<summary> System is allowed to be active, under assisted RC
control. | </summary>
            STABILIZE_DISARMED=80,
             ///<summary> System is allowed to be active, under autonomous
control, manual setpoint | </summary>
            GUIDED_DISARMED=88,
             ///<summary> System is allowed to be active, under autonomous
control and navigation (the trajectory is decided onboard and not pre-programmed
by MISSIONs) | </summary>
            AUTO_DISARMED=92,
             ///<summary> System is allowed to be active, under manual (RC)
control, no stabilization | </summary>
            MANUAL_ARMED=192,
             ///<summary> UNDEFINED mode. This solely depends on the autopilot -
use with caution, intended for developers only. | </summary>
            TEST_ARMED=194,
             ///<summary> System is allowed to be active, under assisted RC
control. | </summary>
            STABILIZE_ARMED=208,
             ///<summary> System is allowed to be active, under autonomous
control, manual setpoint | </summary>
            GUIDED_ARMED=216,
             ///<summary> System is allowed to be active, under autonomous
control and navigation (the trajectory is decided onboard and not pre-programmed
by MISSIONs) | </summary>
            AUTO_ARMED=220,
             ///<summary>  | </summary>
            ENUM_END=221,

        };

        ///<summary>  </summary>
        public enum MAV_STATE
        {
       ///<summary> Uninitialized system, state is unknown. | </summary>
            UNINIT=0,
```

```
            ///<summary> System is booting up. | </summary>
            BOOT=1,
            ///<summary> System is calibrating and not flight-ready. |
</summary>
            CALIBRATING=2,
            ///<summary> System is grounded and on standby. It can be launched
any time. | </summary>
            STANDBY=3,
            ///<summary> System is active and might be already airborne. Motors
are engaged. | </summary>
            ACTIVE=4,
            ///<summary> System is in a non-normal flight mode. It can however
still navigate. | </summary>
            CRITICAL=5,
            ///<summary> System is in a non-normal flight mode. It lost control
over parts or over the whole airframe. It is in mayday and going down. |
</summary>
            EMERGENCY=6,
            ///<summary> System just initialized its power-down sequence, will
shut down now. | </summary>
            POWEROFF=7,
            ///<summary>  | </summary>
            ENUM_END=8,

        };

        ///<summary>  </summary>
        public enum MAV_COMPONENT
        {
       ///<summary>  | </summary>
            MAV_COMP_ID_ALL=0,
            ///<summary>  | </summary>
            MAV_COMP_ID_CAMERA=100,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO1=140,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO2=141,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO3=142,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO4=143,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO5=144,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO6=145,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO7=146,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO8=147,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO9=148,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO10=149,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO11=150,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO12=151,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO13=152,
            ///<summary>  | </summary>
            MAV_COMP_ID_SERVO14=153,
            ///<summary>  | </summary>
```

```
            MAV_COMP_ID_GIMBAL=154,
              ///<summary>  | </summary>
            MAV_COMP_ID_LOG=155,
              ///<summary>  | </summary>
            MAV_COMP_ID_ADSB=156,
              ///<summary>  | </summary>
            MAV_COMP_ID_MAPPER=180,
              ///<summary>  | </summary>
            MAV_COMP_ID_MISSIONPLANNER=190,
              ///<summary>  | </summary>
            MAV_COMP_ID_PATHPLANNER=195,
              ///<summary>  | </summary>
            MAV_COMP_ID_IMU=200,
              ///<summary>  | </summary>
            MAV_COMP_ID_IMU_2=201,
              ///<summary>  | </summary>
            MAV_COMP_ID_IMU_3=202,
              ///<summary>  | </summary>
            MAV_COMP_ID_GPS=220,
              ///<summary>  | </summary>
            MAV_COMP_ID_UDP_BRIDGE=240,
              ///<summary>  | </summary>
            MAV_COMP_ID_UART_BRIDGE=241,
              ///<summary>  | </summary>
            MAV_COMP_ID_SYSTEM_CONTROL=250,
              ///<summary>  | </summary>
            ENUM_END=251,

        };

        ///<summary> These encode the sensors whose status is sent as part of the
SYS_STATUS message. </summary>
        public enum MAV_SYS_STATUS_SENSOR
        {
       ///<summary> 0x01 3D gyro | </summary>
            _3D_GYRO=1,
              ///<summary> 0x02 3D accelerometer | </summary>
            _3D_ACCEL=2,
              ///<summary> 0x04 3D magnetometer | </summary>
            _3D_MAG=4,
              ///<summary> 0x08 absolute pressure | </summary>
            ABSOLUTE_PRESSURE=8,
              ///<summary> 0x10 differential pressure | </summary>
            DIFFERENTIAL_PRESSURE=16,
              ///<summary> 0x20 GPS | </summary>
            GPS=32,
              ///<summary> 0x40 optical flow | </summary>
            OPTICAL_FLOW=64,
              ///<summary> 0x80 computer vision position | </summary>
            VISION_POSITION=128,
              ///<summary> 0x100 laser based position | </summary>
            LASER_POSITION=256,
              ///<summary> 0x200 external ground truth (Vicon or Leica) |
</summary>
            EXTERNAL_GROUND_TRUTH=512,
              ///<summary> 0x400 3D angular rate control | </summary>
            ANGULAR_RATE_CONTROL=1024,
              ///<summary> 0x800 attitude stabilization | </summary>
            ATTITUDE_STABILIZATION=2048,
              ///<summary> 0x1000 yaw position | </summary>
            YAW_POSITION=4096,
              ///<summary> 0x2000 z/altitude control | </summary>
```

```
            Z_ALTITUDE_CONTROL=8192,
              ///<summary> 0x4000 x/y position control | </summary>
            XY_POSITION_CONTROL=16384,
              ///<summary> 0x8000 motor outputs / control | </summary>
            MOTOR_OUTPUTS=32768,
              ///<summary> 0x10000 rc receiver | </summary>
            RC_RECEIVER=65536,
              ///<summary> 0x20000 2nd 3D gyro | </summary>
            _3D_GYRO2=131072,
              ///<summary> 0x40000 2nd 3D accelerometer | </summary>
            _3D_ACCEL2=262144,
              ///<summary> 0x80000 2nd 3D magnetometer | </summary>
            _3D_MAG2=524288,
              ///<summary> 0x100000 geofence | </summary>
            MAV_SYS_STATUS_GEOFENCE=1048576,
              ///<summary> 0x200000 AHRS subsystem health | </summary>
            MAV_SYS_STATUS_AHRS=2097152,
              ///<summary> 0x400000 Terrain subsystem health | </summary>
            MAV_SYS_STATUS_TERRAIN=4194304,
              ///<summary>  | </summary>
            ENUM_END=4194305,

        };

        ///<summary>  </summary>
        public enum MAV_FRAME
        {
       ///<summary> Global coordinate frame, WGS84 coordinate system. First value
/ x: latitude, second value / y: longitude, third value / z: positive altitude
over mean sea level (MSL) | </summary>
            GLOBAL=0,
              ///<summary> Local coordinate frame, Z-up (x: north, y: east, z:
down). | </summary>
            LOCAL_NED=1,
              ///<summary> NOT a coordinate frame, indicates a mission command. |
</summary>
            MISSION=2,
              ///<summary> Global coordinate frame, WGS84 coordinate system,
relative altitude over ground with respect to the home position. First value / x:
latitude, second value / y: longitude, third value / z: positive altitude with 0
being at the altitude of the home location. | </summary>
            GLOBAL_RELATIVE_ALT=3,
              ///<summary> Local coordinate frame, Z-down (x: east, y: north, z:
up) | </summary>
            LOCAL_ENU=4,
              ///<summary> Global coordinate frame, WGS84 coordinate system.
First value / x: latitude in degrees*1.0e-7, second value / y: longitude in
degrees*1.0e-7, third value / z: positive altitude over mean sea level (MSL) |
</summary>
            GLOBAL_INT=5,
              ///<summary> Global coordinate frame, WGS84 coordinate system,
relative altitude over ground with respect to the home position. First value / x:
latitude in degrees*10e-7, second value / y: longitude in degrees*10e-7, third
value / z: positive altitude with 0 being at the altitude of the home location. |
</summary>
            GLOBAL_RELATIVE_ALT_INT=6,
              ///<summary> Offset to the current local frame. Anything expressed
in this frame should be added to the current local frame position. | </summary>
            LOCAL_OFFSET_NED=7,
              ///<summary> Setpoint in body NED frame. This makes sense if all
position control is externalized - e.g. useful to command 2 m/s^2 acceleration to
the right. | </summary>
```

```
            BODY_NED=8,
              ///<summary> Offset in body NED frame. This makes sense if adding
setpoints to the current flight path, to avoid an obstacle - e.g. useful to
command 2 m/s^2 acceleration to the east. | </summary>
            BODY_OFFSET_NED=9,
              ///<summary> Global coordinate frame with above terrain level
altitude. WGS84 coordinate system, relative altitude over terrain with respect to
the waypoint coordinate. First value / x: latitude in degrees, second value / y:
longitude in degrees, third value / z: positive altitude in meters with 0 being
at ground level in terrain model. | </summary>
            GLOBAL_TERRAIN_ALT=10,
              ///<summary> Global coordinate frame with above terrain level
altitude. WGS84 coordinate system, relative altitude over terrain with respect to
the waypoint coordinate. First value / x: latitude in degrees*10e-7, second value
/ y: longitude in degrees*10e-7, third value / z: positive altitude in meters
with 0 being at ground level in terrain model. | </summary>
            GLOBAL_TERRAIN_ALT_INT=11,
              ///<summary>  | </summary>
            ENUM_END=12,

        };

        ///<summary>  </summary>
        public enum MAVLINK_DATA_STREAM_TYPE
        {
       ///<summary>  | </summary>
            MAVLINK_DATA_STREAM_IMG_JPEG=1,
              ///<summary>  | </summary>
            MAVLINK_DATA_STREAM_IMG_BMP=2,
              ///<summary>  | </summary>
            MAVLINK_DATA_STREAM_IMG_RAW8U=3,
              ///<summary>  | </summary>
            MAVLINK_DATA_STREAM_IMG_RAW32U=4,
              ///<summary>  | </summary>
            MAVLINK_DATA_STREAM_IMG_PGM=5,
              ///<summary>  | </summary>
            MAVLINK_DATA_STREAM_IMG_PNG=6,
              ///<summary>  | </summary>
            ENUM_END=7,

        };

        ///<summary>  </summary>
        public enum FENCE_ACTION
        {
       ///<summary> Disable fenced mode | </summary>
            NONE=0,
              ///<summary> Switched to guided mode to return point (fence point
0) | </summary>
            GUIDED=1,
              ///<summary> Report fence breach, but don't take action |
</summary>
            REPORT=2,
              ///<summary> Switched to guided mode to return point (fence point
0) with manual throttle control | </summary>
            GUIDED_THR_PASS=3,
              ///<summary>  | </summary>
            ENUM_END=4,

        };

        ///<summary>  </summary>
```

```
        public enum FENCE_BREACH
        {
       ///<summary> No last fence breach | </summary>
            NONE=0,
             ///<summary> Breached minimum altitude | </summary>
            MINALT=1,
             ///<summary> Breached maximum altitude | </summary>
            MAXALT=2,
             ///<summary> Breached fence boundary | </summary>
            BOUNDARY=3,
             ///<summary>  | </summary>
            ENUM_END=4,

        };

        ///<summary> Enumeration of possible mount operation modes </summary>
        public enum MAV_MOUNT_MODE
        {
       ///<summary> Load and keep safe position (Roll,Pitch,Yaw) from permant
memory and stop stabilization | </summary>
            RETRACT=0,
             ///<summary> Load and keep neutral position (Roll,Pitch,Yaw) from
permanent memory. | </summary>
            NEUTRAL=1,
             ///<summary> Load neutral position and start MAVLink Roll,Pitch,Yaw
control with stabilization | </summary>
            MAVLINK_TARGETING=2,
             ///<summary> Load neutral position and start RC Roll,Pitch,Yaw
control with stabilization | </summary>
            RC_TARGETING=3,
             ///<summary> Load neutral position and start to point to
Lat,Lon,Alt | </summary>
            GPS_POINT=4,
             ///<summary>  | </summary>
            ENUM_END=5,

        };

        ///<summary> Data stream IDs. A data stream is not a fixed set of
messages, but rather a      recommendation to the autopilot software. Individual
autopilots may or may not obey      the recommended messages. </summary>
        public enum MAV_DATA_STREAM
        {
       ///<summary> Enable all data streams | </summary>
            ALL=0,
             ///<summary> Enable IMU_RAW, GPS_RAW, GPS_STATUS packets. |
</summary>
            RAW_SENSORS=1,
             ///<summary> Enable GPS_STATUS, CONTROL_STATUS, AUX_STATUS |
</summary>
            EXTENDED_STATUS=2,
             ///<summary> Enable RC_CHANNELS_SCALED, RC_CHANNELS_RAW,
SERVO_OUTPUT_RAW | </summary>
            RC_CHANNELS=3,
             ///<summary> Enable ATTITUDE_CONTROLLER_OUTPUT,
POSITION_CONTROLLER_OUTPUT, NAV_CONTROLLER_OUTPUT. | </summary>
            RAW_CONTROLLER=4,
             ///<summary> Enable LOCAL_POSITION,
GLOBAL_POSITION/GLOBAL_POSITION_INT messages. | </summary>
            POSITION=6,
             ///<summary> Dependent on the autopilot | </summary>
            EXTRA1=10,
```

```
            ///<summary> Dependent on the autopilot | </summary>
            EXTRA2=11,
            ///<summary> Dependent on the autopilot | </summary>
            EXTRA3=12,
            ///<summary>  | </summary>
            ENUM_END=13,

        };

        ///<summary>  The ROI (region of interest) for the vehicle. This can be
be used by the vehicle for camera/vehicle attitude alignment (see
MAV_CMD_NAV_ROI). </summary>
        public enum MAV_ROI
        {
       ///<summary> No region of interest. | </summary>
            NONE=0,
            ///<summary> Point toward next MISSION. | </summary>
            WPNEXT=1,
            ///<summary> Point toward given MISSION. | </summary>
            WPINDEX=2,
            ///<summary> Point toward fixed location. | </summary>
            LOCATION=3,
            ///<summary> Point toward of given id. | </summary>
            TARGET=4,
            ///<summary>  | </summary>
            ENUM_END=5,

        };

        ///<summary> ACK / NACK / ERROR values as a result of MAV_CMDs and for
mission item transmission. </summary>
        public enum MAV_CMD_ACK
        {
       ///<summary> Command / mission item is ok. | </summary>
            OK=1,
            ///<summary> Generic error message if none of the other reasons
fails or if no detailed error reporting is implemented. | </summary>
            ERR_FAIL=2,
            ///<summary> The system is refusing to accept this command from
this source / communication partner. | </summary>
            ERR_ACCESS_DENIED=3,
            ///<summary> Command or mission item is not supported, other
commands would be accepted. | </summary>
            ERR_NOT_SUPPORTED=4,
            ///<summary> The coordinate frame of this command / mission item is
not supported. | </summary>
            ERR_COORDINATE_FRAME_NOT_SUPPORTED=5,
            ///<summary> The coordinate frame of this command is ok, but he
coordinate values exceed the safety limits of this system. This is a generic
error, please use the more specific error messages below if possible. |
</summary>
            ERR_COORDINATES_OUT_OF_RANGE=6,
            ///<summary> The X or latitude value is out of range. | </summary>
            ERR_X_LAT_OUT_OF_RANGE=7,
            ///<summary> The Y or longitude value is out of range. | </summary>
            ERR_Y_LON_OUT_OF_RANGE=8,
            ///<summary> The Z or altitude value is out of range. | </summary>
            ERR_Z_ALT_OUT_OF_RANGE=9,
            ///<summary>  | </summary>
            ENUM_END=10,

        };
```

```
        ///<summary> Specifies the datatype of a MAVLink parameter. </summary>
        public enum MAV_PARAM_TYPE
        {
       ///<summary> 8-bit unsigned integer | </summary>
            UINT8=1,
             ///<summary> 8-bit signed integer | </summary>
            INT8=2,
             ///<summary> 16-bit unsigned integer | </summary>
            UINT16=3,
             ///<summary> 16-bit signed integer | </summary>
            INT16=4,
             ///<summary> 32-bit unsigned integer | </summary>
            UINT32=5,
             ///<summary> 32-bit signed integer | </summary>
            INT32=6,
             ///<summary> 64-bit unsigned integer | </summary>
            UINT64=7,
             ///<summary> 64-bit signed integer | </summary>
            INT64=8,
             ///<summary> 32-bit floating-point | </summary>
            REAL32=9,
             ///<summary> 64-bit floating-point | </summary>
            REAL64=10,
             ///<summary>  | </summary>
            ENUM_END=11,

        };

        ///<summary> result from a mavlink command </summary>
        public enum MAV_RESULT
        {
       ///<summary> Command ACCEPTED and EXECUTED | </summary>
            ACCEPTED=0,
             ///<summary> Command TEMPORARY REJECTED/DENIED | </summary>
            TEMPORARILY_REJECTED=1,
             ///<summary> Command PERMANENTLY DENIED | </summary>
            DENIED=2,
             ///<summary> Command UNKNOWN/UNSUPPORTED | </summary>
            UNSUPPORTED=3,
             ///<summary> Command executed, but failed | </summary>
            FAILED=4,
             ///<summary>  | </summary>
            ENUM_END=5,

        };

        ///<summary> result in a mavlink mission ack </summary>
        public enum MAV_MISSION_RESULT
        {
       ///<summary> mission accepted OK | </summary>
            MAV_MISSION_ACCEPTED=0,
             ///<summary> generic error / not accepting mission commands at all
right now | </summary>
            MAV_MISSION_ERROR=1,
             ///<summary> coordinate frame is not supported | </summary>
            MAV_MISSION_UNSUPPORTED_FRAME=2,
             ///<summary> command is not supported | </summary>
            MAV_MISSION_UNSUPPORTED=3,
             ///<summary> mission item exceeds storage space | </summary>
            MAV_MISSION_NO_SPACE=4,
```

```
            ///<summary> one of the parameters has an invalid value |
</summary>
            MAV_MISSION_INVALID=5,
            ///<summary> param1 has an invalid value | </summary>
            MAV_MISSION_INVALID_PARAM1=6,
            ///<summary> param2 has an invalid value | </summary>
            MAV_MISSION_INVALID_PARAM2=7,
            ///<summary> param3 has an invalid value | </summary>
            MAV_MISSION_INVALID_PARAM3=8,
            ///<summary> param4 has an invalid value | </summary>
            MAV_MISSION_INVALID_PARAM4=9,
            ///<summary> x/param5 has an invalid value | </summary>
            MAV_MISSION_INVALID_PARAM5_X=10,
            ///<summary> y/param6 has an invalid value | </summary>
            MAV_MISSION_INVALID_PARAM6_Y=11,
            ///<summary> param7 has an invalid value | </summary>
            MAV_MISSION_INVALID_PARAM7=12,
            ///<summary> received waypoint out of sequence | </summary>
            MAV_MISSION_INVALID_SEQUENCE=13,
            ///<summary> not accepting any mission commands from this
communication partner | </summary>
            MAV_MISSION_DENIED=14,
            ///<summary>  | </summary>
            ENUM_END=15,

        };

        ///<summary> Indicates the severity level, generally used for status
messages to indicate their relative urgency. Based on RFC-5424 using expanded
definitions at: http://www.kiwisyslog.com/kb/info:-syslog-message-levels/.
</summary>
        public enum MAV_SEVERITY
        {
       ///<summary> System is unusable. This is a "panic" condition. | </summary>
            EMERGENCY=0,
            ///<summary> Action should be taken immediately. Indicates error in
non-critical systems. | </summary>
            ALERT=1,
            ///<summary> Action must be taken immediately. Indicates failure in
a primary system. | </summary>
            CRITICAL=2,
            ///<summary> Indicates an error in secondary/redundant systems. |
</summary>
            ERROR=3,
            ///<summary> Indicates about a possible future error if this is not
resolved within a given timeframe. Example would be a low battery warning. |
</summary>
            WARNING=4,
            ///<summary> An unusual event has occured, though not an error
condition. This should be investigated for the root cause. | </summary>
            NOTICE=5,
            ///<summary> Normal operational messages. Useful for logging. No
action is required for these messages. | </summary>
            INFO=6,
            ///<summary> Useful non-operational messages that can assist in
debugging. These should not occur during normal operation. | </summary>
            DEBUG=7,
            ///<summary>  | </summary>
            ENUM_END=8,

        };
```

```
        ///<summary> Power supply status flags (bitmask) </summary>
        public enum MAV_POWER_STATUS
        {
       ///<summary> main brick power supply valid | </summary>
            BRICK_VALID=1,
              ///<summary> main servo power supply valid for FMU | </summary>
            SERVO_VALID=2,
              ///<summary> USB power is connected | </summary>
            USB_CONNECTED=4,
              ///<summary> peripheral supply is in over-current state |
</summary>
            PERIPH_OVERCURRENT=8,
              ///<summary> hi-power peripheral supply is in over-current state |
</summary>
            PERIPH_HIPOWER_OVERCURRENT=16,
              ///<summary> Power status has changed since boot | </summary>
            CHANGED=32,
              ///<summary>  | </summary>
            ENUM_END=33,

        };

        ///<summary> SERIAL_CONTROL device types </summary>
        public enum SERIAL_CONTROL_DEV
        {
       ///<summary> First telemetry port | </summary>
            TELEM1=0,
              ///<summary> Second telemetry port | </summary>
            TELEM2=1,
              ///<summary> First GPS port | </summary>
            GPS1=2,
              ///<summary> Second GPS port | </summary>
            GPS2=3,
              ///<summary> system shell | </summary>
            SHELL=10,
              ///<summary>  | </summary>
            ENUM_END=11,

        };

        ///<summary> SERIAL_CONTROL flags (bitmask) </summary>
        public enum SERIAL_CONTROL_FLAG
        {
       ///<summary> Set if this is a reply | </summary>
            REPLY=1,
              ///<summary> Set if the sender wants the receiver to send a
response as another SERIAL_CONTROL message | </summary>
            RESPOND=2,
              ///<summary> Set if access to the serial port should be removed
from whatever driver is currently using it, giving exclusive access to the
SERIAL_CONTROL protocol. The port can be handed back by sending a request without
this flag set | </summary>
            EXCLUSIVE=4,
              ///<summary> Block on writes to the serial port | </summary>
            BLOCKING=8,
              ///<summary> Send multiple replies until port is drained |
</summary>
            MULTI=16,
              ///<summary>  | </summary>
            ENUM_END=17,

