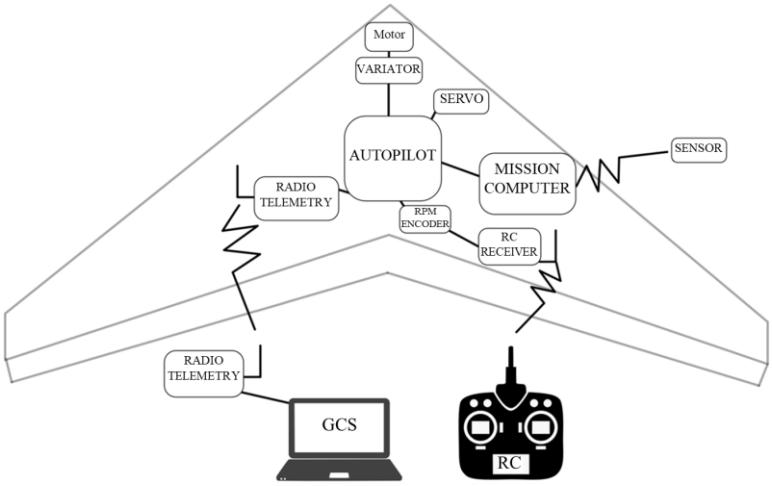
ACADEMIA DA FORÇA AÉREA



Software Architecture for Low-cost UAVs

An application considering automatic target tracking mission scenarios

João Filipe Gomes Moreira Alves

ASPAL/PILAV 140667-B

Dissertation to obtain the Master of Science Degree in

Military and Aeronautical Sciences – Aviator Pilot

Examination Committee

Chairperson:	MGEN/ENGEL 119923 -E Rui Fernando da Costa Ferreira
Supervisor:	MAJ/ENGEL 132274-F Tiago Miguel Monteiro de Oliveira
Co-Supervisors:	CAP/ENGEL 133004-H Gonçalo Charters Santos Cruz
	CAP/ENGEL 136787-A Diogo Alexandre Oliveira Silva
Member of the Committee:	Doctor Ricardo Adriano Ribeiro

Sintra, May 2022

The use of trademarks or names of manufacturers in this dissertation is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by neither the Portuguese Air Force Academy nor the author.

“A winner is a dreamer who never gives up”

Nelson Mandela

Acknowledgements/Agradecimentos

A realização da presente dissertação foi para mim um desafio que superou todas as espectativas inicialmente definidas. Tal trabalho seria inconcebível sem o apoio de um conjunto de pessoas que, direta ou indiretamente, abdicaram de tempo e contribuíram no desenvolvimento desta dissertação. A todos, um grande obrigado.

Agradeço à Academia da Força Aérea, instituição pela qual manifesto grande orgulho, a oportunidade e meios fornecidos para desenvolver este projeto. Todos os ensinamentos, a nível académico e pessoal, contribuíram para o resultado final deste projeto.

Agradeço ao Sr. Major Engenheiro Eletrotécnico Tiago Oliveira (orientador) pela incansável dedicação, disponibilidade, acompanhamento e apoio diário, na orientação para alcançar o objetivo final desta dissertação. Todos os conhecimentos transmitidos foram fulcrais para o seu sucesso.

Agradeço ao Sr. Capitão Engenheiro Eletrotécnico Gonçalo Cruz (coorientador), pelo apoio prestado durante o desenvolvimento da dissertação, pela disponibilidade e acompanhamento em variadas áreas, principalmente na área de Visão Computacional.

Agradeço ao Sr. Capitão Engenheiro Eletrotécnico Diogo Silva, pelo apoio prestado ao nível da orientação, organização e planeamento da componente de programação incluída neste trabalho.

Agradeço aos KAISERS, por todas as vivências e bons momentos proporcionados durante estes 5 anos. O apoio de todos vós, os desafios ultrapassados em conjunto e nossa união, foram motivo de querer ser melhor dia após dia. Deixo um especial agradecimento ao meu grande amigo e "Maria" Paulo Azevedo, por todos os momentos descontraídos, de convívio, e pelo espírito pacífico que transmites. Deixo também um especial agradecimento ao meu grande amigo João Almeida, por todas as conversas motivadoras e por quem me rejo como exemplo a seguir. Um obrigado por todos os esclarecimentos ao longo deste trabalho, mas também durante os últimos 5 anos.

Agradeço ao meu amigo Ricardo, pela excecional disponibilidade, humildade e ajuda prestada na resolução de problemas durante o desenvolvimento deste projeto.

A toda a minha família: ao meu Pai e Mãe, um enorme obrigado por todas as oportunidades, ensinamentos e conselhos. Se cheguei onde cheguei hoje é por vós, não por me terem definido este caminho, mas sim por terem sido os meus pilares de excelência, rigor, esforço, dedicação e felicidade. Às minhas irmãs, Catarina Alves e Margarida Alves, agradeço por toda ajuda na revisão gramatical e ortográfica e todo o apoio durante esta etapa.

Por fim, um agradecimento muito especial à Ana Rita Ramalho, por todo o amor, compreensão, carinho e por seres um pilar essencial na minha vida. Obrigado por seres uma das maiores motivações para atingir os meus objetivos.

Abstract

In recent years, advances in different technological areas enabled the development of low-cost, yet efficient Unmanned Air Vehicles (UAVs) for a broad spectrum of applications. The Air Force Academy Research Centre (CIAFA) is currently exploring new open-source software/hardware options, allowing for rapid prototyping and flight testing of control algorithms for UAVs.

This thesis aims the development of a software architecture based on the hardware developed by Silva et al. (2020), considering open-source software tools, available in the scientific community. Once the software architecture is outlined, an autonomous ground target tracking mission is considered, thus illustrating the effectiveness of the proposed solution.