        };
```

```
        ///<summary> Enumeration of distance sensor types </summary>
        public enum MAV_DISTANCE_SENSOR
        {
       ///<summary> Laser altimeter, e.g. LightWare SF02/F or PulsedLight units |
</summary>
            LASER=0,
              ///<summary> Ultrasound altimeter, e.g. MaxBotix units | </summary>
            ULTRASOUND=1,
              ///<summary>  | </summary>
            ENUM_END=2,

        };

        ///<summary> Bitmask of (optional) autopilot capabilities (64 bit). If a
bit is set, the autopilot supports this capability. </summary>
        public enum MAV_PROTOCOL_CAPABILITY
        {
       ///<summary> Autopilot supports MISSION float message type. | </summary>
            MISSION_FLOAT=1,
              ///<summary> Autopilot supports the new param float message type. |
</summary>
            PARAM_FLOAT=2,
              ///<summary> Autopilot supports MISSION_INT scaled integer message
type. | </summary>
            MISSION_INT=4,
              ///<summary> Autopilot supports COMMAND_INT scaled integer message
type. | </summary>
            COMMAND_INT=8,
              ///<summary> Autopilot supports the new param union message type. |
</summary>
            PARAM_UNION=16,
              ///<summary> Autopilot supports the new param union message type. |
</summary>
            FTP=32,
              ///<summary> Autopilot supports commanding attitude offboard. |
</summary>
            SET_ATTITUDE_TARGET=64,
              ///<summary> Autopilot supports commanding position and velocity
targets in local NED frame. | </summary>
            SET_POSITION_TARGET_LOCAL_NED=128,
              ///<summary> Autopilot supports commanding position and velocity
targets in global scaled integers. | </summary>
            SET_POSITION_TARGET_GLOBAL_INT=256,
              ///<summary> Autopilot supports terrain protocol / data handling. |
</summary>
            TERRAIN=512,
              ///<summary> Autopilot supports direct actuator control. |
</summary>
            SET_ACTUATOR_TARGET=1024,
              ///<summary> Autopilot supports the flight termination command. |
</summary>
            FLIGHT_TERMINATION=2048,
              ///<summary> Autopilot supports onboard compass calibration. |
</summary>
            COMPASS_CALIBRATION=4096,
              ///<summary>  | </summary>
            ENUM_END=4097,

        };

        ///<summary> Enumeration of estimator types </summary>
```

```
        public enum MAV_ESTIMATOR_TYPE
        {
       ///<summary> This is a naive estimator without any real covariance
feedback. | </summary>
            NAIVE=1,
              ///<summary> Computer vision based estimate. Might be up to scale.
| </summary>
            VISION=2,
              ///<summary> Visual-inertial estimate. | </summary>
            VIO=3,
              ///<summary> Plain GPS estimate. | </summary>
            GPS=4,
              ///<summary> Estimator integrating GPS and inertial sensing. |
</summary>
            GPS_INS=5,
              ///<summary>  | </summary>
            ENUM_END=6,

        };

        ///<summary> Enumeration of battery types </summary>
        public enum MAV_BATTERY_TYPE
        {
       ///<summary> Not specified. | </summary>
            UNKNOWN=0,
              ///<summary> Lithium polymere battery | </summary>
            LIPO=1,
              ///<summary> Lithium ferrite battery | </summary>
            LIFE=2,
              ///<summary> Lithium-ION battery | </summary>
            LION=3,
              ///<summary> Nickel metal hydride battery | </summary>
            NIMH=4,
              ///<summary>  | </summary>
            ENUM_END=5,

        };

        ///<summary> Enumeration of battery functions </summary>
        public enum MAV_BATTERY_FUNCTION
        {
       ///<summary> Lithium polymere battery | </summary>
            UNKNOWN=0,
              ///<summary> Battery supports all flight systems | </summary>
            ALL=1,
              ///<summary> Battery for the propulsion system | </summary>
            PROPULSION=2,
              ///<summary> Avionics battery | </summary>
            AVIONICS=3,
              ///<summary> Payload battery | </summary>
            MAV_BATTERY_TYPE_PAYLOAD=4,
              ///<summary>  | </summary>
            ENUM_END=5,

        };

        ///<summary> Enumeration of the ADSB altimeter types </summary>
        public enum ADSB_ALTITUDE_TYPE
        {
       ///<summary> Altitude reported from a Baro source using QNH reference |
</summary>
            PRESSURE_QNH=0,
```

```
            ///<summary> Altitude reported from a GNSS source | </summary>
            GEOMETRIC=1,
            ///<summary>  | </summary>
            ENUM_END=2,

        };

        ///<summary> ADSB classification for the type of vehicle emitting the
transponder signal </summary>
        public enum ADSB_EMITTER_TYPE
        {
       ///<summary>  | </summary>
            NO_INFO=0,
            ///<summary>  | </summary>
            LIGHT=1,
            ///<summary>  | </summary>
            SMALL=2,
            ///<summary>  | </summary>
            LARGE=3,
            ///<summary>  | </summary>
            HIGH_VORTEX_LARGE=4,
            ///<summary>  | </summary>
            HEAVY=5,
            ///<summary>  | </summary>
            HIGHLY_MANUV=6,
            ///<summary>  | </summary>
            ROTOCRAFT=7,
            ///<summary>  | </summary>
            UNASSIGNED=8,
            ///<summary>  | </summary>
            GLIDER=9,
            ///<summary>  | </summary>
            LIGHTER_AIR=10,
            ///<summary>  | </summary>
            PARACHUTE=11,
            ///<summary>  | </summary>
            ULTRA_LIGHT=12,
            ///<summary>  | </summary>
            UNASSIGNED2=13,
            ///<summary>  | </summary>
            UAV=14,
            ///<summary>  | </summary>
            SPACE=15,
            ///<summary>  | </summary>
            UNASSGINED3=16,
            ///<summary>  | </summary>
            EMERGENCY_SURFACE=17,
            ///<summary>  | </summary>
            SERVICE_SURFACE=18,
            ///<summary>  | </summary>
            POINT_OBSTACLE=19,
            ///<summary>  | </summary>
            ENUM_END=20,

        };

        ///<summary> These flags indicate status such as data validity of each
data source. Set = data valid </summary>
        public enum ADSB_FLAGS
        {
       ///<summary>  | </summary>
            VALID_COORDS=1,
```

```
          ///<summary>  | </summary>
        VALID_ALTITUDE=2,
          ///<summary>  | </summary>
        VALID_HEADING=4,
          ///<summary>  | </summary>
        VALID_VELOCITY=8,
          ///<summary>  | </summary>
        VALID_CALLSIGN=16,
          ///<summary>  | </summary>
        VALID_SQUAWK=32,
          ///<summary>  | </summary>
        SIMULATED=64,
          ///<summary>  | </summary>
        ENUM_END=65,

    };


[StructLayout(LayoutKind.Sequential,Pack=1,Size=42)]
public struct mavlink_sensor_offsets_t
{
    /// <summary> magnetic declination (radians) </summary>
    public  Single mag_declination;
        /// <summary> raw pressure from barometer </summary>
    public  Int32 raw_press;
        /// <summary> raw temperature from barometer </summary>
    public  Int32 raw_temp;
        /// <summary> gyro X calibration </summary>
    public  Single gyro_cal_x;
        /// <summary> gyro Y calibration </summary>
    public  Single gyro_cal_y;
        /// <summary> gyro Z calibration </summary>
    public  Single gyro_cal_z;
        /// <summary> accel X calibration </summary>
    public  Single accel_cal_x;
        /// <summary> accel Y calibration </summary>
    public  Single accel_cal_y;
        /// <summary> accel Z calibration </summary>
    public  Single accel_cal_z;
        /// <summary> magnetometer X offset </summary>
    public  Int16 mag_ofs_x;
        /// <summary> magnetometer Y offset </summary>
    public  Int16 mag_ofs_y;
        /// <summary> magnetometer Z offset </summary>
    public  Int16 mag_ofs_z;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=8)]
public struct mavlink_set_mag_offsets_t
{
    /// <summary> magnetometer X offset </summary>
    public  Int16 mag_ofs_x;
        /// <summary> magnetometer Y offset </summary>
    public  Int16 mag_ofs_y;
        /// <summary> magnetometer Z offset </summary>
    public  Int16 mag_ofs_z;
        /// <summary> System ID </summary>
    public  byte target_system;
        /// <summary> Component ID </summary>
    public  byte target_component;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_meminfo_t
    {
        /// <summary> heap top </summary>
        public  UInt16 brkval;
            /// <summary> free memory </summary>
        public  UInt16 freemem;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=12)]
    public struct mavlink_ap_adc_t
    {
        /// <summary> ADC output 1 </summary>
        public  UInt16 adc1;
            /// <summary> ADC output 2 </summary>
        public  UInt16 adc2;
            /// <summary> ADC output 3 </summary>
        public  UInt16 adc3;
            /// <summary> ADC output 4 </summary>
        public  UInt16 adc4;
            /// <summary> ADC output 5 </summary>
        public  UInt16 adc5;
            /// <summary> ADC output 6 </summary>
        public  UInt16 adc6;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=15)]
    public struct mavlink_digicam_configure_t
    {
        /// <summary> Correspondent value to given extra_param </summary>
        public  Single extra_value;
            /// <summary> Divisor number //e.g. 1000 means 1/1000 (0 means
ignore) </summary>
        public  UInt16 shutter_speed;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Mode enumeration from 1 to N //P, TV, AV, M, Etc (0
means ignore) </summary>
        public  byte mode;
            /// <summary> F stop number x 10 //e.g. 28 means 2.8 (0 means ignore)
</summary>
        public  byte aperture;
            /// <summary> ISO enumeration from 1 to N //e.g. 80, 100, 200, Etc (0
means ignore) </summary>
        public  byte iso;
            /// <summary> Exposure type enumeration from 1 to N (0 means ignore)
</summary>
        public  byte exposure_type;
            /// <summary> Command Identity (incremental loop: 0 to 255)//A
command sent multiple times will be executed or pooled just once </summary>
        public  byte command_id;
```

```
            /// <summary> Main engine cut-off time before camera trigger in
seconds/10 (0 means no cut-off) </summary>
        public  byte engine_cut_off;
            /// <summary> Extra parameters enumeration (0 means ignore)
</summary>
        public  byte extra_param;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=13)]
    public struct mavlink_digicam_control_t
    {
        /// <summary> Correspondent value to given extra_param </summary>
        public  Single extra_value;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> 0: stop, 1: start or keep it up //Session control e.g.
show/hide lens </summary>
        public  byte session;
            /// <summary> 1 to N //Zoom's absolute position (0 means ignore)
</summary>
        public  byte zoom_pos;
            /// <summary> -100 to 100 //Zooming step value to offset zoom from
the current position </summary>
        public  byte zoom_step;
            /// <summary> 0: unlock focus or keep unlocked, 1: lock focus or keep
locked, 3: re-lock focus </summary>
        public  byte focus_lock;
            /// <summary> 0: ignore, 1: shot or start filming </summary>
        public  byte shot;
            /// <summary> Command Identity (incremental loop: 0 to 255)//A
command sent multiple times will be executed or pooled just once </summary>
        public  byte command_id;
            /// <summary> Extra parameters enumeration (0 means ignore)
</summary>
        public  byte extra_param;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_mount_configure_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> mount operating mode (see MAV_MOUNT_MODE enum)
</summary>
        public  byte mount_mode;
            /// <summary> (1 = yes, 0 = no) </summary>
        public  byte stab_roll;
            /// <summary> (1 = yes, 0 = no) </summary>
        public  byte stab_pitch;
            /// <summary> (1 = yes, 0 = no) </summary>
        public  byte stab_yaw;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=15)]
    public struct mavlink_mount_control_t
    {
        /// <summary> pitch(deg*100) or lat, depending on mount mode </summary>
        public  Int32 input_a;
            /// <summary> roll(deg*100) or lon depending on mount mode </summary>
        public  Int32 input_b;
            /// <summary> yaw(deg*100) or alt (in cm) depending on mount mode
</summary>
        public  Int32 input_c;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> if "1" it will save current trimmed position on EEPROM
(just valid for NEUTRAL and LANDING) </summary>
        public  byte save_position;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_mount_status_t
    {
        /// <summary> pitch(deg*100) </summary>
        public  Int32 pointing_a;
            /// <summary> roll(deg*100) </summary>
        public  Int32 pointing_b;
            /// <summary> yaw(deg*100) </summary>
        public  Int32 pointing_c;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=12)]
    public struct mavlink_fence_point_t
    {
        /// <summary> Latitude of point </summary>
        public  Single lat;
            /// <summary> Longitude of point </summary>
        public  Single lng;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> point index (first point is 1, 0 is for return point)
</summary>
        public  byte idx;
            /// <summary> total number of points (for sanity checking) </summary>
        public  byte count;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_fence_fetch_point_t
    {
```

```
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> point index (first point is 1, 0 is for return point)
</summary>
        public  byte idx;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=8)]
    public struct mavlink_fence_status_t
    {
        /// <summary> time of last breach in milliseconds since boot </summary>
        public  UInt32 breach_time;
            /// <summary> number of fence breaches </summary>
        public  UInt16 breach_count;
            /// <summary> 0 if currently inside fence, 1 if outside </summary>
        public  byte breach_status;
            /// <summary> last breach type (see FENCE_BREACH_* enum) </summary>
        public  byte breach_type;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=28)]
    public struct mavlink_ahrs_t
    {
        /// <summary> X gyro drift estimate rad/s </summary>
        public  Single omegaIx;
            /// <summary> Y gyro drift estimate rad/s </summary>
        public  Single omegaIy;
            /// <summary> Z gyro drift estimate rad/s </summary>
        public  Single omegaIz;
            /// <summary> average accel_weight </summary>
        public  Single accel_weight;
            /// <summary> average renormalisation value </summary>
        public  Single renorm_val;
            /// <summary> average error_roll_pitch value </summary>
        public  Single error_rp;
            /// <summary> average error_yaw value </summary>
        public  Single error_yaw;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=44)]
    public struct mavlink_simstate_t
    {
        /// <summary> Roll angle (rad) </summary>
        public  Single roll;
            /// <summary> Pitch angle (rad) </summary>
        public  Single pitch;
            /// <summary> Yaw angle (rad) </summary>
        public  Single yaw;
            /// <summary> X acceleration m/s/s </summary>
        public  Single xacc;
            /// <summary> Y acceleration m/s/s </summary>
        public  Single yacc;
            /// <summary> Z acceleration m/s/s </summary>
        public  Single zacc;
```

```
            /// <summary> Angular speed around X axis rad/s </summary>
        public  Single xgyro;
            /// <summary> Angular speed around Y axis rad/s </summary>
        public  Single ygyro;
            /// <summary> Angular speed around Z axis rad/s </summary>
        public  Single zgyro;
            /// <summary> Latitude in degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude in degrees * 1E7 </summary>
        public  Int32 lng;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_hwstatus_t
    {
        /// <summary> board voltage (mV) </summary>
        public  UInt16 Vcc;
            /// <summary> I2C error count </summary>
        public  byte I2Cerr;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=9)]
    public struct mavlink_radio_t
    {
        /// <summary> receive errors </summary>
        public  UInt16 rxerrors;
            /// <summary> count of error corrected packets </summary>
        public  UInt16 @fixed;
            /// <summary> local signal strength </summary>
        public  byte rssi;
            /// <summary> remote signal strength </summary>
        public  byte remrssi;
            /// <summary> how full the tx buffer is as a percentage </summary>
        public  byte txbuf;
            /// <summary> background noise level </summary>
        public  byte noise;
            /// <summary> remote background noise level </summary>
        public  byte remnoise;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_limits_status_t
    {
        /// <summary> time of last breach in milliseconds since boot </summary>
        public  UInt32 last_trigger;
            /// <summary> time of last recovery action in milliseconds since boot
</summary>
        public  UInt32 last_action;
            /// <summary> time of last successful recovery in milliseconds since
boot </summary>
        public  UInt32 last_recovery;
            /// <summary> time of last all-clear in milliseconds since boot
</summary>
        public  UInt32 last_clear;
            /// <summary> number of fence breaches </summary>
        public  UInt16 breach_count;
```

```
            /// <summary> state of AP_Limits, (see enum LimitState, LIMITS_STATE)
</summary>
        public  byte limits_state;
            /// <summary> AP_Limit_Module bitfield of enabled modules, (see enum
moduleid or LIMIT_MODULE) </summary>
        public  byte mods_enabled;
            /// <summary> AP_Limit_Module bitfield of required modules, (see enum
moduleid or LIMIT_MODULE) </summary>
        public  byte mods_required;
            /// <summary> AP_Limit_Module bitfield of triggered modules, (see
enum moduleid or LIMIT_MODULE) </summary>
        public  byte mods_triggered;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=12)]
    public struct mavlink_wind_t
    {
        /// <summary> wind direction that wind is coming from (degrees)
</summary>
        public  Single direction;
            /// <summary> wind speed in ground plane (m/s) </summary>
        public  Single speed;
            /// <summary> vertical wind speed (m/s) </summary>
        public  Single speed_z;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=18)]
    public struct mavlink_data16_t
    {
        /// <summary> data type </summary>
        public  byte type;
            /// <summary> data length </summary>
        public  byte len;
            /// <summary> raw data </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=16)]
             public byte[] data;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=34)]
    public struct mavlink_data32_t
    {
        /// <summary> data type </summary>
        public  byte type;
            /// <summary> data length </summary>
        public  byte len;
            /// <summary> raw data </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=32)]
             public byte[] data;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=66)]
    public struct mavlink_data64_t
    {
        /// <summary> data type </summary>
```

```
    public  byte type;
        /// <summary> data length </summary>
    public  byte len;
        /// <summary> raw data </summary>
    [MarshalAs(UnmanagedType.ByValArray,SizeConst=64)]
          public byte[] data;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=98)]
public struct mavlink_data96_t
{
    /// <summary> data type </summary>
    public  byte type;
        /// <summary> data length </summary>
    public  byte len;
        /// <summary> raw data </summary>
    [MarshalAs(UnmanagedType.ByValArray,SizeConst=96)]
          public byte[] data;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=8)]
public struct mavlink_rangefinder_t
{
    /// <summary> distance in meters </summary>
    public  Single distance;
        /// <summary> raw voltage if available, zero otherwise </summary>
    public  Single voltage;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=48)]
public struct mavlink_airspeed_autocal_t
{
    /// <summary> GPS velocity north m/s </summary>
    public  Single vx;
        /// <summary> GPS velocity east m/s </summary>
    public  Single vy;
        /// <summary> GPS velocity down m/s </summary>
    public  Single vz;
        /// <summary> Differential pressure pascals </summary>
    public  Single diff_pressure;
        /// <summary> Estimated to true airspeed ratio </summary>
    public  Single EAS2TAS;
        /// <summary> Airspeed ratio </summary>
    public  Single ratio;
        /// <summary> EKF state x </summary>
    public  Single state_x;
        /// <summary> EKF state y </summary>
    public  Single state_y;
        /// <summary> EKF state z </summary>
    public  Single state_z;
        /// <summary> EKF Pax </summary>
    public  Single Pax;
        /// <summary> EKF Pby </summary>
    public  Single Pby;
        /// <summary> EKF Pcz </summary>
    public  Single Pcz;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=19)]
    public struct mavlink_rally_point_t
    {
        /// <summary> Latitude of point in degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude of point in degrees * 1E7 </summary>
        public  Int32 lng;
            /// <summary> Transit / loiter altitude in meters relative to home
</summary>
        public  Int16 alt;
            /// <summary> Break altitude in meters relative to home </summary>
        public  Int16 break_alt;
            /// <summary> Heading to aim for when landing. In centi-degrees.
</summary>
        public  UInt16 land_dir;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> point index (first point is 0) </summary>
        public  byte idx;
            /// <summary> total number of points (for sanity checking) </summary>
        public  byte count;
            /// <summary> See RALLY_FLAGS enum for definition of the bitmask.
</summary>
        public  byte flags;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_rally_fetch_point_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> point index (first point is 0) </summary>
        public  byte idx;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=20)]
    public struct mavlink_compassmot_status_t
    {
        /// <summary> current (amps) </summary>
        public  Single current;
            /// <summary> Motor Compensation X </summary>
        public  Single CompensationX;
            /// <summary> Motor Compensation Y </summary>
        public  Single CompensationY;
            /// <summary> Motor Compensation Z </summary>
        public  Single CompensationZ;
            /// <summary> throttle (percent*10) </summary>
        public  UInt16 throttle;
            /// <summary> interference (percent) </summary>
        public  UInt16 interference;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=24)]
    public struct mavlink_ahrs2_t
    {
        /// <summary> Roll angle (rad) </summary>
        public  Single roll;
            /// <summary> Pitch angle (rad) </summary>
        public  Single pitch;
            /// <summary> Yaw angle (rad) </summary>
        public  Single yaw;
            /// <summary> Altitude (MSL) </summary>
        public  Single altitude;
            /// <summary> Latitude in degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude in degrees * 1E7 </summary>
        public  Int32 lng;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=29)]
    public struct mavlink_camera_status_t
    {
        /// <summary> Image timestamp (microseconds since UNIX epoch, according
to camera clock) </summary>
        public  UInt64 time_usec;
            /// <summary> Parameter 1 (meaning depends on event, see
CAMERA_STATUS_TYPES enum) </summary>
        public  Single p1;
            /// <summary> Parameter 2 (meaning depends on event, see
CAMERA_STATUS_TYPES enum) </summary>
        public  Single p2;
            /// <summary> Parameter 3 (meaning depends on event, see
CAMERA_STATUS_TYPES enum) </summary>
        public  Single p3;
            /// <summary> Parameter 4 (meaning depends on event, see
CAMERA_STATUS_TYPES enum) </summary>
        public  Single p4;
            /// <summary> Image index </summary>
        public  UInt16 img_idx;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Camera ID </summary>
        public  byte cam_idx;
            /// <summary> See CAMERA_STATUS_TYPES enum for definition of the
bitmask </summary>
        public  byte event_id;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=45)]
    public struct mavlink_camera_feedback_t
    {
        /// <summary> Image timestamp (microseconds since UNIX epoch), as passed
in by CAMERA_STATUS message (or autopilot if no CCB) </summary>
        public  UInt64 time_usec;
            /// <summary> Latitude in (deg * 1E7) </summary>
        public  Int32 lat;
```

```
            /// <summary> Longitude in (deg * 1E7) </summary>
        public  Int32 lng;
            /// <summary> Altitude Absolute (meters AMSL) </summary>
        public  Single alt_msl;
            /// <summary> Altitude Relative (meters above HOME location)
</summary>
        public  Single alt_rel;
            /// <summary> Camera Roll angle (earth frame, degrees, +-180)
</summary>
        public  Single roll;
            /// <summary> Camera Pitch angle (earth frame, degrees, +-180)
</summary>
        public  Single pitch;
            /// <summary> Camera Yaw (earth frame, degrees, 0-360, true)
</summary>
        public  Single yaw;
            /// <summary> Focal Length (mm) </summary>
        public  Single foc_len;
            /// <summary> Image index </summary>
        public  UInt16 img_idx;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Camera ID </summary>
        public  byte cam_idx;
            /// <summary> See CAMERA_FEEDBACK_FLAGS enum for definition of the
bitmask </summary>
        public  byte flags;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_battery2_t
    {
        /// <summary> voltage in millivolts </summary>
        public  UInt16 voltage;
            /// <summary> Battery current, in 10*milliamperes (1 = 10
milliampere), -1: autopilot does not measure the current </summary>
        public  Int16 current_battery;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=40)]
    public struct mavlink_ahrs3_t
    {
        /// <summary> Roll angle (rad) </summary>
        public  Single roll;
            /// <summary> Pitch angle (rad) </summary>
        public  Single pitch;
            /// <summary> Yaw angle (rad) </summary>
        public  Single yaw;
            /// <summary> Altitude (MSL) </summary>
        public  Single altitude;
            /// <summary> Latitude in degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude in degrees * 1E7 </summary>
        public  Int32 lng;
            /// <summary> test variable1 </summary>
        public  Single v1;
            /// <summary> test variable2 </summary>
        public  Single v2;
```

```
            /// <summary> test variable3 </summary>
        public  Single v3;
            /// <summary> test variable4 </summary>
        public  Single v4;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_autopilot_version_request_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=206)]
    public struct mavlink_remote_log_data_block_t
    {
        /// <summary> log data block sequence number </summary>
        public  UInt32 seqno;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> log data block </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=200)]
              public byte[] data;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=7)]
    public struct mavlink_remote_log_block_status_t
    {
        /// <summary> log data block sequence number </summary>
        public  UInt32 seqno;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> log data block status </summary>
        public  byte status;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=29)]
    public struct mavlink_led_control_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Instance (LED instance to control or 255 for all LEDs)
</summary>
        public  byte instance;
            /// <summary> Pattern (see LED_PATTERN_ENUM) </summary>
        public  byte pattern;
```

```
            /// <summary> Custom Byte Length </summary>
        public  byte custom_len;
            /// <summary> Custom Bytes </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=24)]
              public byte[] custom_bytes;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=27)]
    public struct mavlink_mag_cal_progress_t
    {
        /// <summary> Body frame direction vector for display </summary>
        public  Single direction_x;
            /// <summary> Body frame direction vector for display </summary>
        public  Single direction_y;
            /// <summary> Body frame direction vector for display </summary>
        public  Single direction_z;
            /// <summary> Compass being calibrated </summary>
        public  byte compass_id;
            /// <summary> Bitmask of compasses being calibrated </summary>
        public  byte cal_mask;
            /// <summary> Status (see MAG_CAL_STATUS enum) </summary>
        public  byte cal_status;
            /// <summary> Attempt number </summary>
        public  byte attempt;
            /// <summary> Completion percentage </summary>
        public  byte completion_pct;
            /// <summary> Bitmask of sphere sections (see
http://en.wikipedia.org/wiki/Geodesic_grid) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=10)]
              public byte[] completion_mask;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=44)]
    public struct mavlink_mag_cal_report_t
    {
        /// <summary> RMS milligauss residuals </summary>
        public  Single fitness;
            /// <summary> X offset </summary>
        public  Single ofs_x;
            /// <summary> Y offset </summary>
        public  Single ofs_y;
            /// <summary> Z offset </summary>
        public  Single ofs_z;
            /// <summary> X diagonal (matrix 11) </summary>
        public  Single diag_x;
            /// <summary> Y diagonal (matrix 22) </summary>
        public  Single diag_y;
            /// <summary> Z diagonal (matrix 33) </summary>
        public  Single diag_z;
            /// <summary> X off-diagonal (matrix 12 and 21) </summary>
        public  Single offdiag_x;
            /// <summary> Y off-diagonal (matrix 13 and 31) </summary>
        public  Single offdiag_y;
            /// <summary> Z off-diagonal (matrix 32 and 23) </summary>
        public  Single offdiag_z;
            /// <summary> Compass being calibrated </summary>
        public  byte compass_id;
            /// <summary> Bitmask of compasses being calibrated </summary>
```

```
        public  byte cal_mask;
            /// <summary> Status (see MAG_CAL_STATUS enum) </summary>
        public  byte cal_status;
            /// <summary> 0=requires a MAV_CMD_DO_ACCEPT_MAG_CAL, 1=saved to
parameters </summary>
        public  byte autosaved;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_ekf_status_report_t
    {
        /// <summary> Velocity variance </summary>
        public  Single velocity_variance;
            /// <summary> Horizontal Position variance </summary>
        public  Single pos_horiz_variance;
            /// <summary> Vertical Position variance </summary>
        public  Single pos_vert_variance;
            /// <summary> Compass variance </summary>
        public  Single compass_variance;
            /// <summary> Terrain Altitude variance </summary>
        public  Single terrain_alt_variance;
            /// <summary> Flags </summary>
        public  UInt16 flags;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=25)]
    public struct mavlink_pid_tuning_t
    {
        /// <summary> desired rate (degrees/s) </summary>
        public  Single desired;
            /// <summary> achieved rate (degrees/s) </summary>
        public  Single achieved;
            /// <summary> FF component </summary>
        public  Single FF;
            /// <summary> P component </summary>
        public  Single P;
            /// <summary> I component </summary>
        public  Single I;
            /// <summary> D component </summary>
        public  Single D;
            /// <summary> axis </summary>
        public  byte axis;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=42)]
    public struct mavlink_gimbal_report_t
    {
        /// <summary> Time since last update (seconds) </summary>
        public  Single delta_time;
            /// <summary> Delta angle X (radians) </summary>
        public  Single delta_angle_x;
            /// <summary> Delta angle Y (radians) </summary>
        public  Single delta_angle_y;
            /// <summary> Delta angle X (radians) </summary>
        public  Single delta_angle_z;
            /// <summary> Delta velocity X (m/s) </summary>
```

```
        public  Single delta_velocity_x;
            /// <summary> Delta velocity Y (m/s) </summary>
        public  Single delta_velocity_y;
            /// <summary> Delta velocity Z (m/s) </summary>
        public  Single delta_velocity_z;
            /// <summary>  Joint ROLL (radians) </summary>
        public  Single joint_roll;
            /// <summary>  Joint EL (radians) </summary>
        public  Single joint_el;
            /// <summary>  Joint AZ (radians) </summary>
        public  Single joint_az;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_gimbal_control_t
    {
        /// <summary> Demanded angular rate X (rad/s) </summary>
        public  Single demanded_rate_x;
            /// <summary> Demanded angular rate Y (rad/s) </summary>
        public  Single demanded_rate_y;
            /// <summary> Demanded angular rate Z (rad/s) </summary>
        public  Single demanded_rate_z;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_gimbal_reset_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_gimbal_axis_calibration_progress_t
    {
        /// <summary> Which gimbal axis we're reporting calibration progress for
</summary>
        public  byte calibration_axis;
            /// <summary> The current calibration progress for this axis,
0x64=100% </summary>
        public  byte calibration_progress;
            /// <summary> The status of the running calibration </summary>
        public  byte calibration_status;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_gimbal_set_home_offsets_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=1)]
    public struct mavlink_gimbal_home_offset_calibration_result_t
    {
        /// <summary> The result of the home offset calibration </summary>
        public  byte calibration_result;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=33)]
    public struct mavlink_gimbal_set_factory_parameters_t
    {
        /// <summary> Magic number 1 for validation </summary>
        public  UInt32 magic_1;
            /// <summary> Magic number 2 for validation </summary>
        public  UInt32 magic_2;
            /// <summary> Magic number 3 for validation </summary>
        public  UInt32 magic_3;
            /// <summary> Unit Serial Number Part 1 (part code, design,
language/country) </summary>
        public  UInt32 serial_number_pt_1;
            /// <summary> Unit Serial Number Part 2 (option, year, month)
</summary>
        public  UInt32 serial_number_pt_2;
            /// <summary> Unit Serial Number Part 3 (incrementing serial number
per month) </summary>
        public  UInt32 serial_number_pt_3;
            /// <summary> Assembly Date Year </summary>
        public  UInt16 assembly_year;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Assembly Date Month </summary>
        public  byte assembly_month;
            /// <summary> Assembly Date Day </summary>
        public  byte assembly_day;
            /// <summary> Assembly Time Hour </summary>
        public  byte assembly_hour;
            /// <summary> Assembly Time Minute </summary>
        public  byte assembly_minute;
            /// <summary> Assembly Time Second </summary>
        public  byte assembly_second;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=1)]
    public struct mavlink_gimbal_factory_parameters_loaded_t
    {
```

```
        /// <summary> Dummy field because mavgen doesn't allow messages with no
fields </summary>
        public  byte dummy;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_gimbal_erase_firmware_and_config_t
    {
        /// <summary> Knock value to confirm this is a valid request </summary>
        public  UInt32 knock;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_gimbal_perform_factory_tests_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_gimbal_report_factory_tests_progress_t
    {
        /// <summary> Which factory test is currently running </summary>
        public  byte test;
            /// <summary> Which section of the test is currently running.  The
meaning of this is test-dependent </summary>
        public  byte test_section;
            /// <summary> The progress of the current test section, 0x64=100%
</summary>
        public  byte test_section_progress;
            /// <summary> The status of the currently executing test section.
The meaning of this is test and section-dependent </summary>
        public  byte test_status;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_gimbal_request_axis_calibration_status_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_gimbal_report_axis_calibration_status_t
```

```
    {
        /// <summary> Whether or not the yaw axis requires calibration, see
GIMBAL_AXIS_CALIBRATION_REQUIRED enumeration </summary>
        public  byte yaw_requires_calibration;
            /// <summary> Whether or not the pitch axis requires calibration, see
GIMBAL_AXIS_CALIBRATION_REQUIRED enumeration </summary>
        public  byte pitch_requires_calibration;
            /// <summary> Whether or not the roll axis requires calibration, see
GIMBAL_AXIS_CALIBRATION_REQUIRED enumeration </summary>
        public  byte roll_requires_calibration;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_gimbal_request_axis_calibration_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=1)]
    public struct mavlink_gopro_heartbeat_t
    {
        /// <summary> Status </summary>
        public  byte status;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_gopro_get_request_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Command ID </summary>
        public  byte cmd_id;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_gopro_get_response_t
    {
        /// <summary> Command ID </summary>
        public  byte cmd_id;
            /// <summary> Value </summary>
        public  byte value;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_gopro_set_request_t
    {
        /// <summary> System ID </summary>
```

```
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Command ID </summary>
        public  byte cmd_id;
            /// <summary> Value </summary>
        public  byte value;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_gopro_set_response_t
    {
        /// <summary> Command ID </summary>
        public  byte cmd_id;
            /// <summary> Result </summary>
        public  byte result;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=8)]
    public struct mavlink_rpm_t
    {
        /// <summary> RPM Sensor1 </summary>
        public  Single rpm1;
            /// <summary> RPM Sensor2 </summary>
        public  Single rpm2;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=9)]
    public struct mavlink_heartbeat_t
    {
        /// <summary> A bitfield for use for autopilot-specific flags. </summary>
        public  UInt32 custom_mode;
            /// <summary> Type of the MAV (quadrotor, helicopter, etc., up to 15
types, defined in MAV_TYPE ENUM) </summary>
        public  byte type;
            /// <summary> Autopilot type / class. defined in MAV_AUTOPILOT ENUM
</summary>
        public  byte autopilot;
            /// <summary> System mode bitfield, see MAV_MODE_FLAG ENUM in
mavlink/include/mavlink_types.h </summary>
        public  byte base_mode;
            /// <summary> System status flag, see MAV_STATE ENUM </summary>
        public  byte system_status;
            /// <summary> MAVLink version, not writable by user, gets added by
protocol because of magic data type: uint8_t_mavlink_version </summary>
        public  byte mavlink_version;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=31)]
    public struct mavlink_sys_status_t
    {
        /// <summary> Bitmask showing which onboard controllers and sensors are
present. Value of 0: not present. Value of 1: present. Indices defined by ENUM
MAV_SYS_STATUS_SENSOR </summary>
```

```
        public  UInt32 onboard_control_sensors_present;
            /// <summary> Bitmask showing which onboard controllers and sensors
are enabled:  Value of 0: not enabled. Value of 1: enabled. Indices defined by
ENUM MAV_SYS_STATUS_SENSOR </summary>
        public  UInt32 onboard_control_sensors_enabled;
            /// <summary> Bitmask showing which onboard controllers and sensors
are operational or have an error:  Value of 0: not enabled. Value of 1: enabled.
Indices defined by ENUM MAV_SYS_STATUS_SENSOR </summary>
        public  UInt32 onboard_control_sensors_health;
            /// <summary> Maximum usage in percent of the mainloop time, (0%: 0,
100%: 1000) should be always below 1000 </summary>
        public  UInt16 load;
            /// <summary> Battery voltage, in millivolts (1 = 1 millivolt)
</summary>
        public  UInt16 voltage_battery;
            /// <summary> Battery current, in 10*milliamperes (1 = 10
milliampere), -1: autopilot does not measure the current </summary>
        public  Int16 current_battery;
            /// <summary> Communication drops in percent, (0%: 0, 100%: 10'000),
(UART, I2C, SPI, CAN), dropped packets on all links (packets that were corrupted
on reception on the MAV) </summary>
        public  UInt16 drop_rate_comm;
            /// <summary> Communication errors (UART, I2C, SPI, CAN), dropped
packets on all links (packets that were corrupted on reception on the MAV)
</summary>
        public  UInt16 errors_comm;
            /// <summary> Autopilot-specific errors </summary>
        public  UInt16 errors_count1;
            /// <summary> Autopilot-specific errors </summary>
        public  UInt16 errors_count2;
            /// <summary> Autopilot-specific errors </summary>
        public  UInt16 errors_count3;
            /// <summary> Autopilot-specific errors </summary>
        public  UInt16 errors_count4;
            /// <summary> Remaining battery energy: (0%: 0, 100%: 100), -1:
autopilot estimate the remaining battery </summary>
        public  byte battery_remaining;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=12)]
    public struct mavlink_system_time_t
    {
        /// <summary> Timestamp of the master clock in microseconds since UNIX
epoch. </summary>
        public  UInt64 time_unix_usec;
            /// <summary> Timestamp of the component clock since boot time in
milliseconds. </summary>
        public  UInt32 time_boot_ms;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_ping_t
    {
        /// <summary> Unix timestamp in microseconds or since system boot if
smaller than MAVLink epoch (1.1.2009) </summary>
        public  UInt64 time_usec;
            /// <summary> PING sequence </summary>
        public  UInt32 seq;
```

```
            /// <summary> 0: request ping from all receiving systems, if greater
than 0: message is a ping response and number is the system id of the requesting
system </summary>
        public  byte target_system;
            /// <summary> 0: request ping from all receiving components, if
greater than 0: message is a ping response and number is the system id of the
requesting system </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=28)]
    public struct mavlink_change_operator_control_t
    {
        /// <summary> System the GCS requests control for </summary>
        public  byte target_system;
            /// <summary> 0: request control of this MAV, 1: Release control of
this MAV </summary>
        public  byte control_request;
            /// <summary> 0: key as plaintext, 1-255: future, different
hashing/encryption variants. The GCS should in general use the safest mode
possible initially and then gradually move down the encryption level if it gets a
NACK message indicating an encryption mismatch. </summary>
        public  byte version;
            /// <summary> Password / Key, depending on version plaintext or
encrypted. 25 or less characters, NULL terminated. The characters may involve A-
Z, a-z, 0-9, and "!?,.-" </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=25)]
              public byte[] passkey;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_change_operator_control_ack_t
    {
        /// <summary> ID of the GCS this message  </summary>
        public  byte gcs_system_id;
            /// <summary> 0: request control of this MAV, 1: Release control of
this MAV </summary>
        public  byte control_request;
            /// <summary> 0: ACK, 1: NACK: Wrong passkey, 2: NACK: Unsupported
passkey encryption method, 3: NACK: Already under control </summary>
        public  byte ack;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=32)]
    public struct mavlink_auth_key_t
    {
        /// <summary> key </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=32)]
              public byte[] key;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_set_mode_t
    {
```

```
        /// <summary> The new autopilot-specific mode. This field can be ignored
by an autopilot. </summary>
        public  UInt32 custom_mode;
            /// <summary> The system setting the mode </summary>
        public  byte target_system;
            /// <summary> The new base mode </summary>
        public  byte base_mode;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=20)]
    public struct mavlink_param_request_read_t
    {
        /// <summary> Parameter index. Send -1 to use the param ID field as
identifier (else the param id will be ignored) </summary>
        public  Int16 param_index;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Onboard parameter id, terminated by NULL if the length
is less than 16 human-readable chars and WITHOUT null termination (NULL) byte if
the length is exactly 16 chars - applications have to provide 16+1 bytes storage
if the ID is stored as string </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=16)]
             public byte[] param_id;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_param_request_list_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=25)]
    public struct mavlink_param_value_t
    {
        /// <summary> Onboard parameter value </summary>
        public  Single param_value;
            /// <summary> Total number of onboard parameters </summary>
        public  UInt16 param_count;
            /// <summary> Index of this onboard parameter </summary>
        public  UInt16 param_index;
            /// <summary> Onboard parameter id, terminated by NULL if the length
is less than 16 human-readable chars and WITHOUT null termination (NULL) byte if
the length is exactly 16 chars - applications have to provide 16+1 bytes storage
if the ID is stored as string </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=16)]
             public byte[] param_id;
            /// <summary> Onboard parameter type: see the MAV_PARAM_TYPE enum for
supported data types. </summary>
        public  byte param_type;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=23)]
    public struct mavlink_param_set_t
    {
        /// <summary> Onboard parameter value </summary>
        public  Single param_value;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Onboard parameter id, terminated by NULL if the length
is less than 16 human-readable chars and WITHOUT null termination (NULL) byte if
the length is exactly 16 chars - applications have to provide 16+1 bytes storage
if the ID is stored as string </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=16)]
             public byte[] param_id;
            /// <summary> Onboard parameter type: see the MAV_PARAM_TYPE enum for
supported data types. </summary>
        public  byte param_type;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=30)]
    public struct mavlink_gps_raw_int_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Latitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 lon;
            /// <summary> Altitude (AMSL, NOT WGS84), in meters * 1000 (positive
for up). Note that virtually all GPS modules provide the AMSL altitude in
addition to the WGS84 altitude. </summary>
        public  Int32 alt;
            /// <summary> GPS HDOP horizontal dilution of position in cm (m*100).
If unknown, set to: UINT16_MAX </summary>
        public  UInt16 eph;
            /// <summary> GPS VDOP vertical dilution of position in cm (m*100).
If unknown, set to: UINT16_MAX </summary>
        public  UInt16 epv;
            /// <summary> GPS ground speed (m/s * 100). If unknown, set to:
UINT16_MAX </summary>
        public  UInt16 vel;
            /// <summary> Course over ground (NOT heading, but direction of
movement) in degrees * 100, 0.0..359.99 degrees. If unknown, set to: UINT16_MAX
</summary>
        public  UInt16 cog;
            /// <summary> 0-1: no fix, 2: 2D fix, 3: 3D fix, 4: DGPS, 5: RTK.
Some applications will not use the value of this field unless it is at least two,
so always correctly fill in the fix. </summary>
        public  byte fix_type;
            /// <summary> Number of satellites visible. If unknown, set to 255
</summary>
        public  byte satellites_visible;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=101)]
    public struct mavlink_gps_status_t
    {
        /// <summary> Number of satellites visible </summary>
        public  byte satellites_visible;
            /// <summary> Global satellite ID </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=20)]
             public byte[] satellite_prn;
            /// <summary> 0: Satellite not used, 1: used for localization
</summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=20)]
             public byte[] satellite_used;
            /// <summary> Elevation (0: right on top of receiver, 90: on the
horizon) of satellite </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=20)]
             public byte[] satellite_elevation;
            /// <summary> Direction of satellite, 0: 0 deg, 255: 360 deg.
</summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=20)]
             public byte[] satellite_azimuth;
            /// <summary> Signal to noise ratio of satellite </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=20)]
             public byte[] satellite_snr;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_scaled_imu_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X acceleration (mg) </summary>
        public  Int16 xacc;
            /// <summary> Y acceleration (mg) </summary>
        public  Int16 yacc;
            /// <summary> Z acceleration (mg) </summary>
        public  Int16 zacc;
            /// <summary> Angular speed around X axis (millirad /sec) </summary>
        public  Int16 xgyro;
            /// <summary> Angular speed around Y axis (millirad /sec) </summary>
        public  Int16 ygyro;
            /// <summary> Angular speed around Z axis (millirad /sec) </summary>
        public  Int16 zgyro;
            /// <summary> X Magnetic field (milli tesla) </summary>
        public  Int16 xmag;
            /// <summary> Y Magnetic field (milli tesla) </summary>
        public  Int16 ymag;
            /// <summary> Z Magnetic field (milli tesla) </summary>
        public  Int16 zmag;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=26)]
    public struct mavlink_raw_imu_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> X acceleration (raw) </summary>
        public  Int16 xacc;
```