In order to achieve the thesis' objectives, a literature review is carried out. First, a brief review of the hardware architecture from Silva et al. (2020) is presented, outlining the software constraints for each hardware component. Then, a review of the different available software components mostly used by the academic community is performed. Afterwards, the hardware/software architecture is proposed, detailing the software tools chosen to develop the CIAFA's software architecture. In particular, the proposed software architecture is comprised of five separate software modules: Control and Guidance, Target Estimate, Target Geolocation, Target Detector and Video Acquisition.

Each module is validated independently, followed by a closed-loop control system validation. This simulation aims to reproduce the target tracking mission proposed in this research. Results show that using the proposed architecture for the closed-loop control system, a fixed-wing UAV is able to autonomously detect, estimate and follow a ground target with good performance.

Keywords: Software architecture, Unmanned air systems, YOLOv3, Moving Path Following, PX4 autopilot, Gazebo.

Resumo

Nos últimos anos, os avanços em diferentes áreas tecnológicas permitiram o desenvolvimento de veículos não tripulados (UAVs) de baixo custo mas eficientes para um amplo espectro de aplicações. O Centro de Investigação da Academia da Força Aérea (CIAFA) atualmente explora novas opções de software e hardware de código-fonte aberta que permitam rápida prototipagem e testes de voo de UAVs.

Esta dissertação visa desenvolver uma arquitetura de software baseada no hardware desenvolvido por Silva et al. (2020), considerando as ferramentas de software de código-fonte aberta disponíveis na comunidade científica. Uma vez delineada a arquitetura de software, é considerada uma missão autónoma de seguimento de um alvo terrestre, ilustrando a eficácia da solução proposta.

A fim de alcançar os objetivos da tese, é efetuada uma revisão bibliográfica. Primeiro, realiza-se uma breve revisão da arquitetura de hardware de Silva et al. (2020), delineando os constrangimentos de software. Em seguida, é apresentada uma revisão dos diferentes componentes de software disponíveis e mais utilizados pela comunidade académica. Posteriormente, é proposta a arquitetura de hardware/software, detalhando as ferramentas de software escolhidas para desenvolver a arquitetura de software. A arquitetura proposta é composta por cinco módulos individuais: Control and Guidance, Target Estimate, Target Geolocation, Target Detector and Video Acquisition.

Primeiramente, cada módulo é validado de forma independente e seguidamente como simulação em circuito fechado. Esta simulação visa reproduzir a missão de vigilância proposta nesta dissertação. Os resultados mostram que a implementação da arquitetura proposta num sistema de controlo em circuito fechado com um UAV de asa-fixa, é capaz de detetar, localizar e seguir autonomamente um alvo, com bom desempenho.

Palavras-chave: *Arquitetura de Software, Sistemas Não Tripulados, YOLOv3, Moving Path Following, Piloto Automático PX4, Gazebo.*

Table of contents

1 Introduction 1

1.1 Background and Motivation 1
1.2 Main Objective 2
1.3 Problem Formulation 2
1.4 Dissertation Layout 2

2 Literature Review 5

2.1 Brief Review on the Hardware Architecture for Low-cost UAVs 5
2.2 Hardware Components 7

2.2.1 Autopilot 7
2.2.2 Mission Computer 7
2.2.3 Telemetry Radio 8
2.2.4 Sensors 8

2.3 Software 9

2.3.1 Software Architecture 9
2.3.2 Autopilot Flight Stack - PX4 and Ardupilot 10
2.3.3 Ground Control Station Software 11
2.3.4 Software Applications for Mission Computer 12
2.3.5 Middleware Interaction with Pixhawk 14
2.3.6 Communication through MAVLink 15
2.3.7 Simulator 15

2.4 Guidance and Control Techniques 15
2.5 Ground Vehicle Detection in Airborne Images using Computer Vision 18

2.5.1 Region-Based Conventional Neural Network (R-CNN) 19
2.5.2 Fast R-CNN 19
2.5.3 Faster R-CNN 19
2.5.4 YOLO 20
2.5.5 Single shot multi-box detector (SSD) 21
2.5.6 UAV Vehicle Detections Applications 22

3 System Architecture 25

3.1 Software Architecture 26
3.2 Gazebo Simulator 28
3.3 PX4 Autopilot 28
3.4 ROS Environment 29

3.4.1 Guidance and Control 30
3.4.2 Target Detector 40
3.4.3 Target Geolocation 41
3.4.4 Target Estimate 47

3.5 Closed-loop Validation - Intermediate Results 55

4 Ground Vehicle Detector 59

4.1 System Setup 60

4.1.1 Aircraft 60
4.1.2 Camera 60
4.1.3 Different Scenarios considered for Image Capture 61
4.2 Dataset 63
4.3 Detector Implementation 63
4.4 Neural Network Evaluation 65
4.4.1 Mean Average Precision Metrics 65
4.4.2 Mean Average Precision Computation 66
5 Closed-loop Validation – Final Results 69
5.1 Closed-loop Validation with the Object Standing at a Constant Position 69
5.2 Closed-loop Validation with the Object Moving 73
6 Conclusions and Recommendations 79
6.1 Conclusions 79
6.2 Future Work 81
7 References 83
Annexes 100