```
            /// <summary> Y acceleration (raw) </summary>
        public  Int16 yacc;
            /// <summary> Z acceleration (raw) </summary>
        public  Int16 zacc;
            /// <summary> Angular speed around X axis (raw) </summary>
        public  Int16 xgyro;
            /// <summary> Angular speed around Y axis (raw) </summary>
        public  Int16 ygyro;
            /// <summary> Angular speed around Z axis (raw) </summary>
        public  Int16 zgyro;
            /// <summary> X Magnetic field (raw) </summary>
        public  Int16 xmag;
            /// <summary> Y Magnetic field (raw) </summary>
        public  Int16 ymag;
            /// <summary> Z Magnetic field (raw) </summary>
        public  Int16 zmag;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=16)]
    public struct mavlink_raw_pressure_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Absolute pressure (raw) </summary>
        public  Int16 press_abs;
            /// <summary> Differential pressure 1 (raw) </summary>
        public  Int16 press_diff1;
            /// <summary> Differential pressure 2 (raw) </summary>
        public  Int16 press_diff2;
            /// <summary> Raw Temperature measurement (raw) </summary>
        public  Int16 temperature;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_scaled_pressure_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Absolute pressure (hectopascal) </summary>
        public  Single press_abs;
            /// <summary> Differential pressure 1 (hectopascal) </summary>
        public  Single press_diff;
            /// <summary> Temperature measurement (0.01 degrees celsius)
</summary>
        public  Int16 temperature;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=28)]
    public struct mavlink_attitude_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Roll angle (rad, -pi..+pi) </summary>
        public  Single roll;
            /// <summary> Pitch angle (rad, -pi..+pi) </summary>
```

```
        public  Single pitch;
            /// <summary> Yaw angle (rad, -pi..+pi) </summary>
        public  Single yaw;
            /// <summary> Roll angular speed (rad/s) </summary>
        public  Single rollspeed;
            /// <summary> Pitch angular speed (rad/s) </summary>
        public  Single pitchspeed;
            /// <summary> Yaw angular speed (rad/s) </summary>
        public  Single yawspeed;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=32)]
    public struct mavlink_attitude_quaternion_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Quaternion component 1, w (1 in null-rotation)
</summary>
        public  Single q1;
            /// <summary> Quaternion component 2, x (0 in null-rotation)
</summary>
        public  Single q2;
            /// <summary> Quaternion component 3, y (0 in null-rotation)
</summary>
        public  Single q3;
            /// <summary> Quaternion component 4, z (0 in null-rotation)
</summary>
        public  Single q4;
            /// <summary> Roll angular speed (rad/s) </summary>
        public  Single rollspeed;
            /// <summary> Pitch angular speed (rad/s) </summary>
        public  Single pitchspeed;
            /// <summary> Yaw angular speed (rad/s) </summary>
        public  Single yawspeed;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=28)]
    public struct mavlink_local_position_ned_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X Position </summary>
        public  Single x;
            /// <summary> Y Position </summary>
        public  Single y;
            /// <summary> Z Position </summary>
        public  Single z;
            /// <summary> X Speed </summary>
        public  Single vx;
            /// <summary> Y Speed </summary>
        public  Single vy;
            /// <summary> Z Speed </summary>
        public  Single vz;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=28)]
```

```
    public struct mavlink_global_position_int_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Latitude, expressed as * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude, expressed as * 1E7 </summary>
        public  Int32 lon;
            /// <summary> Altitude in meters, expressed as * 1000 (millimeters),
AMSL (not WGS84 - note that virtually all GPS modules provide the AMSL as well)
</summary>
        public  Int32 alt;
            /// <summary> Altitude above ground in meters, expressed as * 1000
(millimeters) </summary>
        public  Int32 relative_alt;
            /// <summary> Ground X Speed (Latitude), expressed as m/s * 100
</summary>
        public  Int16 vx;
            /// <summary> Ground Y Speed (Longitude), expressed as m/s * 100
</summary>
        public  Int16 vy;
            /// <summary> Ground Z Speed (Altitude), expressed as m/s * 100
</summary>
        public  Int16 vz;
            /// <summary> Compass heading in degrees * 100, 0.0..359.99 degrees.
If unknown, set to: UINT16_MAX </summary>
        public  UInt16 hdg;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_rc_channels_scaled_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> RC channel 1 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan1_scaled;
            /// <summary> RC channel 2 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan2_scaled;
            /// <summary> RC channel 3 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan3_scaled;
            /// <summary> RC channel 4 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan4_scaled;
            /// <summary> RC channel 5 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan5_scaled;
            /// <summary> RC channel 6 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan6_scaled;
            /// <summary> RC channel 7 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan7_scaled;
            /// <summary> RC channel 8 value scaled, (-100%) -10000, (0%) 0,
(100%) 10000, (invalid) INT16_MAX. </summary>
        public  Int16 chan8_scaled;
            /// <summary> Servo output port (set of 8 outputs = 1 port). Most
MAVs will just use one, but this allows for more than 8 servos. </summary>
```

```
        public  byte port;
            /// <summary> Receive signal strength indicator, 0: 0%, 100: 100%,
255: invalid/unknown. </summary>
        public  byte rssi;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_rc_channels_raw_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> RC channel 1 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan1_raw;
            /// <summary> RC channel 2 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan2_raw;
            /// <summary> RC channel 3 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan3_raw;
            /// <summary> RC channel 4 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan4_raw;
            /// <summary> RC channel 5 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan5_raw;
            /// <summary> RC channel 6 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan6_raw;
            /// <summary> RC channel 7 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan7_raw;
            /// <summary> RC channel 8 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan8_raw;
            /// <summary> Servo output port (set of 8 outputs = 1 port). Most
MAVs will just use one, but this allows for more than 8 servos. </summary>
        public  byte port;
            /// <summary> Receive signal strength indicator, 0: 0%, 100: 100%,
255: invalid/unknown. </summary>
        public  byte rssi;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=21)]
    public struct mavlink_servo_output_raw_t
    {
        /// <summary> Timestamp (microseconds since system boot) </summary>
        public  UInt32 time_usec;
            /// <summary> Servo output 1 value, in microseconds </summary>
        public  UInt16 servo1_raw;
            /// <summary> Servo output 2 value, in microseconds </summary>
        public  UInt16 servo2_raw;
            /// <summary> Servo output 3 value, in microseconds </summary>
        public  UInt16 servo3_raw;
            /// <summary> Servo output 4 value, in microseconds </summary>
        public  UInt16 servo4_raw;
            /// <summary> Servo output 5 value, in microseconds </summary>
        public  UInt16 servo5_raw;
```

```
            /// <summary> Servo output 6 value, in microseconds </summary>
        public  UInt16 servo6_raw;
            /// <summary> Servo output 7 value, in microseconds </summary>
        public  UInt16 servo7_raw;
            /// <summary> Servo output 8 value, in microseconds </summary>
        public  UInt16 servo8_raw;
            /// <summary> Servo output port (set of 8 outputs = 1 port). Most
MAVs will just use one, but this allows to encode more than 8 servos. </summary>
        public  byte port;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_mission_request_partial_list_t
    {
        /// <summary> Start index, 0 by default </summary>
        public  Int16 start_index;
            /// <summary> End index, -1 by default (-1: send list to end). Else a
valid index of the list </summary>
        public  Int16 end_index;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_mission_write_partial_list_t
    {
        /// <summary> Start index, 0 by default and smaller / equal to the
largest index of the current onboard list. </summary>
        public  Int16 start_index;
            /// <summary> End index, equal or greater than start index.
</summary>
        public  Int16 end_index;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=37)]
    public struct mavlink_mission_item_t
    {
        /// <summary> PARAM1, see MAV_CMD enum </summary>
        public  Single param1;
            /// <summary> PARAM2, see MAV_CMD enum </summary>
        public  Single param2;
            /// <summary> PARAM3, see MAV_CMD enum </summary>
        public  Single param3;
            /// <summary> PARAM4, see MAV_CMD enum </summary>
        public  Single param4;
            /// <summary> PARAM5 / local: x position, global: latitude </summary>
        public  Single x;
            /// <summary> PARAM6 / y position: global: longitude </summary>
        public  Single y;
```

```
            /// <summary> PARAM7 / z position: global: altitude (relative or
absolute, depending on frame. </summary>
        public  Single z;
            /// <summary> Sequence </summary>
        public  UInt16 seq;
            /// <summary> The scheduled action for the MISSION. see MAV_CMD in
common.xml MAVLink specs </summary>
        public  UInt16 command;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> The coordinate system of the MISSION. see MAV_FRAME in
mavlink_types.h </summary>
        public  byte frame;
            /// <summary> false:0, true:1 </summary>
        public  byte current;
            /// <summary> autocontinue to next wp </summary>
        public  byte autocontinue;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_mission_request_t
    {
        /// <summary> Sequence </summary>
        public  UInt16 seq;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_mission_set_current_t
    {
        /// <summary> Sequence </summary>
        public  UInt16 seq;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_mission_current_t
    {
        /// <summary> Sequence </summary>
        public  UInt16 seq;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_mission_request_list_t
    {
        /// <summary> System ID </summary>
```

```
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_mission_count_t
    {
        /// <summary> Number of mission items in the sequence </summary>
        public  UInt16 count;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_mission_clear_all_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
    public struct mavlink_mission_item_reached_t
    {
        /// <summary> Sequence </summary>
        public  UInt16 seq;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_mission_ack_t
    {
        /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> See MAV_MISSION_RESULT enum </summary>
        public  byte type;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=13)]
    public struct mavlink_set_gps_global_origin_t
    {
        /// <summary> Latitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 latitude;
            /// <summary> Longitude (WGS84, in degrees * 1E7 </summary>
        public  Int32 longitude;
            /// <summary> Altitude (AMSL), in meters * 1000 (positive for up)
</summary>
```

```
        public  Int32 altitude;
            /// <summary> System ID </summary>
        public  byte target_system;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=12)]
    public struct mavlink_gps_global_origin_t
    {
        /// <summary> Latitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 latitude;
            /// <summary> Longitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 longitude;
            /// <summary> Altitude (AMSL), in meters * 1000 (positive for up)
</summary>
        public  Int32 altitude;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=37)]
    public struct mavlink_param_map_rc_t
    {
        /// <summary> Initial parameter value </summary>
        public  Single param_value0;
            /// <summary> Scale, maps the RC range [-1, 1] to a parameter value
</summary>
        public  Single scale;
            /// <summary> Minimum param value. The protocol does not define if
this overwrites an onboard minimum value. (Depends on implementation) </summary>
        public  Single param_value_min;
            /// <summary> Maximum param value. The protocol does not define if
this overwrites an onboard maximum value. (Depends on implementation) </summary>
        public  Single param_value_max;
            /// <summary> Parameter index. Send -1 to use the param ID field as
identifier (else the param id will be ignored), send -2 to disable any existing
map for this rc_channel_index. </summary>
        public  Int16 param_index;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Onboard parameter id, terminated by NULL if the length
is less than 16 human-readable chars and WITHOUT null termination (NULL) byte if
the length is exactly 16 chars - applications have to provide 16+1 bytes storage
if the ID is stored as string </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=16)]
             public byte[] param_id;
            /// <summary> Index of parameter RC channel. Not equal to the RC
channel id. Typically correpsonds to a potentiometer-knob on the RC. </summary>
        public  byte parameter_rc_channel_index;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=27)]
    public struct mavlink_safety_set_allowed_area_t
    {
        /// <summary> x position 1 / Latitude 1 </summary>
        public  Single p1x;
            /// <summary> y position 1 / Longitude 1 </summary>
```

```
        public  Single p1y;
            /// <summary> z position 1 / Altitude 1 </summary>
        public  Single p1z;
            /// <summary> x position 2 / Latitude 2 </summary>
        public  Single p2x;
            /// <summary> y position 2 / Longitude 2 </summary>
        public  Single p2y;
            /// <summary> z position 2 / Altitude 2 </summary>
        public  Single p2z;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Coordinate frame, as defined by MAV_FRAME enum in
mavlink_types.h. Can be either global, GPS, right-handed with Z axis up or local,
right handed, Z axis down. </summary>
        public  byte frame;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=25)]
    public struct mavlink_safety_allowed_area_t
    {
        /// <summary> x position 1 / Latitude 1 </summary>
        public  Single p1x;
            /// <summary> y position 1 / Longitude 1 </summary>
        public  Single p1y;
            /// <summary> z position 1 / Altitude 1 </summary>
        public  Single p1z;
            /// <summary> x position 2 / Latitude 2 </summary>
        public  Single p2x;
            /// <summary> y position 2 / Longitude 2 </summary>
        public  Single p2y;
            /// <summary> z position 2 / Altitude 2 </summary>
        public  Single p2z;
            /// <summary> Coordinate frame, as defined by MAV_FRAME enum in
mavlink_types.h. Can be either global, GPS, right-handed with Z axis up or local,
right handed, Z axis down. </summary>
        public  byte frame;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=68)]
    public struct mavlink_attitude_quaternion_cov_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Quaternion components, w, x, y, z (1 0 0 0 is the null-
rotation) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=4)]
             public float q;
            /// <summary> Roll angular speed (rad/s) </summary>
        public  Single rollspeed;
            /// <summary> Pitch angular speed (rad/s) </summary>
        public  Single pitchspeed;
            /// <summary> Yaw angular speed (rad/s) </summary>
        public  Single yawspeed;
            /// <summary> Attitude covariance </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=9)]
             public float covariance;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=26)]
    public struct mavlink_nav_controller_output_t
    {
        /// <summary> Current desired roll in degrees </summary>
        public  Single nav_roll;
            /// <summary> Current desired pitch in degrees </summary>
        public  Single nav_pitch;
            /// <summary> Current altitude error in meters </summary>
        public  Single alt_error;
            /// <summary> Current airspeed error in meters/second </summary>
        public  Single aspd_error;
            /// <summary> Current crosstrack error on x-y plane in meters
</summary>
        public  Single xtrack_error;
            /// <summary> Current desired heading in degrees </summary>
        public  Int16 nav_bearing;
            /// <summary> Bearing to current MISSION/target in degrees </summary>
        public  Int16 target_bearing;
            /// <summary> Distance to active MISSION in meters </summary>
        public  UInt16 wp_dist;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=185)]
    public struct mavlink_global_position_int_cov_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch) in UTC. 0 for
unknown. Commonly filled by the precision time source of a GPS receiver.
</summary>
        public  UInt64 time_utc;
            /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Latitude, expressed as degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude, expressed as degrees * 1E7 </summary>
        public  Int32 lon;
            /// <summary> Altitude in meters, expressed as * 1000 (millimeters),
above MSL </summary>
        public  Int32 alt;
            /// <summary> Altitude above ground in meters, expressed as * 1000
(millimeters) </summary>
        public  Int32 relative_alt;
            /// <summary> Ground X Speed (Latitude), expressed as m/s </summary>
        public  Single vx;
            /// <summary> Ground Y Speed (Longitude), expressed as m/s </summary>
        public  Single vy;
            /// <summary> Ground Z Speed (Altitude), expressed as m/s </summary>
        public  Single vz;
            /// <summary> Covariance matrix (first six entries are the first ROW,
next six entries are the second row, etc.) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=36)]
             public float covariance;
            /// <summary> Class id of the estimator this estimate originated
from. </summary>
        public  byte estimator_type;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=181)]
    public struct mavlink_local_position_ned_cov_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch) in UTC. 0 for
unknown. Commonly filled by the precision time source of a GPS receiver.
</summary>
        public  UInt64 time_utc;
            /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X Position </summary>
        public  Single x;
            /// <summary> Y Position </summary>
        public  Single y;
            /// <summary> Z Position </summary>
        public  Single z;
            /// <summary> X Speed </summary>
        public  Single vx;
            /// <summary> Y Speed </summary>
        public  Single vy;
            /// <summary> Z Speed </summary>
        public  Single vz;
            /// <summary> Covariance matrix (first six entries are the first ROW,
next six entries are the second row, etc.) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=36)]
              public float covariance;
            /// <summary> Class id of the estimator this estimate originated
from. </summary>
        public  byte estimator_type;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=42)]
    public struct mavlink_rc_channels_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> RC channel 1 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan1_raw;
            /// <summary> RC channel 2 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan2_raw;
            /// <summary> RC channel 3 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan3_raw;
            /// <summary> RC channel 4 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan4_raw;
            /// <summary> RC channel 5 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan5_raw;
            /// <summary> RC channel 6 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan6_raw;
            /// <summary> RC channel 7 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan7_raw;
            /// <summary> RC channel 8 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
```

```
        public  UInt16 chan8_raw;
            /// <summary> RC channel 9 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan9_raw;
            /// <summary> RC channel 10 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan10_raw;
            /// <summary> RC channel 11 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan11_raw;
            /// <summary> RC channel 12 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan12_raw;
            /// <summary> RC channel 13 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan13_raw;
            /// <summary> RC channel 14 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan14_raw;
            /// <summary> RC channel 15 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan15_raw;
            /// <summary> RC channel 16 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan16_raw;
            /// <summary> RC channel 17 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan17_raw;
            /// <summary> RC channel 18 value, in microseconds. A value of
UINT16_MAX implies the channel is unused. </summary>
        public  UInt16 chan18_raw;
            /// <summary> Total number of RC channels being received. This can be
larger than 18, indicating that more channels are available but not given in this
message. This value should be 0 when no RC channels are available. </summary>
        public  byte chancount;
            /// <summary> Receive signal strength indicator, 0: 0%, 100: 100%,
255: invalid/unknown. </summary>
        public  byte rssi;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_request_data_stream_t
    {
        /// <summary> The requested interval between two messages of this type
</summary>
        public  UInt16 req_message_rate;
            /// <summary> The target requested to send the message stream.
</summary>
        public  byte target_system;
            /// <summary> The target requested to send the message stream.
</summary>
        public  byte target_component;
            /// <summary> The ID of the requested data stream </summary>
        public  byte req_stream_id;
            /// <summary> 1 to start sending, 0 to stop sending. </summary>
        public  byte start_stop;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=4)]
    public struct mavlink_data_stream_t
    {
        /// <summary> The requested interval between two messages of this type
</summary>
        public  UInt16 message_rate;
            /// <summary> The ID of the requested data stream </summary>
        public  byte stream_id;
            /// <summary> 1 stream is enabled, 0 stream is stopped. </summary>
        public  byte on_off;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=11)]
    public struct mavlink_manual_control_t
    {
        /// <summary> X-axis, normalized to the range [-1000,1000]. A value of
INT16_MAX indicates that this axis is invalid. Generally corresponds to
forward(1000)-backward(-1000) movement on a joystick and the pitch of a vehicle.
</summary>
        public  Int16 x;
            /// <summary> Y-axis, normalized to the range [-1000,1000]. A value
of INT16_MAX indicates that this axis is invalid. Generally corresponds to left(-
1000)-right(1000) movement on a joystick and the roll of a vehicle. </summary>
        public  Int16 y;
            /// <summary> Z-axis, normalized to the range [-1000,1000]. A value
of INT16_MAX indicates that this axis is invalid. Generally corresponds to a
separate slider movement with maximum being 1000 and minimum being -1000 on a
joystick and the thrust of a vehicle. </summary>
        public  Int16 z;
            /// <summary> R-axis, normalized to the range [-1000,1000]. A value
of INT16_MAX indicates that this axis is invalid. Generally corresponds to a
twisting of the joystick, with counter-clockwise being 1000 and clockwise being -
1000, and the yaw of a vehicle. </summary>
        public  Int16 r;
            /// <summary> A bitfield corresponding to the joystick buttons'
current state, 1 for pressed, 0 for released. The lowest bit corresponds to
Button 1. </summary>
        public  UInt16 buttons;
            /// <summary> The system to be controlled. </summary>
        public  byte target;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=18)]
    public struct mavlink_rc_channels_override_t
    {
        /// <summary> RC channel 1 value, in microseconds. A value of UINT16_MAX
means to ignore this field. </summary>
        public  UInt16 chan1_raw;
            /// <summary> RC channel 2 value, in microseconds. A value of
UINT16_MAX means to ignore this field. </summary>
        public  UInt16 chan2_raw;
            /// <summary> RC channel 3 value, in microseconds. A value of
UINT16_MAX means to ignore this field. </summary>
        public  UInt16 chan3_raw;
            /// <summary> RC channel 4 value, in microseconds. A value of
UINT16_MAX means to ignore this field. </summary>
        public  UInt16 chan4_raw;
```

```
            /// <summary> RC channel 5 value, in microseconds. A value of
UINT16_MAX means to ignore this field. </summary>
        public  UInt16 chan5_raw;
            /// <summary> RC channel 6 value, in microseconds. A value of
UINT16_MAX means to ignore this field. </summary>
        public  UInt16 chan6_raw;
            /// <summary> RC channel 7 value, in microseconds. A value of
UINT16_MAX means to ignore this field. </summary>
        public  UInt16 chan7_raw;
            /// <summary> RC channel 8 value, in microseconds. A value of
UINT16_MAX means to ignore this field. </summary>
        public  UInt16 chan8_raw;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=37)]
    public struct mavlink_mission_item_int_t
    {
        /// <summary> PARAM1, see MAV_CMD enum </summary>
        public  Single param1;
            /// <summary> PARAM2, see MAV_CMD enum </summary>
        public  Single param2;
            /// <summary> PARAM3, see MAV_CMD enum </summary>
        public  Single param3;
            /// <summary> PARAM4, see MAV_CMD enum </summary>
        public  Single param4;
            /// <summary> PARAM5 / local: x position in meters * 1e4, global:
latitude in degrees * 10^7 </summary>
        public  Int32 x;
            /// <summary> PARAM6 / y position: local: x position in meters * 1e4,
global: longitude in degrees *10^7 </summary>
        public  Int32 y;
            /// <summary> PARAM7 / z position: global: altitude in meters
(relative or absolute, depending on frame. </summary>
        public  Single z;
            /// <summary> Waypoint ID (sequence number). Starts at zero.
Increases monotonically for each waypoint, no gaps in the sequence (0,1,2,3,4).
</summary>
        public  UInt16 seq;
            /// <summary> The scheduled action for the MISSION. see MAV_CMD in
common.xml MAVLink specs </summary>
        public  UInt16 command;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> The coordinate system of the MISSION. see MAV_FRAME in
mavlink_types.h </summary>
        public  byte frame;
            /// <summary> false:0, true:1 </summary>
        public  byte current;
            /// <summary> autocontinue to next wp </summary>
        public  byte autocontinue;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=20)]
    public struct mavlink_vfr_hud_t
    {
        /// <summary> Current airspeed in m/s </summary>
        public  Single airspeed;
            /// <summary> Current ground speed in m/s </summary>
        public  Single groundspeed;
            /// <summary> Current altitude (MSL), in meters </summary>
        public  Single alt;
            /// <summary> Current climb rate in meters/second </summary>
        public  Single climb;
            /// <summary> Current heading in degrees, in compass units (0..360,
0=north) </summary>
        public  Int16 heading;
            /// <summary> Current throttle setting in integer percent, 0 to 100
</summary>
        public  UInt16 throttle;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=35)]
    public struct mavlink_command_int_t
    {
        /// <summary> PARAM1, see MAV_CMD enum </summary>
        public  Single param1;
            /// <summary> PARAM2, see MAV_CMD enum </summary>
        public  Single param2;
            /// <summary> PARAM3, see MAV_CMD enum </summary>
        public  Single param3;
            /// <summary> PARAM4, see MAV_CMD enum </summary>
        public  Single param4;
            /// <summary> PARAM5 / local: x position in meters * 1e4, global:
latitude in degrees * 10^7 </summary>
        public  Int32 x;
            /// <summary> PARAM6 / local: y position in meters * 1e4, global:
longitude in degrees * 10^7 </summary>
        public  Int32 y;
            /// <summary> PARAM7 / z position: global: altitude in meters
(relative or absolute, depending on frame. </summary>
        public  Single z;
            /// <summary> The scheduled action for the mission item. see MAV_CMD
in common.xml MAVLink specs </summary>
        public  UInt16 command;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> The coordinate system of the COMMAND. see MAV_FRAME in
mavlink_types.h </summary>
        public  byte frame;
            /// <summary> false:0, true:1 </summary>
        public  byte current;
            /// <summary> autocontinue to next wp </summary>
        public  byte autocontinue;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=33)]
    public struct mavlink_command_long_t
    {
```