List of figures

Figure 1. Main components of a UAV system (Retrieved from Pastor et al., 2006). 5
Figure 2. Example of a system architecture (Adapted from Silva et al., 2020). 6
Figure 3. Options available for Flight Control Software and Ground Control Software (Retrieved from LambDrive, 2016). 9
Figure 4. Flight controller and Mission computer architecture adapted from PX4 Autopilot (Adapted from PX4 User Guide, 2021). 9
Figure 5. Hardware and Software architecture. 25
Figure 6. System Software architecture. 26
Figure 7. Flight stack architecture (Adapted from 'PX4 Architectural Overview', 2021). 29
Figure 8. Outer loop controller. 31
Figure 9. UAV climb to 100 m. 32
Figure 10. UAV pitch variation during climb to 100 m. 32
Figure 11. UAV airspeed variation during climb to 100 m. 33
Figure 12. UAV thrust during climb to 100 m. 33
Figure 13. UAV pitch variation during descent to 50 m. 34
Figure 14. UAV altitude during descent to 50 m. 34
Figure 15. UAV airspeed during descent to 50 m. 34
Figure 16. UAV thrust during descent to 50 m. 35
Figure 17. Moving-path following: relevant variables (Retrieved from Oliveira & Encarnação, 2013). 35
Figure 18. Path following lateral dynamics relevant variables (Retrieved from Oliveira & Encarnação, 2013). 36
Figure 19. Architecture for the simulation of MPF with fictitious target. 37
Figure 20. 2-dimensional display of the UAV and target during the simulation. 38
Figure 21. Lateral error. 39
Figure 22. Fictitious target Parameters. 39
Figure 23. Picture from the onboard camera with the object detection in colors and the binary mask indicating the position of the detected object. 41
Figure 24. Longitudinal perspective of the coordinate frame (Retrieved from Barber et al. (2006)). 42
Figure 25. Lateral perspective of the coordinate's frames (Retrieved from Barber et al. (2006)). 42
Figure 26. 2-dimensional display of the UAV and target during the simulation. 44
Figure 27. Front (a) and top (b) views of the camera's frame in relation to the aircraft. 44
Figure 28. Target position error. 44
Figure 29. 2-Dimensional view of the target's path. 45
Figure 30. 2-Dimensional view of the UAV, target estimated position and target's true position. 46

Figure 31. Comparison between the east and north position measured by the Geolocation algorithm and the actual position of the target. 47
Figure 32. Block diagram of the Kalman filtering algorithm (Retrieved from Lim et al. (2016)). 50
Figure 33. Architecture for the simulation of target measured by the Geolocation with Kalman Filter estimation 51
Figure 34. 2-dimensional display of the UAV, target measured by Kalman Filter, Geolocation and true location of the target during the simulation. 51
Figure 35. Target position estimated by Geolocation and Kalman Filter and true target position. 52
Figure 36. Target parameters estimated by Kalman Filter during the simulation. 52
Figure 37. 2-dimensional display of the UAV, target measured by Kalman Filter, Geolocation and true location of the target during the simulation. 53
Figure 38. East and north positions from Kalman Filter, Geolocation and true target position. 54
Figure 39. Comparison between the target velocity estimated by the Kalman Filter and true velocity. 54
Figure 40. Acceleration, angular acceleration and orientation from Kalman Filter estimate. 55
Figure 41. 2-Dimensional view of the target path. 56
Figure 42. 2-Dimensional view of the simulation. 57
Figure 43. Target position estimated by Kalman Filter and true target position. 57
Figure 44. Target velocity estimated by Kalman Filter and true target velocity. 58
Figure 45. Lateral error from UAV. 58
Figure 46. Standard Plane from PX4 60
Figure 47. Front (a) and side (b) views of the camera's orientation. This configuration was used to capture images from a vertical perspective. 61
Figure 48. Front (a) and top (b) views of the camera's orientation. This configuration was used to capture images from a side view and is the setup used for further simulations. 61
Figure 49. Example images of the four scenarios used for the dataset development. Image (a) corresponds to the Runway scenario, image (b) to the Mcmillan Airfield, image (c) to the Ground Plane scenario and image (d) to KSQL Airport. Images (a). (b) and (c) show two ground vehicles and image (c) displays six ground vehicles. 62
Figure 50. Yellow Beetle car from Gazebo. 62
Figure 51. Average loss function regarding the trainning of the neural network. 64
Figure 52. Variation of mAP as a function of IoU threshold. 67
Figure 53. Images with the detection bounding box in blue and the ground truth bounding box in green. Image (a) corresponds to a threshold of 60%, (b) to a threshold of 50%, (c) to a threshold of 40% and (d) to a threshold of 30% 68
Figure 54. 2-Dimensional view of the simulation. 70
Figure 55. Position estimate and error from Kalman Filter and true target position. 71
Figure 56. East and north axis velocity, acceleration, angular acceleration and orientation estimated by the Kalman Filter. 71
Figure 57. Lateral error during simulation. 72

Figure 58. 2-Dimensional view of the target path. 74
Figure 59. 2-Dimensional view of the UAV, target estimated position and target's true position. 75
Figure 60. Position estimate and error from Kalman Filter and true target position. 75
Figure 61. Velocity estimated from Kalman Filter and true velocity of the target. 76
Figure 62. Lateral error during simulation. 77
Figure B-1. PX4 Flight Stack High-Level Software Architecture (Retrieved from PX4 User Guide, 2021d). B-1
Figure I- 1. Example of four images used in the test set (marked with 1) and the corresponding detection images (marked with 2) from the four different worlds used in our dataset. Images (a) and (b) have a resolution of 920x920 pixels, and images (c) and (d) have a resolution of 640x640 pixels. I-1
Figure H- 1. Architecture diagram of YOLOv3 (Retrieved from Cheng et al., 2020). H-1
Figure L- 1. YOLOv3 network architecture diagram (Retrieved from Kathuria, 2018). L-1

List of tables

Table 1. Ground Control Stations Options (Adapted from Sabikan & Nawawi, 2016).....12

Table 2. Detailed information regarding the fictitious target parameters and UAV lateral error.....40

Table 3. Detailed information regarding the Geolocation validation in four simulations with the target standing at a constant position.....45

Table 4. Results from the validation of Geolocation with the target moving.....47

Table 5. Parameters estimated by the Kalman Filter. 52

Table 6. Results from Kalman Filter position estimation.....53

Table 7. Results from Kalman filter parameters estimation.....55

Table 8. Information regarding Kalman Filter target position.....57

Table 9. Information about the lateral error from the UAV.....58

Table 10. Network evaluation results for different IoU Thresholds. 67

Table 11. Results from Kalman Filter estimation. 70

Table 12. Results regarding the MPF lateral error. 72

Table 13. Detection results. 73

Table 14. Results from the Kalman Filter position estimation. 75

Table 15. Results regarding the MPF lateral error. 77

Table 16. Detection results. 77

Table A-1. Comparison of the speed and accuracy of different object detectors on the MS-COCO (Retrieved from Long et al., 2020). A-1

Table A-2. Results on Pascal VOC2007 test (Retrieved from Liu et al., 2016). A-1

Table A-3. Comparison detection performance in terms of mean average precision (mAP) of several state-of-art detectors (Retrieved from Mandal et al., 2019). A-1

Table A-4. Computation and space complexity comparison of the SSDNet, YOLOv3 and YOLOv3-tiny with input layer size=608x608x3. The FPS is computed over a CPU system (Retrieved from Mandal et al., 2019). A-1

Table A-5. Comparison of SSD512, YOLOv3 and RetinaNet methods in the Dorrer et al. (2019) dataset (Retrieved from Dorrer et al., 2019). A-2

Table A-6. Comparison Detection Performance of ADVNet and several state-of-art detectors (Retrieved from Mandal et al., 2020). A-2

Table C-1. Detailed information regarding the PI controller validation in climb and descent scenarios..C-1

Table D- 1. Detailed information regarding the path from the target..... D-1

Table D- 2. Camera. UAV and target settings used in this simulation. D-1

Table E- 1. Camera. UAV and target settings used in this simulation.....E-1

Table F- 1. Camera. UAV and target settings used in the simulation. F-1

Table F- 2. Detailed information about the path of the target. F-2

Table G- 1. Camera and UAV settings used to capture the dataset images. Tests from 1 to 6 were conducted in the Ground plane scenario displayed in Figure 45(c). Tests 7 and 8 were performed in the Runway scenario depicted in Figure 50(a). G-1
Table J- 1. Parameters maintained during simulation from Camera, UAV and Target. J-1
Table K- 1. Detailed information about the path with the target moving. K-1
Table K- 2. Camera, UAV and target settings used in the simulation. K-2

List of Abbreviations

AdaGrad	Adaptative Gradient
Adam	Adaptative Moment Estimation Optimizer
AI	Artificial Intelligence
AP	Autopilot
APM	AutoPilot Mega
AVDNet	Vehicle Detection Network
BoF	Bag of Freebies
BoS	Bag of Specials
CIAFA	Air Force Academy Research Centre
COCO	Common Objects in Context
COWC	Cars Overhead With Context
CPU	Central Processing Unit
DCNN	Deep Convolutional Neural Network
DL	Deep Learning
DOTA	Dataset for Object Detection in Aerial Images
FPN	Feature Pyramid Network
FPS	Frames per second
GCS	Ground Control Station
GMM	Gaussian mixture model
GPS	Global Positioning System
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
IoU	Intersection over Union
LGVF	Lyapunov guidance vector field
LIDAR	Light Detection and Ranging LIDAR
mAP	Mean Average Precision
MAV	Micro Air Vehicle
MB	MegaByte
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MPF	Moving Path Following

NED	North-East-Down
NMS	Non-Maximuim Suppression
NN	Neural Networks
PID	Proportional-Integral-Derivative
PPM	Pulse Position Modulation
RC	Remote Control
R-CNN	Region-Based Convolutional Neural Network
RGB	Red, Green, and Blue
RHC	Receding Horizon Control
RMSProp	Root Mean Square Propagation
ROI	Region of Interest
ROS	Robot Operating System
RPN	Region Proposal Network
SILT	Software-In-The-Loop Testing
SSD	Single Shot Multi-Box Detector
SVM	Support Vector Machine
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicles
UGV	Unmanned Ground Vehicle
VAID	Vehicle Aerial Image Dataset
VEDAI	Detection in Aerial Imagery
VTOL	Vertical Take-Off and Landing
YOLO	You only look Once

1 Introduction

1.1 Background and Motivation

The increasing use of Unmanned Aerial Vehicles (UAVs) for military and civilian purposes determines the need to adopt low-cost yet efficient configurations to fulfil the mission for which they are being committed. In addition, the recent advances in different areas, such as communication systems, solid-state devices, and battery technology, enables low-cost and low-weight Unmanned Aerial Systems (UAS) to be an excellent solution for a broad spectrum of applications (Beard et al., 2005). Furthermore, low-cost UAVs are also an excellent option considering dangerous operations and repetitive situations where these systems are engaged (López et al., 2007).

Since 2009, the Portuguese Air Force Academy Research Centre (CIAFA) has been involved in several research projects (for example, PITVANT, PERSEUS, SUNNY, SEAGULL, FIREFRONT) using UAVs. Framed in the above-mentioned research projects, CIAFA's main research areas and corresponding publications include path following and target tracking for fixed-wing UAVs (Oliveira et al., 2013, 2016a, 2016b, 2017), computer vision (Marques et al., 2014; Bernardino & Cruz, 2015; Cruz & Bernardino, 2016), micro and nano flapping-wing aerial vehicles (Caetano et al., 2013; Caetano, Percin, et al., 2015; Caetano, Weehuizen, et al., 2015; Armanini et al., 2016), multidisciplinary design optimization (Felix et al., 2012, 2013), and reliability and airworthiness for small UAVs (Gonçalves et al., 2016, 2017).