```
        /// <summary> Parameter 1, as defined by MAV_CMD enum. </summary>
        public  Single param1;
            /// <summary> Parameter 2, as defined by MAV_CMD enum. </summary>
        public  Single param2;
            /// <summary> Parameter 3, as defined by MAV_CMD enum. </summary>
        public  Single param3;
            /// <summary> Parameter 4, as defined by MAV_CMD enum. </summary>
        public  Single param4;
            /// <summary> Parameter 5, as defined by MAV_CMD enum. </summary>
        public  Single param5;
            /// <summary> Parameter 6, as defined by MAV_CMD enum. </summary>
        public  Single param6;
            /// <summary> Parameter 7, as defined by MAV_CMD enum. </summary>
        public  Single param7;
            /// <summary> Command ID, as defined by MAV_CMD enum. </summary>
        public  UInt16 command;
            /// <summary> System which should execute the command </summary>
        public  byte target_system;
            /// <summary> Component which should execute the command, 0 for all
components </summary>
        public  byte target_component;
            /// <summary> 0: First transmission of this command. 1-255:
Confirmation transmissions (e.g. for kill command) </summary>
        public  byte confirmation;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=3)]
    public struct mavlink_command_ack_t
    {
        /// <summary> Command ID, as defined by MAV_CMD enum. </summary>
        public  UInt16 command;
            /// <summary> See MAV_RESULT enum </summary>
        public  byte result;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_manual_setpoint_t
    {
        /// <summary> Timestamp in milliseconds since system boot </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Desired roll rate in radians per second </summary>
        public  Single roll;
            /// <summary> Desired pitch rate in radians per second </summary>
        public  Single pitch;
            /// <summary> Desired yaw rate in radians per second </summary>
        public  Single yaw;
            /// <summary> Collective thrust, normalized to 0 .. 1 </summary>
        public  Single thrust;
            /// <summary> Flight mode switch position, 0.. 255 </summary>
        public  byte mode_switch;
            /// <summary> Override mode switch position, 0.. 255 </summary>
        public  byte manual_override_switch;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=39)]
    public struct mavlink_set_attitude_target_t
```

```
    {
        /// <summary> Timestamp in milliseconds since system boot </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Attitude quaternion (w, x, y, z order, zero-rotation is
1, 0, 0, 0) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=4)]
             public float q;
            /// <summary> Body roll rate in radians per second </summary>
        public  Single body_roll_rate;
            /// <summary> Body roll rate in radians per second </summary>
        public  Single body_pitch_rate;
            /// <summary> Body roll rate in radians per second </summary>
        public  Single body_yaw_rate;
            /// <summary> Collective thrust, normalized to 0 .. 1 (-1 .. 1 for
vehicles capable of reverse trust) </summary>
        public  Single thrust;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Mappings: If any of these bits are set, the
corresponding input should be ignored: bit 1: body roll rate, bit 2: body pitch
rate, bit 3: body yaw rate. bit 4-bit 6: reserved, bit 7: throttle, bit 8:
attitude </summary>
        public  byte type_mask;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=37)]
    public struct mavlink_attitude_target_t
    {
        /// <summary> Timestamp in milliseconds since system boot </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Attitude quaternion (w, x, y, z order, zero-rotation is
1, 0, 0, 0) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=4)]
             public float q;
            /// <summary> Body roll rate in radians per second </summary>
        public  Single body_roll_rate;
            /// <summary> Body roll rate in radians per second </summary>
        public  Single body_pitch_rate;
            /// <summary> Body roll rate in radians per second </summary>
        public  Single body_yaw_rate;
            /// <summary> Collective thrust, normalized to 0 .. 1 (-1 .. 1 for
vehicles capable of reverse trust) </summary>
        public  Single thrust;
            /// <summary> Mappings: If any of these bits are set, the
corresponding input should be ignored: bit 1: body roll rate, bit 2: body pitch
rate, bit 3: body yaw rate. bit 4-bit 7: reserved, bit 8: attitude </summary>
        public  byte type_mask;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=53)]
    public struct mavlink_set_position_target_local_ned_t
    {
        /// <summary> Timestamp in milliseconds since system boot </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X Position in NED frame in meters </summary>
        public  Single x;
```

```
            /// <summary> Y Position in NED frame in meters </summary>
        public  Single y;
            /// <summary> Z Position in NED frame in meters (note, altitude is
negative in NED) </summary>
        public  Single z;
            /// <summary> X velocity in NED frame in meter / s </summary>
        public  Single vx;
            /// <summary> Y velocity in NED frame in meter / s </summary>
        public  Single vy;
            /// <summary> Z velocity in NED frame in meter / s </summary>
        public  Single vz;
            /// <summary> X acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afx;
            /// <summary> Y acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afy;
            /// <summary> Z acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afz;
            /// <summary> yaw setpoint in rad </summary>
        public  Single yaw;
            /// <summary> yaw rate setpoint in rad/s </summary>
        public  Single yaw_rate;
            /// <summary> Bitmask to indicate which dimensions should be ignored
by the vehicle: a value of 0b0000000000000000 or 0b0000001000000000 indicates
that none of the setpoint dimensions should be ignored. If bit 10 is set the
floats afx afy afz should be interpreted as force instead of acceleration.
Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7:
ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw
rate </summary>
        public  UInt16 type_mask;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Valid options are: MAV_FRAME_LOCAL_NED = 1,
MAV_FRAME_LOCAL_OFFSET_NED = 7, MAV_FRAME_BODY_NED = 8, MAV_FRAME_BODY_OFFSET_NED
= 9 </summary>
        public  byte coordinate_frame;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=51)]
    public struct mavlink_position_target_local_ned_t
    {
        /// <summary> Timestamp in milliseconds since system boot </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X Position in NED frame in meters </summary>
        public  Single x;
            /// <summary> Y Position in NED frame in meters </summary>
        public  Single y;
            /// <summary> Z Position in NED frame in meters (note, altitude is
negative in NED) </summary>
        public  Single z;
            /// <summary> X velocity in NED frame in meter / s </summary>
        public  Single vx;
            /// <summary> Y velocity in NED frame in meter / s </summary>
        public  Single vy;
            /// <summary> Z velocity in NED frame in meter / s </summary>
        public  Single vz;
```

```
            /// <summary> X acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afx;
            /// <summary> Y acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afy;
            /// <summary> Z acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afz;
            /// <summary> yaw setpoint in rad </summary>
        public  Single yaw;
            /// <summary> yaw rate setpoint in rad/s </summary>
        public  Single yaw_rate;
            /// <summary> Bitmask to indicate which dimensions should be ignored
by the vehicle: a value of 0b0000000000000000 or 0b0000001000000000 indicates
that none of the setpoint dimensions should be ignored. If bit 10 is set the
floats afx afy afz should be interpreted as force instead of acceleration.
Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7:
ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw
rate </summary>
        public  UInt16 type_mask;
            /// <summary> Valid options are: MAV_FRAME_LOCAL_NED = 1,
MAV_FRAME_LOCAL_OFFSET_NED = 7, MAV_FRAME_BODY_NED = 8, MAV_FRAME_BODY_OFFSET_NED
= 9 </summary>
        public  byte coordinate_frame;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=53)]
    public struct mavlink_set_position_target_global_int_t
    {
        /// <summary> Timestamp in milliseconds since system boot. The rationale
for the timestamp in the setpoint is to allow the system to compensate for the
transport delay of the setpoint. This allows the system to compensate processing
latency. </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X Position in WGS84 frame in 1e7 * meters </summary>
        public  Int32 lat_int;
            /// <summary> Y Position in WGS84 frame in 1e7 * meters </summary>
        public  Int32 lon_int;
            /// <summary> Altitude in meters in AMSL altitude, not WGS84 if
absolute or relative, above terrain if GLOBAL_TERRAIN_ALT_INT </summary>
        public  Single alt;
            /// <summary> X velocity in NED frame in meter / s </summary>
        public  Single vx;
            /// <summary> Y velocity in NED frame in meter / s </summary>
        public  Single vy;
            /// <summary> Z velocity in NED frame in meter / s </summary>
        public  Single vz;
            /// <summary> X acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afx;
            /// <summary> Y acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afy;
            /// <summary> Z acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afz;
            /// <summary> yaw setpoint in rad </summary>
        public  Single yaw;
            /// <summary> yaw rate setpoint in rad/s </summary>
```

```
        public  Single yaw_rate;
            /// <summary> Bitmask to indicate which dimensions should be ignored
by the vehicle: a value of 0b0000000000000000 or 0b0000001000000000 indicates
that none of the setpoint dimensions should be ignored. If bit 10 is set the
floats afx afy afz should be interpreted as force instead of acceleration.
Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7:
ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw
rate </summary>
        public  UInt16 type_mask;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> Valid options are: MAV_FRAME_GLOBAL_INT = 5,
MAV_FRAME_GLOBAL_RELATIVE_ALT_INT = 6, MAV_FRAME_GLOBAL_TERRAIN_ALT_INT = 11
</summary>
        public  byte coordinate_frame;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=51)]
    public struct mavlink_position_target_global_int_t
    {
        /// <summary> Timestamp in milliseconds since system boot. The rationale
for the timestamp in the setpoint is to allow the system to compensate for the
transport delay of the setpoint. This allows the system to compensate processing
latency. </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X Position in WGS84 frame in 1e7 * meters </summary>
        public  Int32 lat_int;
            /// <summary> Y Position in WGS84 frame in 1e7 * meters </summary>
        public  Int32 lon_int;
            /// <summary> Altitude in meters in AMSL altitude, not WGS84 if
absolute or relative, above terrain if GLOBAL_TERRAIN_ALT_INT </summary>
        public  Single alt;
            /// <summary> X velocity in NED frame in meter / s </summary>
        public  Single vx;
            /// <summary> Y velocity in NED frame in meter / s </summary>
        public  Single vy;
            /// <summary> Z velocity in NED frame in meter / s </summary>
        public  Single vz;
            /// <summary> X acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afx;
            /// <summary> Y acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afy;
            /// <summary> Z acceleration or force (if bit 10 of type_mask is set)
in NED frame in meter / s^2 or N </summary>
        public  Single afz;
            /// <summary> yaw setpoint in rad </summary>
        public  Single yaw;
            /// <summary> yaw rate setpoint in rad/s </summary>
        public  Single yaw_rate;
            /// <summary> Bitmask to indicate which dimensions should be ignored
by the vehicle: a value of 0b0000000000000000 or 0b0000001000000000 indicates
that none of the setpoint dimensions should be ignored. If bit 10 is set the
floats afx afy afz should be interpreted as force instead of acceleration.
Mapping: bit 1: x, bit 2: y, bit 3: z, bit 4: vx, bit 5: vy, bit 6: vz, bit 7:
ax, bit 8: ay, bit 9: az, bit 10: is force setpoint, bit 11: yaw, bit 12: yaw
rate </summary>
```

```
        public  UInt16 type_mask;
            /// <summary> Valid options are: MAV_FRAME_GLOBAL_INT = 5,
MAV_FRAME_GLOBAL_RELATIVE_ALT_INT = 6, MAV_FRAME_GLOBAL_TERRAIN_ALT_INT = 11
</summary>
        public  byte coordinate_frame;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=28)]
    public struct mavlink_local_position_ned_system_global_offset_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X Position </summary>
        public  Single x;
            /// <summary> Y Position </summary>
        public  Single y;
            /// <summary> Z Position </summary>
        public  Single z;
            /// <summary> Roll </summary>
        public  Single roll;
            /// <summary> Pitch </summary>
        public  Single pitch;
            /// <summary> Yaw </summary>
        public  Single yaw;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=56)]
    public struct mavlink_hil_state_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Roll angle (rad) </summary>
        public  Single roll;
            /// <summary> Pitch angle (rad) </summary>
        public  Single pitch;
            /// <summary> Yaw angle (rad) </summary>
        public  Single yaw;
            /// <summary> Body frame roll / phi angular speed (rad/s) </summary>
        public  Single rollspeed;
            /// <summary> Body frame pitch / theta angular speed (rad/s)
</summary>
        public  Single pitchspeed;
            /// <summary> Body frame yaw / psi angular speed (rad/s) </summary>
        public  Single yawspeed;
            /// <summary> Latitude, expressed as * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude, expressed as * 1E7 </summary>
        public  Int32 lon;
            /// <summary> Altitude in meters, expressed as * 1000 (millimeters)
</summary>
        public  Int32 alt;
            /// <summary> Ground X Speed (Latitude), expressed as m/s * 100
</summary>
        public  Int16 vx;
            /// <summary> Ground Y Speed (Longitude), expressed as m/s * 100
</summary>
        public  Int16 vy;
```

```
            /// <summary> Ground Z Speed (Altitude), expressed as m/s * 100
</summary>
        public  Int16 vz;
            /// <summary> X acceleration (mg) </summary>
        public  Int16 xacc;
            /// <summary> Y acceleration (mg) </summary>
        public  Int16 yacc;
            /// <summary> Z acceleration (mg) </summary>
        public  Int16 zacc;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=42)]
    public struct mavlink_hil_controls_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Control output -1 .. 1 </summary>
        public  Single roll_ailerons;
            /// <summary> Control output -1 .. 1 </summary>
        public  Single pitch_elevator;
            /// <summary> Control output -1 .. 1 </summary>
        public  Single yaw_rudder;
            /// <summary> Throttle 0 .. 1 </summary>
        public  Single throttle;
            /// <summary> Aux 1, -1 .. 1 </summary>
        public  Single aux1;
            /// <summary> Aux 2, -1 .. 1 </summary>
        public  Single aux2;
            /// <summary> Aux 3, -1 .. 1 </summary>
        public  Single aux3;
            /// <summary> Aux 4, -1 .. 1 </summary>
        public  Single aux4;
            /// <summary> System mode (MAV_MODE) </summary>
        public  byte mode;
            /// <summary> Navigation mode (MAV_NAV_MODE) </summary>
        public  byte nav_mode;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=33)]
    public struct mavlink_hil_rc_inputs_raw_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> RC channel 1 value, in microseconds </summary>
        public  UInt16 chan1_raw;
            /// <summary> RC channel 2 value, in microseconds </summary>
        public  UInt16 chan2_raw;
            /// <summary> RC channel 3 value, in microseconds </summary>
        public  UInt16 chan3_raw;
            /// <summary> RC channel 4 value, in microseconds </summary>
        public  UInt16 chan4_raw;
            /// <summary> RC channel 5 value, in microseconds </summary>
        public  UInt16 chan5_raw;
            /// <summary> RC channel 6 value, in microseconds </summary>
        public  UInt16 chan6_raw;
            /// <summary> RC channel 7 value, in microseconds </summary>
```

```
        public  UInt16 chan7_raw;
            /// <summary> RC channel 8 value, in microseconds </summary>
        public  UInt16 chan8_raw;
            /// <summary> RC channel 9 value, in microseconds </summary>
        public  UInt16 chan9_raw;
            /// <summary> RC channel 10 value, in microseconds </summary>
        public  UInt16 chan10_raw;
            /// <summary> RC channel 11 value, in microseconds </summary>
        public  UInt16 chan11_raw;
            /// <summary> RC channel 12 value, in microseconds </summary>
        public  UInt16 chan12_raw;
            /// <summary> Receive signal strength indicator, 0: 0%, 255: 100%
</summary>
        public  byte rssi;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=26)]
    public struct mavlink_optical_flow_t
    {
        /// <summary> Timestamp (UNIX) </summary>
        public  UInt64 time_usec;
            /// <summary> Flow in meters in x-sensor direction, angular-speed
compensated </summary>
        public  Single flow_comp_m_x;
            /// <summary> Flow in meters in y-sensor direction, angular-speed
compensated </summary>
        public  Single flow_comp_m_y;
            /// <summary> Ground distance in meters. Positive value: distance
known. Negative value: Unknown distance </summary>
        public  Single ground_distance;
            /// <summary> Flow in pixels * 10 in x-sensor direction (dezi-pixels)
</summary>
        public  Int16 flow_x;
            /// <summary> Flow in pixels * 10 in y-sensor direction (dezi-pixels)
</summary>
        public  Int16 flow_y;
            /// <summary> Sensor ID </summary>
        public  byte sensor_id;
            /// <summary> Optical flow quality / confidence. 0: bad, 255: maximum
quality </summary>
        public  byte quality;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=32)]
    public struct mavlink_global_vision_position_estimate_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 usec;
            /// <summary> Global X position </summary>
        public  Single x;
            /// <summary> Global Y position </summary>
        public  Single y;
            /// <summary> Global Z position </summary>
        public  Single z;
            /// <summary> Roll angle in rad </summary>
        public  Single roll;
            /// <summary> Pitch angle in rad </summary>
```

```
        public  Single pitch;
            /// <summary> Yaw angle in rad </summary>
        public  Single yaw;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=32)]
    public struct mavlink_vision_position_estimate_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 usec;
            /// <summary> Global X position </summary>
        public  Single x;
            /// <summary> Global Y position </summary>
        public  Single y;
            /// <summary> Global Z position </summary>
        public  Single z;
            /// <summary> Roll angle in rad </summary>
        public  Single roll;
            /// <summary> Pitch angle in rad </summary>
        public  Single pitch;
            /// <summary> Yaw angle in rad </summary>
        public  Single yaw;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=20)]
    public struct mavlink_vision_speed_estimate_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 usec;
            /// <summary> Global X speed </summary>
        public  Single x;
            /// <summary> Global Y speed </summary>
        public  Single y;
            /// <summary> Global Z speed </summary>
        public  Single z;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=32)]
    public struct mavlink_vicon_position_estimate_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 usec;
            /// <summary> Global X position </summary>
        public  Single x;
            /// <summary> Global Y position </summary>
        public  Single y;
            /// <summary> Global Z position </summary>
        public  Single z;
            /// <summary> Roll angle in rad </summary>
        public  Single roll;
            /// <summary> Pitch angle in rad </summary>
        public  Single pitch;
            /// <summary> Yaw angle in rad </summary>
```

```
        public  Single yaw;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=62)]
    public struct mavlink_highres_imu_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> X acceleration (m/s^2) </summary>
        public  Single xacc;
            /// <summary> Y acceleration (m/s^2) </summary>
        public  Single yacc;
            /// <summary> Z acceleration (m/s^2) </summary>
        public  Single zacc;
            /// <summary> Angular speed around X axis (rad / sec) </summary>
        public  Single xgyro;
            /// <summary> Angular speed around Y axis (rad / sec) </summary>
        public  Single ygyro;
            /// <summary> Angular speed around Z axis (rad / sec) </summary>
        public  Single zgyro;
            /// <summary> X Magnetic field (Gauss) </summary>
        public  Single xmag;
            /// <summary> Y Magnetic field (Gauss) </summary>
        public  Single ymag;
            /// <summary> Z Magnetic field (Gauss) </summary>
        public  Single zmag;
            /// <summary> Absolute pressure in millibar </summary>
        public  Single abs_pressure;
            /// <summary> Differential pressure in millibar </summary>
        public  Single diff_pressure;
            /// <summary> Altitude calculated from pressure </summary>
        public  Single pressure_alt;
            /// <summary> Temperature in degrees celsius </summary>
        public  Single temperature;
            /// <summary> Bitmask for fields that have updated since last
message, bit 0 = xacc, bit 12: temperature </summary>
        public  UInt16 fields_updated;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=44)]
    public struct mavlink_optical_flow_rad_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Integration time in microseconds. Divide integrated_x
and integrated_y by the integration time to obtain average flow. The integration
time also indicates the. </summary>
        public  UInt32 integration_time_us;
            /// <summary> Flow in radians around X axis (Sensor RH rotation about
the X axis induces a positive flow. Sensor linear motion along the positive Y
axis induces a negative flow.) </summary>
        public  Single integrated_x;
            /// <summary> Flow in radians around Y axis (Sensor RH rotation about
the Y axis induces a positive flow. Sensor linear motion along the positive X
axis induces a positive flow.) </summary>
        public  Single integrated_y;
```