Typically, Piccolo (Collins Aerospace, 2022) autopilots have been used by CIAFA for research purposes. However, Piccolo autopilots are relatively expensive and resort to closed-source software, making it very difficult to sustain and operate due to the need for specific skills for that purpose (Kortunov et al., 2015). Therefore, CIAFA is currently exploring new open-source hardware and software options that allow for rapid prototyping and flight testing of UAVs. One of the most used open-source autopilots is the Pixhawk (Kortunov et al., 2015). This autopilot has already been integrated into previous thesis, as in Silva & Oliveira (2017), who compared Pixhawk and Piccolo. Later, Ferreira et al. (2019), Silva et al. (2020) and Mendes et al. (2021) developed aerial platforms using this same hardware.

This thesis aims to implement a rapid prototyping and testing architecture, which will be used to develop trajectory controllers and computer vision algorithms using low-cost platforms.

1.2 Main Objective

In the scope of this thesis, the main goal is to develop and validate a software architecture compliant with the hardware architecture proposed by Silva et al. (2020) for later implementation in a fixed-wing UAV. The effectiveness of the proposed architecture is demonstrated by considering a target tracking mission, in which a UAV should detect and track a moving ground vehicle.

1.3 Problem Formulation

The first problem that this thesis intends to solve is to propose a software architecture suitable for prototyping and expediting the validation of flight controllers taking into account the hardware architecture proposed in Silva et al. (2020). This can be formulated as follows: i) consider the hardware architecture proposed in Silva et al. (2020), ii) propose a compatible software architecture for rapid prototyping of computer vision, guidance and control algorithms for low-cost, fixed-wing UAVs.

Once the previous problem is successfully addressed, the proposed software architecture will be validated by implementing a case study described in the following: i) consider the problem of a UAV performing an autonomous detection and tracking of a ground vehicle, ii) using the software architecture previously proposed, implement a modular control architecture that allows the UAV to:

1- Detect a ground vehicle.

2- Estimate its location based on the data collected using a Red, Green and Blue (RGB) camera.

3- Autonomously follow the ground vehicle based on that information.

1.4 Dissertation Layout

This dissertation is divided into six main chapters. Chapter 1 starts by presenting the background and motivations of this research based on the CIAFAs projects and future objectives, followed by the main objectives and problem formulation.

The second chapter displays a brief review of the hardware architecture and some notes of the software restrictions in each hardware component. Afterwards, the software commonly used and supported by the hardware architecture for data acquisition and processing is studied in depth. The last part of this review focuses on Computer Vision and Control and Guidance Techniques to solve the problem at hand.

Subsequently, in Chapter 3, the proposed hardware/software architecture and the software tools decision are presented. Subsequently, the software architecture is analysed and each block, as well as each connection, is explained in detail. Each block is validated individually and then as a closed-loop architecture. In these tests, the target is defined as a red block, and the detector is a colour filter.

Chapter 4 comprises the implementation of a YOLO detector and a brief description of its characteristics. Furthermore, the detector is tested and validated through an evaluation based on mean average precision.

The fifth chapter culminates in the closed-loop simulation, where the whole software architecture is tested to perform the detection and tracking of a ground vehicle. This chapter intends to verify if the architecture is valid to respond to the proposed surveillance mission.

Finally, the Chapter 6 provides an overview of the dissertation and summarizes the conclusions obtained. Limitations and recommendations from the several stages of the research are then exposed. Future studies perspectives that appeared during the execution of this study are then provided.

2 Literature Review

Over the years, UAVs have been used for numerous applications. Monitoring forest fires (Merino et al., 2015), volcanic phenomena (Melita et al., 2015), traffic monitoring (Kanistras et al., 2015), precision agriculture (Alsalam et al., 2017), cooperative UAVs for search, surveillance, and target tracking (Beard et al., 2006; Kingston et al., 2008; Sujit & Beard, 2008; Manathara et al., 2011) are only a few examples of many domains where UAVs can prove their worth. (R. Beard et al., 2005).

Since the engagement of UAVs has a wide range of applications, there has been a development in small and less expensive UAVs (Skulstad et al., 2015; Wang et al., 2016). Furthermore, even though UAV research is a field of long history, the availability of low-cost sensors and the evolution of computing power have provided modernized UAVs at a fraction of the original value (Klausen et al., 2017). Therefore, to conduct many missions, the loss of a UAV should not be detrimental to the owner.

2.1 Brief Review on the Hardware Architecture for Low-cost UAVs

A UAV architecture is somewhat complex due to its need to accommodate all the systems in a small volume, ensuring low weight, limited power resources, increased computational requirements, and never hindering the system's performance (Pastor et al., 2006).

Figure 1. Main components of a UAV system (Retrieved from Pastor et al., 2006).

According to Pastor et al. (2006), it is possible to state that a UAV system comprises six main sub-modules, as shown in Figure 1. The first is the UAV airframe, which corresponds to the aircraft's structure. The second is the flight computer, which is the heart of the UAV. It gathers all the information from a set of sensors and guides the aircraft through a flight path. The third system is the payload, for example, sensors. The fourth is the mission controller, which controls the operation of the sensors. The base station or Ground Control Station (GCS)

is the fifth module. It is located on the ground, which monitors the mission execution and ultimately allows for the operation of the UAV navigation systems and payload. Finally, the communication infrastructure is responsible for establishing a continuous link between the UAV and the GCS.

Since this thesis aims to develop software architecture based on the hardware architecture proposed by Silva et al. (2020), it imposed restrictions on the software and firmware running in the different UAS systems. As a result, to define the software architecture, some hardware restrictions must be considered.