```
            /// <summary> RH rotation around X axis (rad) </summary>
        public  Single integrated_xgyro;
            /// <summary> RH rotation around Y axis (rad) </summary>
        public  Single integrated_ygyro;
            /// <summary> RH rotation around Z axis (rad) </summary>
        public  Single integrated_zgyro;
            /// <summary> Time in microseconds since the distance was sampled.
</summary>
        public  UInt32 time_delta_distance_us;
            /// <summary> Distance to the center of the flow field in meters.
Positive value (including zero): distance known. Negative value: Unknown
distance. </summary>
        public  Single distance;
            /// <summary> Temperature * 100 in centi-degrees Celsius </summary>
        public  Int16 temperature;
            /// <summary> Sensor ID </summary>
        public  byte sensor_id;
            /// <summary> Optical flow quality / confidence. 0: no valid flow,
255: maximum quality </summary>
        public  byte quality;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=64)]
    public struct mavlink_hil_sensor_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> X acceleration (m/s^2) </summary>
        public  Single xacc;
            /// <summary> Y acceleration (m/s^2) </summary>
        public  Single yacc;
            /// <summary> Z acceleration (m/s^2) </summary>
        public  Single zacc;
            /// <summary> Angular speed around X axis in body frame (rad / sec)
</summary>
        public  Single xgyro;
            /// <summary> Angular speed around Y axis in body frame (rad / sec)
</summary>
        public  Single ygyro;
            /// <summary> Angular speed around Z axis in body frame (rad / sec)
</summary>
        public  Single zgyro;
            /// <summary> X Magnetic field (Gauss) </summary>
        public  Single xmag;
            /// <summary> Y Magnetic field (Gauss) </summary>
        public  Single ymag;
            /// <summary> Z Magnetic field (Gauss) </summary>
        public  Single zmag;
            /// <summary> Absolute pressure in millibar </summary>
        public  Single abs_pressure;
            /// <summary> Differential pressure (airspeed) in millibar </summary>
        public  Single diff_pressure;
            /// <summary> Altitude calculated from pressure </summary>
        public  Single pressure_alt;
            /// <summary> Temperature in degrees celsius </summary>
        public  Single temperature;
            /// <summary> Bitmask for fields that have updated since last
message, bit 0 = xacc, bit 12: temperature </summary>
        public  UInt32 fields_updated;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=84)]
    public struct mavlink_sim_state_t
    {
        /// <summary> True attitude quaternion component 1, w (1 in null-
rotation) </summary>
        public  Single q1;
            /// <summary> True attitude quaternion component 2, x (0 in null-
rotation) </summary>
        public  Single q2;
            /// <summary> True attitude quaternion component 3, y (0 in null-
rotation) </summary>
        public  Single q3;
            /// <summary> True attitude quaternion component 4, z (0 in null-
rotation) </summary>
        public  Single q4;
            /// <summary> Attitude roll expressed as Euler angles, not
recommended except for human-readable outputs </summary>
        public  Single roll;
            /// <summary> Attitude pitch expressed as Euler angles, not
recommended except for human-readable outputs </summary>
        public  Single pitch;
            /// <summary> Attitude yaw expressed as Euler angles, not recommended
except for human-readable outputs </summary>
        public  Single yaw;
            /// <summary> X acceleration m/s/s </summary>
        public  Single xacc;
            /// <summary> Y acceleration m/s/s </summary>
        public  Single yacc;
            /// <summary> Z acceleration m/s/s </summary>
        public  Single zacc;
            /// <summary> Angular speed around X axis rad/s </summary>
        public  Single xgyro;
            /// <summary> Angular speed around Y axis rad/s </summary>
        public  Single ygyro;
            /// <summary> Angular speed around Z axis rad/s </summary>
        public  Single zgyro;
            /// <summary> Latitude in degrees </summary>
        public  Single lat;
            /// <summary> Longitude in degrees </summary>
        public  Single lon;
            /// <summary> Altitude in meters </summary>
        public  Single alt;
            /// <summary> Horizontal position standard deviation </summary>
        public  Single std_dev_horz;
            /// <summary> Vertical position standard deviation </summary>
        public  Single std_dev_vert;
            /// <summary> True velocity in m/s in NORTH direction in earth-fixed
NED frame </summary>
        public  Single vn;
            /// <summary> True velocity in m/s in EAST direction in earth-fixed
NED frame </summary>
        public  Single ve;
            /// <summary> True velocity in m/s in DOWN direction in earth-fixed
NED frame </summary>
        public  Single vd;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=9)]
    public struct mavlink_radio_status_t
    {
        /// <summary> Receive errors </summary>
        public  UInt16 rxerrors;
            /// <summary> Count of error corrected packets </summary>
        public  UInt16 @fixed;
            /// <summary> Local signal strength </summary>
        public  byte rssi;
            /// <summary> Remote signal strength </summary>
        public  byte remrssi;
            /// <summary> Remaining free buffer space in percent. </summary>
        public  byte txbuf;
            /// <summary> Background noise level </summary>
        public  byte noise;
            /// <summary> Remote background noise level </summary>
        public  byte remnoise;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=254)]
    public struct mavlink_file_transfer_protocol_t
    {
        /// <summary> Network ID (0 for broadcast) </summary>
        public  byte target_network;
            /// <summary> System ID (0 for broadcast) </summary>
        public  byte target_system;
            /// <summary> Component ID (0 for broadcast) </summary>
        public  byte target_component;
            /// <summary> Variable length payload. The length is defined by the
remaining message length when subtracting the header and other fields.  The
entire content of this block is opaque unless you understand any the encoding
message_type.  The particular encoding used can be extension specific and might
not always be documented as part of the mavlink specification. </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=251)]
             public byte[] payload;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=16)]
    public struct mavlink_timesync_t
    {
        /// <summary> Time sync timestamp 1 </summary>
        public  Int64 tc1;
            /// <summary> Time sync timestamp 2 </summary>
        public  Int64 ts1;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=36)]
    public struct mavlink_hil_gps_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Latitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude (WGS84), in degrees * 1E7 </summary>
```

```
        public  Int32 lon;
            /// <summary> Altitude (AMSL, not WGS84), in meters * 1000 (positive
for up) </summary>
        public  Int32 alt;
            /// <summary> GPS HDOP horizontal dilution of position in cm (m*100).
If unknown, set to: 65535 </summary>
        public  UInt16 eph;
            /// <summary> GPS VDOP vertical dilution of position in cm (m*100).
If unknown, set to: 65535 </summary>
        public  UInt16 epv;
            /// <summary> GPS ground speed (m/s * 100). If unknown, set to: 65535
</summary>
        public  UInt16 vel;
            /// <summary> GPS velocity in cm/s in NORTH direction in earth-fixed
NED frame </summary>
        public  Int16 vn;
            /// <summary> GPS velocity in cm/s in EAST direction in earth-fixed
NED frame </summary>
        public  Int16 ve;
            /// <summary> GPS velocity in cm/s in DOWN direction in earth-fixed
NED frame </summary>
        public  Int16 vd;
            /// <summary> Course over ground (NOT heading, but direction of
movement) in degrees * 100, 0.0..359.99 degrees. If unknown, set to: 65535
</summary>
        public  UInt16 cog;
            /// <summary> 0-1: no fix, 2: 2D fix, 3: 3D fix. Some applications
will not use the value of this field unless it is at least two, so always
correctly fill in the fix. </summary>
        public  byte fix_type;
            /// <summary> Number of satellites visible. If unknown, set to 255
</summary>
        public  byte satellites_visible;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=44)]
    public struct mavlink_hil_optical_flow_t
    {
        /// <summary> Timestamp (microseconds, synced to UNIX time or since
system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Integration time in microseconds. Divide integrated_x
and integrated_y by the integration time to obtain average flow. The integration
time also indicates the. </summary>
        public  UInt32 integration_time_us;
            /// <summary> Flow in radians around X axis (Sensor RH rotation about
the X axis induces a positive flow. Sensor linear motion along the positive Y
axis induces a negative flow.) </summary>
        public  Single integrated_x;
            /// <summary> Flow in radians around Y axis (Sensor RH rotation about
the Y axis induces a positive flow. Sensor linear motion along the positive X
axis induces a positive flow.) </summary>
        public  Single integrated_y;
            /// <summary> RH rotation around X axis (rad) </summary>
        public  Single integrated_xgyro;
            /// <summary> RH rotation around Y axis (rad) </summary>
        public  Single integrated_ygyro;
            /// <summary> RH rotation around Z axis (rad) </summary>
        public  Single integrated_zgyro;
```

```
            /// <summary> Time in microseconds since the distance was sampled.
</summary>
        public  UInt32 time_delta_distance_us;
            /// <summary> Distance to the center of the flow field in meters.
Positive value (including zero): distance known. Negative value: Unknown
distance. </summary>
        public  Single distance;
            /// <summary> Temperature * 100 in centi-degrees Celsius </summary>
        public  Int16 temperature;
            /// <summary> Sensor ID </summary>
        public  byte sensor_id;
            /// <summary> Optical flow quality / confidence. 0: no valid flow,
255: maximum quality </summary>
        public  byte quality;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=64)]
    public struct mavlink_hil_state_quaternion_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Vehicle attitude expressed as normalized quaternion in
w, x, y, z order (with 1 0 0 0 being the null-rotation) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=4)]
              public float attitude_quaternion;
            /// <summary> Body frame roll / phi angular speed (rad/s) </summary>
        public  Single rollspeed;
            /// <summary> Body frame pitch / theta angular speed (rad/s)
</summary>
        public  Single pitchspeed;
            /// <summary> Body frame yaw / psi angular speed (rad/s) </summary>
        public  Single yawspeed;
            /// <summary> Latitude, expressed as * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude, expressed as * 1E7 </summary>
        public  Int32 lon;
            /// <summary> Altitude in meters, expressed as * 1000 (millimeters)
</summary>
        public  Int32 alt;
            /// <summary> Ground X Speed (Latitude), expressed as m/s * 100
</summary>
        public  Int16 vx;
            /// <summary> Ground Y Speed (Longitude), expressed as m/s * 100
</summary>
        public  Int16 vy;
            /// <summary> Ground Z Speed (Altitude), expressed as m/s * 100
</summary>
        public  Int16 vz;
            /// <summary> Indicated airspeed, expressed as m/s * 100 </summary>
        public  UInt16 ind_airspeed;
            /// <summary> True airspeed, expressed as m/s * 100 </summary>
        public  UInt16 true_airspeed;
            /// <summary> X acceleration (mg) </summary>
        public  Int16 xacc;
            /// <summary> Y acceleration (mg) </summary>
        public  Int16 yacc;
            /// <summary> Z acceleration (mg) </summary>
        public  Int16 zacc;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_scaled_imu2_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X acceleration (mg) </summary>
        public  Int16 xacc;
            /// <summary> Y acceleration (mg) </summary>
        public  Int16 yacc;
            /// <summary> Z acceleration (mg) </summary>
        public  Int16 zacc;
            /// <summary> Angular speed around X axis (millirad /sec) </summary>
        public  Int16 xgyro;
            /// <summary> Angular speed around Y axis (millirad /sec) </summary>
        public  Int16 ygyro;
            /// <summary> Angular speed around Z axis (millirad /sec) </summary>
        public  Int16 zgyro;
            /// <summary> X Magnetic field (milli tesla) </summary>
        public  Int16 xmag;
            /// <summary> Y Magnetic field (milli tesla) </summary>
        public  Int16 ymag;
            /// <summary> Z Magnetic field (milli tesla) </summary>
        public  Int16 zmag;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_log_request_list_t
    {
        /// <summary> First log id (0 for first available) </summary>
        public  UInt16 start;
            /// <summary> Last log id (0xffff for last available) </summary>
        public  UInt16 end;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_log_entry_t
    {
        /// <summary> UTC timestamp of log in seconds since 1970, or 0 if not
available </summary>
        public  UInt32 time_utc;
            /// <summary> Size of the log (may be approximate) in bytes
</summary>
        public  UInt32 size;
            /// <summary> Log id </summary>
        public  UInt16 id;
            /// <summary> Total number of logs </summary>
        public  UInt16 num_logs;
            /// <summary> High log number </summary>
        public  UInt16 last_log_num;

    };
```

```
[StructLayout(LayoutKind.Sequential,Pack=1,Size=12)]
public struct mavlink_log_request_data_t
{
    /// <summary> Offset into the log </summary>
    public  UInt32 ofs;
        /// <summary> Number of bytes </summary>
    public  UInt32 count;
        /// <summary> Log id (from LOG_ENTRY reply) </summary>
    public  UInt16 id;
        /// <summary> System ID </summary>
    public  byte target_system;
        /// <summary> Component ID </summary>
    public  byte target_component;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=97)]
public struct mavlink_log_data_t
{
    /// <summary> Offset into the log </summary>
    public  UInt32 ofs;
        /// <summary> Log id (from LOG_ENTRY reply) </summary>
    public  UInt16 id;
        /// <summary> Number of bytes (zero for end of log) </summary>
    public  byte count;
        /// <summary> log data </summary>
    [MarshalAs(UnmanagedType.ByValArray,SizeConst=90)]
          public byte[] data;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
public struct mavlink_log_erase_t
{
    /// <summary> System ID </summary>
    public  byte target_system;
        /// <summary> Component ID </summary>
    public  byte target_component;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=2)]
public struct mavlink_log_request_end_t
{
    /// <summary> System ID </summary>
    public  byte target_system;
        /// <summary> Component ID </summary>
    public  byte target_component;

};


[StructLayout(LayoutKind.Sequential,Pack=1,Size=113)]
public struct mavlink_gps_inject_data_t
{
    /// <summary> System ID </summary>
    public  byte target_system;
```

```
            /// <summary> Component ID </summary>
        public  byte target_component;
            /// <summary> data length </summary>
        public  byte len;
            /// <summary> raw data (110 is enough for 12 satellites of RTCMv2)
</summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=110)]
             public byte[] data;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=35)]
    public struct mavlink_gps2_raw_t
    {
        /// <summary> Timestamp (microseconds since UNIX epoch or microseconds
since system boot) </summary>
        public  UInt64 time_usec;
            /// <summary> Latitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 lon;
            /// <summary> Altitude (AMSL, not WGS84), in meters * 1000 (positive
for up) </summary>
        public  Int32 alt;
            /// <summary> Age of DGPS info </summary>
        public  UInt32 dgps_age;
            /// <summary> GPS HDOP horizontal dilution of position in cm (m*100).
If unknown, set to: UINT16_MAX </summary>
        public  UInt16 eph;
            /// <summary> GPS VDOP vertical dilution of position in cm (m*100).
If unknown, set to: UINT16_MAX </summary>
        public  UInt16 epv;
            /// <summary> GPS ground speed (m/s * 100). If unknown, set to:
UINT16_MAX </summary>
        public  UInt16 vel;
            /// <summary> Course over ground (NOT heading, but direction of
movement) in degrees * 100, 0.0..359.99 degrees. If unknown, set to: UINT16_MAX
</summary>
        public  UInt16 cog;
            /// <summary> 0-1: no fix, 2: 2D fix, 3: 3D fix, 4: DGPS fix, 5: RTK
Fix. Some applications will not use the value of this field unless it is at least
two, so always correctly fill in the fix. </summary>
        public  byte fix_type;
            /// <summary> Number of satellites visible. If unknown, set to 255
</summary>
        public  byte satellites_visible;
            /// <summary> Number of DGPS satellites </summary>
        public  byte dgps_numch;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=6)]
    public struct mavlink_power_status_t
    {
        /// <summary> 5V rail voltage in millivolts </summary>
        public  UInt16 Vcc;
            /// <summary> servo rail voltage in millivolts </summary>
        public  UInt16 Vservo;
            /// <summary> power supply status flags (see MAV_POWER_STATUS enum)
</summary>
```

```
        public  UInt16 flags;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=79)]
    public struct mavlink_serial_control_t
    {
        /// <summary> Baudrate of transfer. Zero means no change. </summary>
        public  UInt32 baudrate;
            /// <summary> Timeout for reply data in milliseconds </summary>
        public  UInt16 timeout;
            /// <summary> See SERIAL_CONTROL_DEV enum </summary>
        public  byte device;
            /// <summary> See SERIAL_CONTROL_FLAG enum </summary>
        public  byte flags;
            /// <summary> how many bytes in this transfer </summary>
        public  byte count;
            /// <summary> serial data </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=70)]
              public byte[] data;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=35)]
    public struct mavlink_gps_rtk_t
    {
        /// <summary> Time since boot of last baseline message received in ms.
</summary>
        public  UInt32 time_last_baseline_ms;
            /// <summary> GPS Time of Week of last baseline </summary>
        public  UInt32 tow;
            /// <summary> Current baseline in ECEF x or NED north component in
mm. </summary>
        public  Int32 baseline_a_mm;
            /// <summary> Current baseline in ECEF y or NED east component in mm.
</summary>
        public  Int32 baseline_b_mm;
            /// <summary> Current baseline in ECEF z or NED down component in mm.
</summary>
        public  Int32 baseline_c_mm;
            /// <summary> Current estimate of baseline accuracy. </summary>
        public  UInt32 accuracy;
            /// <summary> Current number of integer ambiguity hypotheses.
</summary>
        public  Int32 iar_num_hypotheses;
            /// <summary> GPS Week Number of last baseline </summary>
        public  UInt16 wn;
            /// <summary> Identification of connected RTK receiver. </summary>
        public  byte rtk_receiver_id;
            /// <summary> GPS-specific health report for RTK data. </summary>
        public  byte rtk_health;
            /// <summary> Rate of baseline messages being received by GPS, in HZ
</summary>
        public  byte rtk_rate;
            /// <summary> Current number of sats used for RTK calculation.
</summary>
        public  byte nsats;
            /// <summary> Coordinate system of baseline. 0 == ECEF, 1 == NED
</summary>
        public  byte baseline_coords_type;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=35)]
    public struct mavlink_gps2_rtk_t
    {
        /// <summary> Time since boot of last baseline message received in ms.
</summary>
        public  UInt32 time_last_baseline_ms;
            /// <summary> GPS Time of Week of last baseline </summary>
        public  UInt32 tow;
            /// <summary> Current baseline in ECEF x or NED north component in
mm. </summary>
        public  Int32 baseline_a_mm;
            /// <summary> Current baseline in ECEF y or NED east component in mm.
</summary>
        public  Int32 baseline_b_mm;
            /// <summary> Current baseline in ECEF z or NED down component in mm.
</summary>
        public  Int32 baseline_c_mm;
            /// <summary> Current estimate of baseline accuracy. </summary>
        public  UInt32 accuracy;
            /// <summary> Current number of integer ambiguity hypotheses.
</summary>
        public  Int32 iar_num_hypotheses;
            /// <summary> GPS Week Number of last baseline </summary>
        public  UInt16 wn;
            /// <summary> Identification of connected RTK receiver. </summary>
        public  byte rtk_receiver_id;
            /// <summary> GPS-specific health report for RTK data. </summary>
        public  byte rtk_health;
            /// <summary> Rate of baseline messages being received by GPS, in HZ
</summary>
        public  byte rtk_rate;
            /// <summary> Current number of sats used for RTK calculation.
</summary>
        public  byte nsats;
            /// <summary> Coordinate system of baseline. 0 == ECEF, 1 == NED
</summary>
        public  byte baseline_coords_type;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_scaled_imu3_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> X acceleration (mg) </summary>
        public  Int16 xacc;
            /// <summary> Y acceleration (mg) </summary>
        public  Int16 yacc;
            /// <summary> Z acceleration (mg) </summary>
        public  Int16 zacc;
            /// <summary> Angular speed around X axis (millirad /sec) </summary>
        public  Int16 xgyro;
            /// <summary> Angular speed around Y axis (millirad /sec) </summary>
        public  Int16 ygyro;
            /// <summary> Angular speed around Z axis (millirad /sec) </summary>
        public  Int16 zgyro;
```

```
            /// <summary> X Magnetic field (milli tesla) </summary>
        public  Int16 xmag;
            /// <summary> Y Magnetic field (milli tesla) </summary>
        public  Int16 ymag;
            /// <summary> Z Magnetic field (milli tesla) </summary>
        public  Int16 zmag;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=13)]
    public struct mavlink_data_transmission_handshake_t
    {
        /// <summary> total data size in bytes (set on ACK only) </summary>
        public  UInt32 size;
            /// <summary> Width of a matrix or image </summary>
        public  UInt16 width;
            /// <summary> Height of a matrix or image </summary>
        public  UInt16 height;
            /// <summary> number of packets beeing sent (set on ACK only)
</summary>
        public  UInt16 packets;
            /// <summary> type of requested/acknowledged data (as defined in ENUM
DATA_TYPES in mavlink/include/mavlink_types.h) </summary>
        public  byte type;
            /// <summary> payload size per packet (normally 253 byte, see DATA
field size in message ENCAPSULATED_DATA) (set on ACK only) </summary>
        public  byte payload;
            /// <summary> JPEG quality out of [1,100] </summary>
        public  byte jpg_quality;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=255)]
    public struct mavlink_encapsulated_data_t
    {
        /// <summary> sequence number (starting with 0 on every transmission)
</summary>
        public  UInt16 seqnr;
            /// <summary> image data bytes </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=253)]
              public byte[] data;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_distance_sensor_t
    {
        /// <summary> Time since system boot </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Minimum distance the sensor can measure in centimeters
</summary>
        public  UInt16 min_distance;
            /// <summary> Maximum distance the sensor can measure in centimeters
</summary>
        public  UInt16 max_distance;
            /// <summary> Current distance reading </summary>
        public  UInt16 current_distance;
            /// <summary> Type from MAV_DISTANCE_SENSOR enum. </summary>
        public  byte type;
```

```
            /// <summary> Onboard ID of the sensor </summary>
        public  byte id;
            /// <summary> Direction the sensor faces from FIXME enum. </summary>
        public  byte orientation;
            /// <summary> Measurement covariance in centimeters, 0 for unknown /
invalid readings </summary>
        public  byte covariance;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=18)]
    public struct mavlink_terrain_request_t
    {
        /// <summary> Bitmask of requested 4x4 grids (row major 8x7 array of
grids, 56 bits) </summary>
        public  UInt64 mask;
            /// <summary> Latitude of SW corner of first grid (degrees *10^7)
</summary>
        public  Int32 lat;
            /// <summary> Longitude of SW corner of first grid (in degrees *10^7)
</summary>
        public  Int32 lon;
            /// <summary> Grid spacing in meters </summary>
        public  UInt16 grid_spacing;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=43)]
    public struct mavlink_terrain_data_t
    {
        /// <summary> Latitude of SW corner of first grid (degrees *10^7)
</summary>
        public  Int32 lat;
            /// <summary> Longitude of SW corner of first grid (in degrees *10^7)
</summary>
        public  Int32 lon;
            /// <summary> Grid spacing in meters </summary>
        public  UInt16 grid_spacing;
            /// <summary> Terrain data in meters AMSL </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=16)]
              public Int16[] data;
            /// <summary> bit within the terrain request mask </summary>
        public  byte gridbit;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=8)]
    public struct mavlink_terrain_check_t
    {
        /// <summary> Latitude (degrees *10^7) </summary>
        public  Int32 lat;
            /// <summary> Longitude (degrees *10^7) </summary>
        public  Int32 lon;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=22)]
    public struct mavlink_terrain_report_t
```