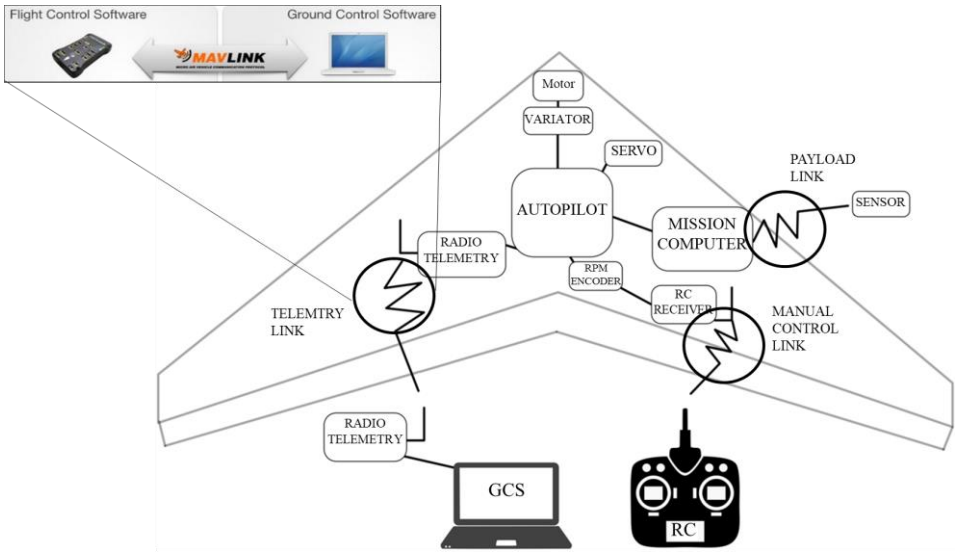


Figure 2. Example of a system architecture (Adapted from Silva et al., 2020).

Figure 2 represents a hardware architecture from Silva et al. (2020) research that displays the link to GCS, Autopilot, and Mission Computer. According to Silva et al. (2020), the telemetry link connection allows communication between the GCS and the aircraft's autopilot. This link is established using MAVLink (PX4 User Guide, 2022a), enabling the UAV system to exchange flight parameters (relative position, altitude, speed, and Global Positioning System (GPS location of the model) and energy parameters during flight. In addition, the GCS is able, through this connection, to send flight commands, for example, flight plans (Silva et al., 2020).

The manual control link is made through radio frequency between the remote control (RC) and the UAV, enabling the pilot to control the model manually through radio command. The information is sent directly by radio (RC IN), which sends that information to the autopilot channel by a Pulse Position Modulation (PPM). The autopilot will act accordingly when it receives this information (Silva et al., 2020).

The payload link connection contemplates the autopilot and the Flight computer. This link enables onboard sensor information processing, such as the sensors embedded in the autopilot or other UAV sensors, such as a RGB camera for image capture (Choi et al., 2016).

Once the hardware architecture has been briefly described, the explanation of the software options used for the Autopilot, Mission Computer, Telemetry Radio and Sensors will be performed, along with the constraints imposed at the software and firmware level.

2.2 Hardware Components

2.2.1 Autopilot

A UAV consists of specific parts, in which the flight control system, including autopilot and guidance devices, is one of the critical components (S. Wang et al., 2016). There are several options in the context of small and low-cost autopilots for UAVs, particularly Paparazzi, Kestrel, and Piccolo (Kortunov et al., 2015); however, Silva et al. (2020) proposed the use of Pixhawk (PX4 User Guide, n.d.-a).

Pixhawk is one of the leading open-source autopilot projects (Kortunov et al., 2015), which further develops Ardupilot and ArduPilot Mega (APM) projects (ArduPilot Dev Team, n.d.). This autopilot includes a GPS and two Inertial Measurement Unit (IMU) (Silva et al., 2020). In terms of specifications, Pixhawk is a small size, light-weighted, and low-cost Autopilot (Kortunov et al., 2015).

The main advantage of this autopilot is that the source code is available for all users, from beginners to professionals, which can adapt it for their use. Furthermore, the project evolves quickly due to many enthusiastic developers (Kortunov et al., 2015).

As for firmware restrictions, two major software packages are fully compatible with the Pixhawk autopilot, the Ardupilot and the PX4 (LambDrive, 2016a). The choice of the most suitable software flight stack for this autopilot will be addressed in this thesis.

2.2.2 Mission Computer

The choice of the mission computer must be based on the performance required by the system. On the one hand, the mission computer with less computing load can be powered by less powerful hardware (Pokhrel, 2018). On the other hand, if the system requires a high level of computation and real-time calculation, a powerful mission computer should be used (Pokhrel, 2018).

Given its availability in the CIAFA laboratory, two mission computer options can be considered for hardware architecture namely, the Raspberry Pi (Raspberry Pi Foundation, n.d.) and NVidia TX2 (NVidia Developer, 2022). Raspberry Pi is a computer widely used by the whole academic community due to its diversity of applications and accessories for the board and connectivity (Polo et al., 2015). In addition, this computer is low-cost, lightweight, low-

size, and capable of running a complete Linux distribution (Polo et al., 2015). The NVidia TX2 is a computational system that delivers great improvement in Artificial Intelligence (AI) computing performance. This device is compact, power-efficient, and ideal for several UAV applications (Xu et al., 2018). In addition, it is a module smaller than a credit card and lightweight, so it is suitable for small UAVs (Burdziakowski, 2018).