```
    {
        /// <summary> Latitude (degrees *10^7) </summary>
        public  Int32 lat;
            /// <summary> Longitude (degrees *10^7) </summary>
        public  Int32 lon;
            /// <summary> Terrain height in meters AMSL </summary>
        public  Single terrain_height;
            /// <summary> Current vehicle height above lat/lon terrain height
(meters) </summary>
        public  Single current_height;
            /// <summary> grid spacing (zero if terrain at this location
unavailable) </summary>
        public  UInt16 spacing;
            /// <summary> Number of 4x4 terrain blocks waiting to be received or
read from disk </summary>
        public  UInt16 pending;
            /// <summary> Number of 4x4 terrain blocks in memory </summary>
        public  UInt16 loaded;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_scaled_pressure2_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Absolute pressure (hectopascal) </summary>
        public  Single press_abs;
            /// <summary> Differential pressure 1 (hectopascal) </summary>
        public  Single press_diff;
            /// <summary> Temperature measurement (0.01 degrees celsius)
</summary>
        public  Int16 temperature;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=36)]
    public struct mavlink_att_pos_mocap_t
    {
        /// <summary> Timestamp (micros since boot or Unix epoch) </summary>
        public  UInt64 time_usec;
            /// <summary> Attitude quaternion (w, x, y, z order, zero-rotation is
1, 0, 0, 0) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=4)]
              public float q;
            /// <summary> X position in meters (NED) </summary>
        public  Single x;
            /// <summary> Y position in meters (NED) </summary>
        public  Single y;
            /// <summary> Z position in meters (NED) </summary>
        public  Single z;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=43)]
    public struct mavlink_set_actuator_control_target_t
    {
        /// <summary> Timestamp (micros since boot or Unix epoch) </summary>
        public  UInt64 time_usec;
```

```
            /// <summary> Actuator controls. Normed to -1..+1 where 0 is neutral
position. Throttle for single rotation direction motors is 0..1, negative range
for reverse direction. Standard mapping for attitude controls (group 0): (index
0-7): roll, pitch, yaw, throttle, flaps, spoilers, airbrakes, landing gear. Load
a pass-through mixer to repurpose them as generic outputs. </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=8)]
             public float controls;
            /// <summary> Actuator group. The "_mlx" indicates this is a multi-
instance message and a MAVLink parser should use this field to difference between
instances. </summary>
        public  byte group_mlx;
            /// <summary> System ID </summary>
        public  byte target_system;
            /// <summary> Component ID </summary>
        public  byte target_component;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=41)]
    public struct mavlink_actuator_control_target_t
    {
        /// <summary> Timestamp (micros since boot or Unix epoch) </summary>
        public  UInt64 time_usec;
            /// <summary> Actuator controls. Normed to -1..+1 where 0 is neutral
position. Throttle for single rotation direction motors is 0..1, negative range
for reverse direction. Standard mapping for attitude controls (group 0): (index
0-7): roll, pitch, yaw, throttle, flaps, spoilers, airbrakes, landing gear. Load
a pass-through mixer to repurpose them as generic outputs. </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=8)]
             public float controls;
            /// <summary> Actuator group. The "_mlx" indicates this is a multi-
instance message and a MAVLink parser should use this field to difference between
instances. </summary>
        public  byte group_mlx;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=14)]
    public struct mavlink_scaled_pressure3_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Absolute pressure (hectopascal) </summary>
        public  Single press_abs;
            /// <summary> Differential pressure 1 (hectopascal) </summary>
        public  Single press_diff;
            /// <summary> Temperature measurement (0.01 degrees celsius)
</summary>
        public  Int16 temperature;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=36)]
    public struct mavlink_battery_status_t
    {
        /// <summary> Consumed charge, in milliampere hours (1 = 1 mAh), -1:
autopilot does not provide mAh consumption estimate </summary>
        public  Int32 current_consumed;
```

```
            /// <summary> Consumed energy, in 100*Joules (intergrated U*I*dt)  (1
= 100 Joule), -1: autopilot does not provide energy consumption estimate
</summary>
        public  Int32 energy_consumed;
            /// <summary> Temperature of the battery in centi-degrees celsius.
INT16_MAX for unknown temperature. </summary>
        public  Int16 temperature;
            /// <summary> Battery voltage of cells, in millivolts (1 = 1
millivolt) </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=10)]
             public UInt16[] voltages;
            /// <summary> Battery current, in 10*milliamperes (1 = 10
milliampere), -1: autopilot does not measure the current </summary>
        public  Int16 current_battery;
            /// <summary> Battery ID </summary>
        public  byte id;
            /// <summary> Function of the battery </summary>
        public  byte battery_function;
            /// <summary> Type (chemistry) of the battery </summary>
        public  byte type;
            /// <summary> Remaining battery energy: (0%: 0, 100%: 100), -1:
autopilot does not estimate the remaining battery </summary>
        public  byte battery_remaining;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=60)]
    public struct mavlink_autopilot_version_t
    {
        /// <summary> bitmask of capabilities (see MAV_PROTOCOL_CAPABILITY enum)
</summary>
        public  UInt64 capabilities;
            /// <summary> UID if provided by hardware </summary>
        public  UInt64 uid;
            /// <summary> Firmware version number </summary>
        public  UInt32 flight_sw_version;
            /// <summary> Middleware version number </summary>
        public  UInt32 middleware_sw_version;
            /// <summary> Operating system version number </summary>
        public  UInt32 os_sw_version;
            /// <summary> HW / board version (last 8 bytes should be silicon ID,
if any) </summary>
        public  UInt32 board_version;
            /// <summary> ID of the board vendor </summary>
        public  UInt16 vendor_id;
            /// <summary> ID of the product </summary>
        public  UInt16 product_id;
            /// <summary> Custom version field, commonly the first 8 bytes of the
git hash. This is not an unique identifier, but should allow to identify the
commit using the main version number even for very large code bases. </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=8)]
             public byte[] flight_custom_version;
            /// <summary> Custom version field, commonly the first 8 bytes of the
git hash. This is not an unique identifier, but should allow to identify the
commit using the main version number even for very large code bases. </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=8)]
             public byte[] middleware_custom_version;
            /// <summary> Custom version field, commonly the first 8 bytes of the
git hash. This is not an unique identifier, but should allow to identify the
commit using the main version number even for very large code bases. </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=8)]
```

```
            public byte[] os_custom_version;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=30)]
    public struct mavlink_landing_target_t
    {
        /// <summary> Timestamp (micros since boot or Unix epoch) </summary>
        public  UInt64 time_usec;
            /// <summary> X-axis angular offset (in radians) of the target from
the center of the image </summary>
        public  Single angle_x;
            /// <summary> Y-axis angular offset (in radians) of the target from
the center of the image </summary>
        public  Single angle_y;
            /// <summary> Distance to the target from the vehicle in meters
</summary>
        public  Single distance;
            /// <summary> Size in radians of target along x-axis </summary>
        public  Single size_x;
            /// <summary> Size in radians of target along y-axis </summary>
        public  Single size_y;
            /// <summary> The ID of the target if multiple targets are present
</summary>
        public  byte target_num;
            /// <summary> MAV_FRAME enum specifying the whether the following
feilds are earth-frame, body-frame, etc. </summary>
        public  byte frame;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=32)]
    public struct mavlink_vibration_t
    {
        /// <summary> Timestamp (micros since boot or Unix epoch) </summary>
        public  UInt64 time_usec;
            /// <summary> Vibration levels on X-axis </summary>
        public  Single vibration_x;
            /// <summary> Vibration levels on Y-axis </summary>
        public  Single vibration_y;
            /// <summary> Vibration levels on Z-axis </summary>
        public  Single vibration_z;
            /// <summary> first accelerometer clipping count </summary>
        public  UInt32 clipping_0;
            /// <summary> second accelerometer clipping count </summary>
        public  UInt32 clipping_1;
            /// <summary> third accelerometer clipping count </summary>
        public  UInt32 clipping_2;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=52)]
    public struct mavlink_home_position_t
    {
        /// <summary> Latitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 latitude;
            /// <summary> Longitude (WGS84, in degrees * 1E7 </summary>
        public  Int32 longitude;
```

```
            /// <summary> Altitude (AMSL), in meters * 1000 (positive for up)
</summary>
        public  Int32 altitude;
            /// <summary> Local X position of this position in the local
coordinate frame </summary>
        public  Single x;
            /// <summary> Local Y position of this position in the local
coordinate frame </summary>
        public  Single y;
            /// <summary> Local Z position of this position in the local
coordinate frame </summary>
        public  Single z;
            /// <summary> World to surface normal and heading transformation of
the takeoff position. Used to indicate the heading and slope of the ground
</summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=4)]
             public float q;
            /// <summary> Local X position of the end of the approach vector.
Multicopters should set this position based on their takeoff path. Grass-landing
fixed wing aircraft should set it the same way as multicopters. Runway-landing
fixed wing aircraft should set it to the opposite direction of the takeoff,
assuming the takeoff happened from the threshold / touchdown zone. </summary>
        public  Single approach_x;
            /// <summary> Local Y position of the end of the approach vector.
Multicopters should set this position based on their takeoff path. Grass-landing
fixed wing aircraft should set it the same way as multicopters. Runway-landing
fixed wing aircraft should set it to the opposite direction of the takeoff,
assuming the takeoff happened from the threshold / touchdown zone. </summary>
        public  Single approach_y;
            /// <summary> Local Z position of the end of the approach vector.
Multicopters should set this position based on their takeoff path. Grass-landing
fixed wing aircraft should set it the same way as multicopters. Runway-landing
fixed wing aircraft should set it to the opposite direction of the takeoff,
assuming the takeoff happened from the threshold / touchdown zone. </summary>
        public  Single approach_z;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=53)]
    public struct mavlink_set_home_position_t
    {
        /// <summary> Latitude (WGS84), in degrees * 1E7 </summary>
        public  Int32 latitude;
            /// <summary> Longitude (WGS84, in degrees * 1E7 </summary>
        public  Int32 longitude;
            /// <summary> Altitude (AMSL), in meters * 1000 (positive for up)
</summary>
        public  Int32 altitude;
            /// <summary> Local X position of this position in the local
coordinate frame </summary>
        public  Single x;
            /// <summary> Local Y position of this position in the local
coordinate frame </summary>
        public  Single y;
            /// <summary> Local Z position of this position in the local
coordinate frame </summary>
        public  Single z;
            /// <summary> World to surface normal and heading transformation of
the takeoff position. Used to indicate the heading and slope of the ground
</summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=4)]
```

```
            public float q;
           /// <summary> Local X position of the end of the approach vector.
Multicopters should set this position based on their takeoff path. Grass-landing
fixed wing aircraft should set it the same way as multicopters. Runway-landing
fixed wing aircraft should set it to the opposite direction of the takeoff,
assuming the takeoff happened from the threshold / touchdown zone. </summary>
        public  Single approach_x;
            /// <summary> Local Y position of the end of the approach vector.
Multicopters should set this position based on their takeoff path. Grass-landing
fixed wing aircraft should set it the same way as multicopters. Runway-landing
fixed wing aircraft should set it to the opposite direction of the takeoff,
assuming the takeoff happened from the threshold / touchdown zone. </summary>
        public  Single approach_y;
            /// <summary> Local Z position of the end of the approach vector.
Multicopters should set this position based on their takeoff path. Grass-landing
fixed wing aircraft should set it the same way as multicopters. Runway-landing
fixed wing aircraft should set it to the opposite direction of the takeoff,
assuming the takeoff happened from the threshold / touchdown zone. </summary>
        public  Single approach_z;
            /// <summary> System ID. </summary>
        public  byte target_system;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=38)]
    public struct mavlink_adsb_vehicle_t
    {
        /// <summary> ICAO address </summary>
        public  UInt32 ICAO_address;
            /// <summary> Latitude, expressed as degrees * 1E7 </summary>
        public  Int32 lat;
            /// <summary> Longitude, expressed as degrees * 1E7 </summary>
        public  Int32 lon;
            /// <summary> Altitude(ASL) in millimeters </summary>
        public  Int32 altitude;
            /// <summary> Course over ground in centidegrees </summary>
        public  UInt16 heading;
            /// <summary> The horizontal velocity in centimeters/second
</summary>
        public  UInt16 hor_velocity;
            /// <summary> The vertical velocity in centimeters/second, positive
is up </summary>
        public  UInt16 ver_velocity;
            /// <summary> Flags to indicate various statuses including valid data
fields </summary>
        public  UInt16 flags;
            /// <summary> Squawk code </summary>
        public  UInt16 squawk;
            /// <summary> Type from ADSB_ALTITUDE_TYPE enum </summary>
        public  byte altitude_type;
            /// <summary> The callsign, 8+null </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=9)]
              public byte[] callsign;
            /// <summary> Type from ADSB_EMITTER_TYPE enum </summary>
        public  byte emitter_type;
            /// <summary> Time since last communication in seconds </summary>
        public  byte tslc;

    };
```

```
    [StructLayout(LayoutKind.Sequential,Pack=1,Size=254)]
    public struct mavlink_v2_extension_t
    {
        /// <summary> A code that identifies the software component that
understands this message (analogous to usb device classes or mime type strings).
If this code is less than 32768, it is considered a 'registered' protocol
extension and the corresponding entry should be added to
https://github.com/mavlink/mavlink/extension-message-ids.xml.  Software creators
can register blocks of message IDs as needed (useful for GCS specific metadata,
etc...). Message_types greater than 32767 are considered local experiments and
should not be checked in to any widely distributed codebase. </summary>
        public  UInt16 message_type;
            /// <summary> Network ID (0 for broadcast) </summary>
        public  byte target_network;
            /// <summary> System ID (0 for broadcast) </summary>
        public  byte target_system;
            /// <summary> Component ID (0 for broadcast) </summary>
        public  byte target_component;
            /// <summary> Variable length payload. The length is defined by the
remaining message length when subtracting the header and other fields.  The
entire content of this block is opaque unless you understand any the encoding
message_type.  The particular encoding used can be extension specific and might
not always be documented as part of the mavlink specification. </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=249)]
             public byte[] payload;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=36)]
    public struct mavlink_memory_vect_t
    {
        /// <summary> Starting address of the debug variables </summary>
        public  UInt16 address;
            /// <summary> Version code of the type variable. 0=unknown, type
ignored and assumed int16_t. 1=as below </summary>
        public  byte ver;
            /// <summary> Type code of the memory variables. for ver = 1: 0=16 x
int16_t, 1=16 x uint16_t, 2=16 x Q15, 3=16 x 1Q14 </summary>
        public  byte type;
            /// <summary> Memory contents at specified address </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=32)]
             public byte[] value;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=30)]
    public struct mavlink_debug_vect_t
    {
        /// <summary> Timestamp </summary>
        public  UInt64 time_usec;
            /// <summary> x </summary>
        public  Single x;
            /// <summary> y </summary>
        public  Single y;
            /// <summary> z </summary>
        public  Single z;
            /// <summary> Name </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=10)]
             public byte[] name;
```

```
    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=18)]
    public struct mavlink_named_value_float_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Floating point value </summary>
        public  Single value;
            /// <summary> Name of the debug variable </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=10)]
              public byte[] name;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=18)]
    public struct mavlink_named_value_int_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> Signed integer value </summary>
        public  Int32 value;
            /// <summary> Name of the debug variable </summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=10)]
              public byte[] name;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=51)]
    public struct mavlink_statustext_t
    {
        /// <summary> Severity of status. Relies on the definitions within RFC-
5424. See enum MAV_SEVERITY. </summary>
        public  byte severity;
            /// <summary> Status text message, without null termination character
</summary>
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=50)]
              public byte[] text;

    };


    [StructLayout(LayoutKind.Sequential,Pack=1,Size=9)]
    public struct mavlink_debug_t
    {
        /// <summary> Timestamp (milliseconds since system boot) </summary>
        public  UInt32 time_boot_ms;
            /// <summary> DEBUG value </summary>
        public  Single value;
            /// <summary> index of debug variable </summary>
        public  byte ind;

    }
```

```
------------------------------Sub Classe MAVLink CRC-----------------------------

public class MavlinkCRC
    {
        const int X25_INIT_CRC = 0xffff;
        const int X25_VALIDATE_CRC = 0xf0b8;

        public static ushort crc_accumulate(byte b, ushort crc)
        {
            unchecked
            {
                byte ch = (byte)(b ^ (byte)(crc & 0x00ff));
                ch = (byte)(ch ^ (ch << 4));
                return (ushort)((crc >> 8) ^ (ch << 8) ^ (ch << 3) ^ (ch >> 4));
            }
        }

        public static ushort crc_calculate(byte[] pBuffer, int length)
        {
            if (length < 1)
            {
                return 0xffff;
            }
            // For a "message" of length bytes contained in the unsigned char
            array
            // pointed to by pBuffer, calculate the CRC
            // crcCalculate(unsigned char* pBuffer, int length, unsigned short*
            checkConst) < not needed

            ushort crcTmp;
            int i;

            crcTmp = X25_INIT_CRC;

            for (i = 1; i < length; i++) // skips header
            {
                crcTmp = crc_accumulate(pBuffer[i], crcTmp);
                //Console.WriteLine(crcTmp + " " + pBuffer[i] + " " + length);
            }

            return (crcTmp);
        }

    }
}

}
-----------------------------------Sub Classe MAVLink PARAM--------------------

public class MAVLinkParam
    {
        /// <summary>
        /// Paramater name
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Value of paramter as a double
        /// </summary>

        public double Value
        {
            get
```

```
    {
        return GetValue();
    }
    set
    {
        SetValue(value);
    }
}
/// <summary>
/// Over the wire storage format
/// </summary>
public MAV_PARAM_TYPE Type { get; set; }

public byte uint8_value { get { return data[0]; } }
public sbyte int8_value { get { return (sbyte)data[0]; } }
        public ushort uint16_value { get { return
BitConverter.ToUInt16(data, 0); } }
        public short int16_value { get { return
BitConverter.ToInt16(data, 0); } }
        public UInt32 uint32_value { get { return
BitConverter.ToUInt32(data, 0); } }
        public Int32 int32_value { get { return
BitConverter.ToInt32(data, 0); } }
        public float float_value { get { return
BitConverter.ToSingle(data, 0); } }

internal byte[] data = new byte[4];

public MAVLinkParam(string name, double value, MAV_PARAM_TYPE type)
{
    Name = name;
    Type = type;
    Value = value;
}

public MAVLinkParam(string name, float inputwire, MAV_PARAM_TYPE type)
{
    Name = name;
    Type = type;
    data = BitConverter.GetBytes(inputwire);
    Array.Resize(ref data, 4);
}

public MAVLinkParam(string name, byte[] inputwire, MAV_PARAM_TYPE type)
{
    Name = name;
    Type = type;
    data = inputwire;
    Array.Resize(ref data, 4);
}

public double GetValue()
{
    switch (Type)
    {
        case MAV_PARAM_TYPE.UINT8:
            return (double)uint8_value;
        case MAV_PARAM_TYPE.INT8:
            return (double)int8_value;
        case MAV_PARAM_TYPE.UINT16:
            return (double)uint16_value;
        case MAV_PARAM_TYPE.INT16:
```

```
            return (double)int16_value;
        case MAV_PARAM_TYPE.UINT32:
            return (double)uint32_value;
        case MAV_PARAM_TYPE.INT32:
            return (double)int32_value;
        case MAV_PARAM_TYPE.REAL32:
            return (double)float_value;
    }

    throw new FormatException("invalid type");
}

public void SetValue(double input)
{
    switch (Type)
    {
        case MAV_PARAM_TYPE.UINT8:
            data = BitConverter.GetBytes((byte)input);
            Array.Resize(ref data, 4);
            break;
        case MAV_PARAM_TYPE.INT8:
            data = BitConverter.GetBytes((sbyte)input);
            Array.Resize(ref data, 4);
            break;
        case MAV_PARAM_TYPE.UINT16:
            data = BitConverter.GetBytes((ushort)input);
            Array.Resize(ref data, 4);
            break;
        case MAV_PARAM_TYPE.INT16:
            data = BitConverter.GetBytes((short)input);
            Array.Resize(ref data, 4);
            break;
        case MAV_PARAM_TYPE.UINT32:
            data = BitConverter.GetBytes((UInt32)input);
            break;
        case MAV_PARAM_TYPE.INT32:
            data = BitConverter.GetBytes((Int32)input);
            break;
        case MAV_PARAM_TYPE.REAL32:
            data = BitConverter.GetBytes((float)input);
            break;
    }
}

public static explicit operator byte(MAVLinkParam v)
{
    return (byte)v.Value;
}

public static explicit operator sbyte(MAVLinkParam v)
{
    return (sbyte)v.Value;
}

public static explicit operator short(MAVLinkParam v)
{
    return (short)v.Value;
}

public static explicit operator ushort(MAVLinkParam v)
{
    return (ushort)v.Value;
```

```
        }

        public static explicit operator int(MAVLinkParam v)
        {
            return (int)v.Value;
        }

        public static explicit operator uint(MAVLinkParam v)
        {
            return (uint)v.Value;
        }

        public static explicit operator float (MAVLinkParam v)
        {
            return (float)v.Value;
        }

        public override string ToString()
        {
            return Value.ToString();
        }
    }
}
```

----------Sub Classe MAVLink PARAMList----------

```
public class MAVLinkParamList: List<MAVLinkParam>
    {
        public MAVLinkParam this[string name]
        {
            get
            {
                foreach (var item in this)
                {
                    if (item.Name == name)
                        return item;
                }

                return null;
            }

            set
            {
                int index = 0;
                foreach (var item in this)
                {
                    if (item.Name == name)
                    {
                        this[index] = value;
                        return;
                    }

                    index++;
                }

                this.Add(value);
            }
        }

        public IEnumerable<string> Keys
        {
```

```
            get
            {
                foreach (MAVLinkParam item in this)
                {
                    yield return item.Name;
                }
            }
        }

        public bool ContainsKey(string v)
        {
            foreach (MAVLinkParam item in this)
            {
                if (item.Name == v)
                    return true;
            }

            return false;
        }

        public static implicit operator Hashtable(MAVLinkParamList list)
        {
            Hashtable copy = new Hashtable();

            foreach (MAVLinkParam item in list)
            {
                copy[item.Name] = item.Value;
            }

            return copy;
        }
    }
}
```

---------Sub Classe MAVLinkParse----------

```
public class MavlinkParse
    {
        public int packetcount = 0;
        public byte[] buffer1;

        public int badCRC = 0;
        public int badLength = 0;

        public static void ReadWithTimeout(Stream BaseStream, byte[] buffer, int
        offset, int count)
        {
            int timeout = BaseStream.ReadTimeout;

            if (timeout == -1)
                timeout = 60000;

            DateTime to = DateTime.Now.AddMilliseconds(timeout);

            int toread = count;
            int pos = offset;

            while (true)
            {
                // read from stream
                int read = BaseStream.Read(buffer, pos, toread);
```

```
            // Console.WriteLine(buffer[pos]);
            // update counter
            toread -= read;
            pos += read;

            // reset timeout if we get data
            if (read > 0)
                to = DateTime.Now.AddMilliseconds(timeout);

            if (toread == 0)
                break;

            if (DateTime.Now > to)
            {
                throw new TimeoutException("Timeout waiting for data");
            }
            System.Threading.Thread.Sleep(1);
        }
    }

    public byte[] ReadPacket(Stream BaseStream)
    {
        byte[] buffer = new byte[270];

        int readcount = 0;

        while (readcount < 200)
        {
            // read STX byte
            ReadWithTimeout(BaseStream, buffer, 0, 1);

            if (buffer[0] == MAVLink.MAVLINK_STX)
                break;

            readcount++;
        }

        // read header
        ReadWithTimeout(BaseStream, buffer, 1, 5);

        // packet length
        int lengthtoread = buffer[1] + 6 + 2 - 2; // data + header + checksum
        - STX - length

        //read rest of packet
        ReadWithTimeout(BaseStream, buffer, 6, lengthtoread - 4);

        // check message length vs table
        if (buffer[1] != MAVLINK_MESSAGE_LENGTHS[buffer[5]])
        {
            badLength++;
            // bad or unknown packet
            return null;
        }

        // resize the packet to the correct length
        Array.Resize<byte>(ref buffer, lengthtoread + 2);

        // calc crc
        ushort crc = MavlinkCRC.crc_calculate(buffer, buffer.Length - 2);
```

```
            // calc extra bit of crc for mavlink 1.0
            if (buffer.Length > 5 && buffer[0] == 254)
            {
            crc = MavlinkCRC.crc_accumulate(MAVLINK_MESSAGE_CRCS[buffer[5]],
            crc);
            }

            // Console.WriteLine(buffer[buffer.Length - 2]);
            // check crc
            if (buffer.Length < 5 || buffer[buffer.Length - 1] != (crc >> 8) ||
                buffer[buffer.Length - 2] != (crc & 0xff))
            {

                badCRC++;
                // crc fail
                return null;
            }

            return buffer;
        }

        public object ReadPacketObj(Stream BaseStream)
        {
            buffer1 = ReadPacket(BaseStream);

            byte header = buffer1[0];
            byte length = buffer1[1];
            byte seq = buffer1[2];
            byte sysid = buffer1[3];
            byte compid = buffer1[4];
            byte messid = buffer1[5];

            // create the object specified by the packet type
            object data = Activator.CreateInstance(MAVLINK_MESSAGE_INFO[messid]);

            // fill in the data of the object
            MavlinkUtil.ByteArrayToStructure(buffer1, ref data, 6);

            return data;
        }

        public byte[] GenerateMAVLinkPacket(MAVLINK_MSG_ID messageType, object
        indata)
        {
            byte[] data;

            data = MavlinkUtil.StructureToByteArray(indata);

            byte[] packet = new byte[data.Length + 6 + 2];

            packet[0] = 254;
            packet[1] = (byte)data.Length;
            packet[2] = (byte)packetcount;

            packetcount++;

            packet[3] = 255; // this is always 255 - MYGCS
            packet[4] = (byte)MAV_COMPONENT.MAV_COMP_ID_ALL;
            packet[5] = (byte)messageType;


            int i = 6;
```

```
            foreach (byte b in data)
            {
                packet[i] = b;
                i++;
            }

            ushort checksum = MavlinkCRC.crc_calculate(packet, packet[1] + 6);

            checksum =
            MavlinkCRC.crc_accumulate(MAVLINK_MESSAGE_CRCS[(byte)messageType],
            checksum);

            byte ck_a = (byte)(checksum & 0xFF); ///< High byte
            byte ck_b = (byte)(checksum >> 8); ///< Low byte

            packet[i] = ck_a;
            i += 1;
            packet[i] = ck_b;
            i += 1;

            return packet;
        }
    }
}
```

----------Sub Classse MAVLinkUtil----------

```
public static class MavlinkUtil
{

    public static TMavlinkPacket ByteArrayToStructure<TMavlinkPacket>(this byte[]
bytearray,
        int startoffset = 6) where TMavlinkPacket : struct
    {
        return ReadUsingPointer<TMavlinkPacket>(bytearray,startoffset);
    }

    public static TMavlinkPacket
ByteArrayToStructureBigEndian<TMavlinkPacket>(this byte[] bytearray,
        int startoffset = 6) where TMavlinkPacket : struct
    {
        object newPacket = new TMavlinkPacket();
        ByteArrayToStructureEndian(bytearray, ref newPacket, startoffset);
        return (TMavlinkPacket)newPacket;
    }

    public static void ByteArrayToStructure(byte[] bytearray, ref object obj, int
startoffset)
    {
        int len = Marshal.SizeOf(obj);

        IntPtr i = Marshal.AllocHGlobal(len);

        try
        {
            // copy byte array to ptr
            Marshal.Copy(bytearray, startoffset, i, len);
        }
        catch (Exception ex)
        {
            Console.WriteLine("ByteArrayToStructure FAIL " + ex.Message);
```

```
        }

        obj = Marshal.PtrToStructure(i, obj.GetType());

        Marshal.FreeHGlobal(i);
    }

    public static TMavlinkPacket ByteArrayToStructureT<TMavlinkPacket>(byte[]
bytearray, int startoffset)
    {
        int len = bytearray.Length - startoffset;

        IntPtr i = Marshal.AllocHGlobal(len);

        try
        {
            // copy byte array to ptr
            Marshal.Copy(bytearray, startoffset, i, len);
        }
        catch (Exception ex)
        {
            Console.WriteLine("ByteArrayToStructure FAIL " + ex.Message);
        }

        var obj = Marshal.PtrToStructure(i, typeof(TMavlinkPacket));

        Marshal.FreeHGlobal(i);

        return (TMavlinkPacket)obj;
    }

    public static T ReadUsingPointer<T>(byte[] data, int startoffset) where T :
struct
    {
        unsafe
        {
            fixed (byte* p = &data[startoffset])
            {
                return (T)Marshal.PtrToStructure(new IntPtr(p), typeof(T));
            }
        }
    }

    public static T ByteArrayToStructureGC<T>(byte[] bytearray, int startoffset)
where T : struct
    {
        GCHandle gch = GCHandle.Alloc(bytearray, GCHandleType.Pinned);
        try
        {
            return (T)Marshal.PtrToStructure(new
IntPtr(gch.AddrOfPinnedObject().ToInt64()
                + startoffset) , typeof(T));
        }
        finally
        {
            gch.Free();
        }
    }

    public static void ByteArrayToStructureEndian(byte[] bytearray, ref object
obj, int startoffset)
    {
```

```
        int len = Marshal.SizeOf(obj);
        IntPtr i = Marshal.AllocHGlobal(len);
        byte[] temparray = (byte[])bytearray.Clone();

        // create structure from ptr
        obj = Marshal.PtrToStructure(i, obj.GetType());

        // do endian swap
        object thisBoxed = obj;
        Type test = thisBoxed.GetType();

        int reversestartoffset = startoffset;

        // Enumerate each structure field using reflection.
        foreach (var field in test.GetFields())
        {
            // field.Name has the field's name.
            object fieldValue = field.GetValue(thisBoxed); // Get value

            // Get the TypeCode enumeration. Multiple types get mapped to a
            common typecode.
            TypeCode typeCode = Type.GetTypeCode(fieldValue.GetType());

            if (typeCode != TypeCode.Object)
            {
              Array.Reverse(temparray, reversestartoffset,
              Marshal.SizeOf(fieldValue));
                reversestartoffset += Marshal.SizeOf(fieldValue);
            }
            else
            {
                reversestartoffset += ((byte[])fieldValue).Length;
            }

        }

        try
        {
            // copy byte array to ptr
            Marshal.Copy(temparray, startoffset, i, len);
        }
        catch (Exception ex)
        {
            Console.WriteLine("ByteArrayToStructure FAIL" + ex.ToString());
        }

        obj = Marshal.PtrToStructure(i, obj.GetType());

        Marshal.FreeHGlobal(i);

    }

    /// <summary>
/// Convert a struct to an array of bytes, struct fields being reperesented in
little endian (LSB first)

    public static byte[] StructureToByteArray(object obj)
    {
        int len = Marshal.SizeOf(obj);
        byte[] arr = new byte[len];
        IntPtr ptr = Marshal.AllocHGlobal(len);
```

```
        Marshal.StructureToPtr(obj, ptr, true);
        Marshal.Copy(ptr, arr, 0, len);
        Marshal.FreeHGlobal(ptr);
        return arr;
    }

    /// <summary>
    /// Convert a struct to an array of bytes, struct fields being reperesented
in
    /// big endian (MSB first)
    /// </summary>
    public static byte[] StructureToByteArrayBigEndian(params object[] list)
    {
        // The copy is made because SetValue won't work on a struct.
        // Boxing was used because SetValue works on classes/objects.

        object thisBoxed = list[0]; // Why make a copy?
        Type test = thisBoxed.GetType();

        int offset = 0;
        byte[] data = new byte[Marshal.SizeOf(thisBoxed)];

        object fieldValue;
        TypeCode typeCode;

        byte[] temp;

        // Enumerate each structure field using reflection.
        foreach (var field in test.GetFields())
        {
            // field.Name has the field's name.

            fieldValue = field.GetValue(thisBoxed); // Get value

            // Get the TypeCode enumeration. Multiple types get mapped to a
            common typecode.
            typeCode = Type.GetTypeCode(fieldValue.GetType());

            switch (typeCode)
            {
                case TypeCode.Single: // float
                    {
                        temp = BitConverter.GetBytes((Single)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(Single));
                        break;
                    }
                case TypeCode.Int32:
                    {
                        temp = BitConverter.GetBytes((Int32)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(Int32));
                        break;
                    }
                case TypeCode.UInt32:
                    {
                        temp = BitConverter.GetBytes((UInt32)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(UInt32));
                        break;
                    }
                case TypeCode.Int16:
```

```
                    {
                        temp = BitConverter.GetBytes((Int16)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(Int16));
                        break;
                    }
                case TypeCode.UInt16:
                    {
                        temp = BitConverter.GetBytes((UInt16)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(UInt16));
                        break;
                    }
                case TypeCode.Int64:
                    {
                        temp = BitConverter.GetBytes((Int64)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(Int64));
                        break;
                    }
                case TypeCode.UInt64:
                    {
                        temp = BitConverter.GetBytes((UInt64)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(UInt64));
                        break;
                    }
                case TypeCode.Double:
                    {
                        temp = BitConverter.GetBytes((Double)fieldValue);
                        Array.Reverse(temp);
                        Array.Copy(temp, 0, data, offset, sizeof(Double));
                        break;
                    }
                case TypeCode.Byte:
                    {
                        data[offset] = (Byte)fieldValue;
                        break;
                    }
                default:
                    {
                        break;
                    }
            }; // switch
            if (typeCode == TypeCode.Object)
            {
                int length = ((byte[])fieldValue).Length;
                Array.Copy(((byte[])fieldValue), 0, data, offset, length);
                offset += length;
            }
            else
            {
                offset += Marshal.SizeOf(fieldValue);
            }
        } // foreach

        return data;
    } // Swap
}
```

```
-------------------------Classe STANAG 4586-------------------------------------


class STANAG_4586
    {
        public byte delay_stanag=0;
        public byte[] presence_vector= new byte[3];
        public byte[] msg_stanag = new byte[39];
        public byte length1 = 0;
        public byte length2 = 0;
        public byte sourceid1 = 0;
        public byte sourceid2 = 0;
        public byte sourceid3 = 0;
        public byte sourceid4 = 0;
        public byte destid1 = 0;
        public byte destid2 = 0;
        public byte destid3 = 0;
        public byte destid4 = 0;
        public byte msgtype1 = 0;
        public byte msgtype2 = 0;
        public byte msgprop1 = 0;
        public byte msgprop2 = 0;

        public byte[] GenerateMSG(string type,
            int initial_time,
            double Phi,
            double Theta,
            double Psi,
            double Phi_dot,
            double Theta_dot,
            double Psi_dot)
        {



            if (type == "#4000")
            {

                length1 = msg_stanag[0] = 0x00;   // message length
                length2 = msg_stanag[1] = 0x19;   // message length


                sourceid1 = msg_stanag[2] = 0x14;
                sourceid2 = msg_stanag[3] = 0x60;
                sourceid3 = msg_stanag[4] = 0x00;
                sourceid4 = msg_stanag[5] = 0x02;   // source id


                destid1 = msg_stanag[6] = 0x14;
                destid2 = msg_stanag[7] = 0x20;
                destid3 = msg_stanag[7] = 0x00;
                destid4 = msg_stanag[9] = 0x02;    // destination id

                msgtype1 = msg_stanag[10] = 0x0F;
                msgtype2 = msg_stanag[11] = 0xA0;  // message type

                msgprop1 = msg_stanag[12] = 0x1E;
                msgprop2 = msg_stanag[13] = 0x00;  // message properties
```

```
                msg_stanag[14] = presence_vector[0] = 0x01;
                msg_stanag[15] = presence_vector[1] = 0xF8;
                msg_stanag[16] = presence_vector[2] = 0x01;



                msg_stanag[17] = 0; // time stamp
                msg_stanag[18] = 0; // time stamp
                msg_stanag[19] = 0; // time stamp
                msg_stanag[20] = 0; // time stamp
                msg_stanag[21] = 0; // time stamp

                for (int n=22; n<=31;n++) // field 2-11
                {
                    msg_stanag[n] = 0;
                }


                msg_stanag[32] = Convert.ToByte(Phi);

                msg_stanag[33] = Convert.ToByte(Theta);

                msg_stanag[34] = Convert.ToByte(Psi);


                msg_stanag[35] = Convert.ToByte((Convert.ToInt32(Phi_dot)) * (5 *
                10 ^ (-4)));//yaw rate em 0.0005rad/s

                msg_stanag[36] = Convert.ToByte((Convert.ToInt32(Theta_dot)) * (5
                * 10 ^ (-4)));//pitch rate em 0.0005rad/s

                msg_stanag[37] = Convert.ToByte((Convert.ToInt32(Psi_dot)) * (5 *
                10 ^ (-4)));//roll rate em 0.0005rad/s

                msg_stanag[38] = 0; //Magnetic Variation

    }
                return msg_stanag;
        }
    }
}
```

Anexo A

ID’s das mensagens MAVLink

Mensagem	**ID (hexadecimal)**
Heartbeat	0
Attitude	1E
Mission Clear All	2D
Request Data Stream	42
Power Status	7D

Anexo B

Código de País

País	**Numero designado (decimal)**
Bélgica	002
Bulgária	003
Canada	004
Republica Checa	005
Dinamarca	006
Estónia	007
França	008
Alemanha	009
Grécia	010
Hungria	011
Islândia	012
Itália	013
Holanda	017
Noruega	018
Polonia	019
Portugal	020
Roménia	021
EUA	027
Israel	119

Anexo C

Processo GT-VENT

S. R.

MINISTÉRIO DA DEFESA NACIONAL
MARINHA

ESTADO-MAIOR DA ARMADA
Lisboa, 26 de Junho de 2015

Informação N.º .....

Parecer N.º .....

Proposta N.º 46 /DIVPLAN

Processo: 000.25.15

Assunto: GRUPO DE TRABALHO PARA OS VEÍCULOS NÃO TRIPULADOS – GT-VENT.

Referência: Despacho do Almirante CEMA Nº 06/15, de 12 de fevereiro.

Ao
Vice-almirante
Vice-Chefe do Estado-Maior da Armada

Nos termos do determinado no Despacho em referência, realizou-se a primeira sessão ordinária do GT-VENT em 08 junho p.p., no Estado-Maior da Armada, contando com a presença do EMA, CN, CINAV, SM; STI, DGAM, CGPM, e IH.

Na sequência dos trabalhos desenvolvidos e em cumprimento do determinado no parágrafo 5. do Despacho em referência, propõe-se a Vossa Excelência a aprovação dos seguintes documentos:

- Normas de funcionamento do GT-VENT (TOR);
- Objetivos para 2015 e calendarização.

À consideração superior de V.Exa.

O CHEFE DA DIVISÃO DE PLANEAMENTO,

Pedro Miguel de Sousa Costa
Capitão-de-mar-e-guerra

ESTADO-MAIOR DA ARMADA

Divisão de Planeamento

Normas de funcionamento do grupo de trabalho relativo ao desenvolvimento da capacidade de operação de veículos não tripulados na Marinha (GT-VENT)

Referência: Despacho do Almirante CEMA Nº 06/15, de 12 de fevereiro

1. ÂMBITO

O GT-VENT tem representadas várias áreas funcionais da Marinha, de modo a assegurar a articulação necessária à execução de um programa de desenvolvimento de veículos não tripulados na Marinha. A coordenação do GT-VENT é efetuada pelo Estado-Maior da Armada (EMA), Divisão de Planeamento.

2. INCUMBÊNCIAS

a. Ao GT-VENT incumbe propor / assegurar:

(1) A definição do enquadramento estratégico e as orientações relativas à atuação da Marinha em atividades relacionadas com veículos não tripulados;

(2) As linhas gerais orientadoras do desenvolvimento e operação de veículos não tripulados na Marinha;

(3) A definição dos elementos de doutrina (conceitos), organização (planeamento) e interoperabilidade (requisitos) dos veículos não tripulados na Marinha;

(4) O acompanhamento, no âmbito da Defesa Nacional, dos trabalhos que vierem a ser desenvolvidos nesta área;

(5) O desenvolvimento do conceito de emprego operacional e requisitos operacionais dos veículos não tripulados aéreos e submarinos na Marinha;

(6) A definição das linhas base para o conceito de emprego e requisitos operacionais de veículos não tripulados de superfície na Marinha, incluindo os terrestres;

(7) Coordenação das atividades de desenvolvimento e experimentação, autónomas ou no âmbito de protocolos assinados, sem prejuízo das competências próprias do Centro de Investigação Naval e do Instituto Hidrográfico;

(8) A criação ou extinção de Grupos de Projeto Integrados (GPI) associados à temática, bem como alterações de cariz genético, estrutural ou operacional relativas aos veículos não tripulados;

(9) A elaboração do plano anual de objetivos a alcançar e o calendário dos trabalhos a desenvolver.

b. As recomendações produzidas em sede do GT-VENT serão submetidas à aprovação do VALM VCEMA, sempre que necessário.

3. COMPOSIÇÃO

a. Na sua versão plenária, o GT-VENT é composto, para além do chefe da DIVPLAN/EMA, que o chefia, pelos seguintes representantes:

(1) Um representante do Estado-Maior da Armada;

(2) Um representante do Comando Naval;

(3) Um representante da Superintendência do Material;

(4) Um representante do Centro de Investigação Naval;

(5) Um representante do Instituto Hidrográfico;

(6) Um representante da Superintendência das Tecnologias de Informação.

b. Sempre que a razão dos trabalhos o justifique, o grupo pode ainda agregar representantes de outros organismos de Marinha ou entidades externas e constituir subgrupos de acordo com as matérias a ser abordadas. Desde já, identifica-se o interesse em contar com a participação da Direção Geral da Autoridade Marítima (DGAM) e Comando Geral da Polícia Marítima (CGPM). O

GT-VENT está organizado em quatro núcleos, na dependência do chefe do grupo, de acordo com o seguinte:

N1 – Direção e Coordenação; que dirige e coordena os trabalhos do grupo e desenvolve as matérias de natureza doutrinária. Este núcleo assegura as orientações relativas à ligação entre a Marinha e entidades externas no que respeita a esta temática e que os objetivos gerais da Marinha neste domínio são observados – EMA (Coordenador), CN, DGAM, CGPM, CINAV, IH, STI e SM.

N2 – Núcleo Operacional e de Planeamento; com a responsabilidade de planear e orientar o desenvolvimento dos trabalhos de natureza operacional e de preparar, coordenar, articular e acompanhar as ações de experimentação. Tem também a responsabilidade de fornecer os elementos de natureza operacional para elaboração das matérias doutrinárias – CN (Coordenador), DGAM (Coord. aspetos de natureza específica), CGPM (Coord. aspetos de natureza específica), CINAV, IH e EMA.

N3 – Núcleo de Sistemas; com a responsabilidade de identificar, integrar e articular, sob perspetiva técnica, o desenvolvimento dos diversos sistemas não tripulados em linha com os requisitos operacionais definidos – SM (Coordenador), DGAM, STI, IH, CINAV e EMA.

N4 – Núcleo de Investigação e Desenvolvimento, com a responsabilidade de contribuir com o conhecimento científico para o desenvolvimento tecnológico e para o apoio a todas as tarefas dos núcleos supracitados. – CINAV (Coordenador), IH, EMA, STI, CN e SM.

Caso seja necessário para uma sessão de trabalho, um núcleo pode solicitar a presença de outro representante. O representante do EMA presidirá sempre às ordens de trabalho, de acordo com as agendas de trabalho proposta pelo coordenador do núcleo.

4. FUNCIONAMENTO

a. O GT-VENT reúne ordinariamente em sessão plenária trimestral ou, extraordinariamente, se tal for solicitado por qualquer dos órgãos nele representados;

b. As sessões devem ser convocadas pelo coordenador, com uma antecedência mínima de 10 dias úteis, devendo a agenda final da reunião ser divulgada com uma antecedência mínima de 5 dias úteis;

c. Quando convocada uma sessão, as entidades participantes podem solicitar a inclusão de tópicos na agenda;

d. No final de cada sessão, ou *à posteriori*, através de correio eletrónico, será efetuada a respetiva súmula.

MARINHA

ESTADO-MAIOR DA ARMADA

GRUPO DE TRABALHO PARA OS VEÍCULOS NÃO TRIPULADOS (GT-VENT)

OBJETIVOS PARA 2015

1. Levantamento de todas as iniciativas de I&D em curso na Marinha;
2. Elaboração do Conceito de Emprego Operacional (CEO) dos veículos não tripulados;
3. Iniciar os trabalhos de identificação dos Requisitos Operacionais (RO) dos veículos não tripulados aéreos e de sub superficie;
4. Desenvolvimento de um programa de atividades no âmbito dos protocolos existentes relacionados com os veículos não tripulados;
5. Análise e enquadramento do quadro legal aplicável no que diz respeito a veículos não tripulados aéreos e elaboração de um guia geral sobre os procedimentos a adotar pela Marinha nesta matéria.

CALENDÁRIO PARA 2015

Até 30 de julho	Desenvolvimento o calendário de atividades planeadas de I&D, atividades planeadas de experimentação de UAV's e atividades de AUV's, com objetivos por atividade
Até Novembro 2015	Elaboração do Conceito de Emprego Operacional (CEO) dos veículos não tripulados.
Início dos trabalhos. Conclusão até março 2016	Elaboração dos Requisitos Operacionais (RO) dos veículos não tripulados aéreos e de sub superfície.