Both mission computers have no significant software restrictions since they can operate with various software solutions to communicate with the autopilot and process its data (ArduPilot Dev Team, 2021b). Moreover, these two choices are available in CIAFA. As a result, the choice between these two mission computers depends mainly on the intended application. In particular, Jetson TX2 provides higher Central Processing Unit (CPU) and Graphics Processing Pnit (GPU) computational power (Süzen et al., 2020).

2.2.3 Telemetry Radio

Typically, the hardware associated with a GCS is a laptop connected to a radio receiver with an antenna that allows sending and receiving data from the UAV. This component provides the establishment of the telemetry link presented in Figure 2.

Several radio telemetry systems are available to the community, for instance, RMILEC, DragonLink, EzUHF, and HawkEye Open LRS (Umair Iqbal et al., 2015). However, Silva et al. (2020) proposed the SIK telemetry radio (ArduPilot Dev Team, 2021f) because of its small size, lightweight, low-cost, and enables short-range communication (300m). Furthermore, an open-source firmware is embedded in this radio, explicitly designed to operate with MAVLink (ArduPilot Dev Team, 2021h). The SIK telemetry radio's firmware is compatible with different GCS software applications that allow for the graphical display of telemetry information (refer to Section 2.3.3).

2.2.4 Sensors

Different mission objectives can be satisfied by changing the payloads while using the same airframe. The selection of a UAV sensor should consider several factors, including the mission to be performed, available size, weight and power, required accuracy, etc. (Alvarado et al., 2015). Given the context of this thesis, it was considered that for (eventual) early laboratorial/flight validation of the architecture proposed in this work, the use of a RaspberryPi camera (Raspberry Pi Foundation, n.d.) rigidly attached to the UAV airframe is an adequate solution.

2.3 Software

Once all hardware components and their software constraints have been presented, the options available for the software architecture is addressed. Several software possibilities can be loaded onto the previously addressed hardware devices. For instance, in Figure 3, the Pixhawk autopilot can operate with PX4 or Ardupilot firmware, and the GCS has the QGroundControl (QGC) (QGroundControl, 2019), APM Planner 2 (APM Planner 2, 2021) and other software available.

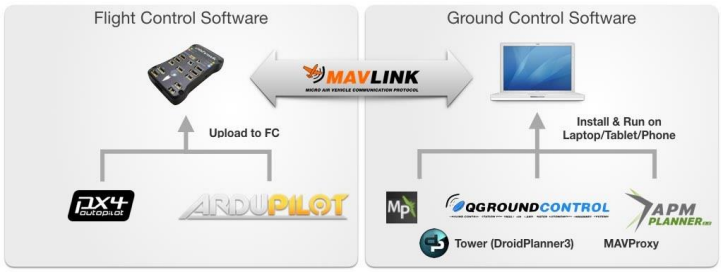


Figure 3. Options available for Flight Control Software and Ground Control Software (Retrieved from LambDrive, 2016).

This sub-section initially describes a common software architecture implemented together with the Pixhawk autopilot system. Afterwards, the two firmware options compatible with the Pixhawk autopilot are presented. Then a brief characterization of the options for the Ground Stations software and the Mission Computer to command and control is addressed. Finally, the interaction with Pixhawk as well as the MAVLink communication are described.

2.3.1 Software Architecture

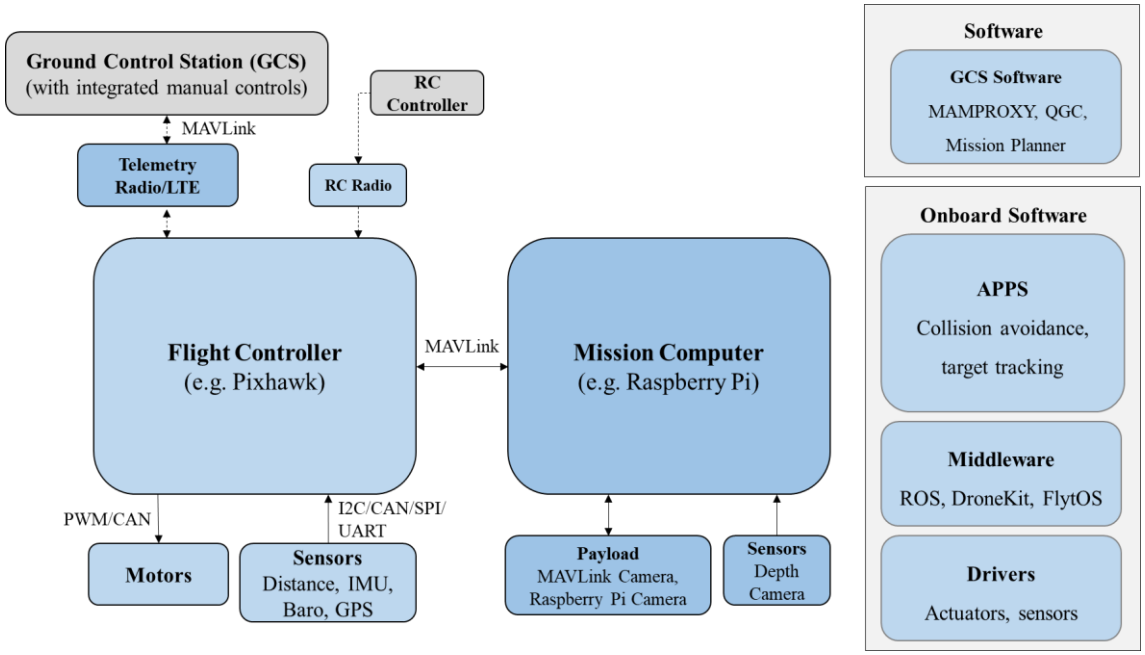


Figure 4. Flight controller and Mission computer architecture adapted from PX4 Autopilot (Adapted from PX4 User Guide, 2021).

Figure 4 shows a typical software stack that is compatible with the hardware architecture proposed by Silva et al. (2020). The different options available in the academic community for the autopilot firmware, the ground station software, and the interaction with the autopilot will be discussed in the following sections.

2.3.2 Autopilot Flight Stack - PX4 and Ardupilot

This sub-section covers the two firmware most used by the community in the Pixhawk autopilot: PX4 and Ardupilot. In the following paragraphs, the main differences between the two flight stacks and choose the most suitable option are presented.

Generally, Ardupilot and PX4 use MAVLink protocol to establish communications with vehicles, submit commands, and receive Telemetry and specific data (Torrico, 2020). Moreover, even though they jointly developed this hardware, the two software chose different approaches and open licensing codes. The better support and information from the Ardupilot community are some of the main differences between those firmware (Silva & Oliveira, 2017).

Ardupilot can be installed in open hardware, for instance, Beagle Bone Blue (Linux) and Holybro Pixhawk 4, as well as in closed-source hardware. However, some limitations regarding this firmware are related to some configurations exceeding 1 MegaByte (MB) size. In addition, some autopilots may not have enough flash memory to store the entire firmware. For this reason, in such cases, reduced firmware is generated by omitting less commonly used features (ArduPilot Dev Team, n.d.).

Depending on the type of vehicle, the Ardupilot firmware integrates several sensors. It uses gyroscopes, accelerometers, compass, altimeter, and GPS for primary flight control and stabilization. Apart from those sensors, it can also support additional sensors that allow enhanced functionality, for instance, sonar, laser, optical flow, and airspeed sensors. As for the sensors supported by the PX4, it can provide many peripherals, including IMU sensors, Light Detection and Ranging (LIDAR), range finders, and optical flow. In addition, it has flexible and powerful flight modes, which provide distinctive types/levels of vehicle automation and autopilot assistance to the user.

Both PX4 and Ardupilot have safety features that protect and recover the vehicle safely if something does not go as planned, such as low battery, data link loss, offboard loss, to name some. Furthermore, in both autopilots, the controller can insert specific areas and conditions where the vehicle can fly safely and program how to perform upon failsafe (PX4 User Guide, n.d.-a).

One of the main differences between Ardupilot and PX4 is its license. Ardupilot holds a GPLv3 license, while PX4 uses a BSD-3. The first license compels the user to copyleft the source code developed, which is less attractive to commercial enterprises that want to sell a UAV product. On the other hand, the BSD-3 license is a permissive license in which the users do not have to release their source code openly (Benowitz, 2021).

One of the key points of the PX4 over the Ardupilot is the availability of the Offboard mode for fixed-wing aircraft. The Ardupilot for fixed-wing aircraft provides the Guided mode. However, this mode only allows sending position commands, i.e., sending the fixed-wing aircraft to a certain waypoint, and when it reaches the predetermined coordinate, it starts a loiter (ArduPilot Dev Team, 2021c). Conversely, the PX4 provides the Offboard mode, which enables sending vehicle movement and attitude commands, yet supports only a limited set of MAVLink messages. In particular, the *Set_Attitude_Target* message allows sending attitude commands and *thrust setpoint* to the aircraft through a Mission Computer (PX4 User Guide, 2021a).

Silva & Oliveira (2017) concluded that the Ardupilot community is broader and more active compared to PX4. Consequently, there is more information sharing and progressing, which benefits from incorporating such software. Sørensen et al. (2017) also concluded that Ardupilot provided good support for the drone and its software ecosystem.

2.3.3 Ground Control Station Software

Although UAVs are already equipped with high autonomy, the contact with a ground station is always necessary (Erdos & Watkins, 2008). A GCS is generally a software application that runs in a ground-based computer and enables communication with a UAV through wireless Telemetry (ArduPilot Dev Team, 2021a).

The GCS simplifies the control and monitoring of the vehicle and the exploitation of information from the UAV (Natarajan, 2001). Moreover, a typical GCS provides a straightforward, intuitive user interface allowing the operator to insert waypoints on a map and then command the UAV to fly according to the created path (Erdos & Watkins, 2008). Lastly, Natarajan (2001) stated that the design of a GCS should address some requirements, for instance, air vehicle control, payload control, mission planning, payload data analysis and dissemination, post-flight analysis, etc. Therefore, there are several GCS software options that include different features, such as QGroundControl, APM Planner, MAVProxy (ArduPilot Dev Team, 2019) and Mission Planner (ArduPilot Dev Team, 2021e). Moreover, all these software

GCS (presented below) support the MAVLink communication protocol and are user-free (Sabikan & Nawawi, 2016).

MAVProxy is a fully functioning open-source GCS for UAVs, developed to be portable, small, and used by any autonomous vehicle (ArduPilot Dev Team, 2019). In addition, the Mission Planner is open-source full-featured software GCS, where the customer can configure the hardware and interact with the aircraft during flight. (Figueiredo, 2014). Conversely, the APM Planner 2 is an open-source GCS that provides a smaller user base and a reduced feature compared to the Mission Planner. This software offers configurations and calibrations to autopilots for autonomous vehicle control in particular missions (APM Planner 2, 2021). Finally, the QGroundControl is an open-source GCS that provides complete flight control and mission planning for any MAVLink enabled drone. From the GCS addressed in this research, it is the only GCS that can deliver flight control and configuration to multiple UAVs simultaneously (Ramirez-Atencia & Camacho, 2018) and also runs on all platforms, desktops and mobiles (QGroundControl, 2019).

Table 1 gathers the relevant information for the choice of software for the GCS.

Table 1. Ground Control Stations Options (Adapted from Sabikan & Nawawi, 2016).

GCS	Userbase	Supported OS	Compatibility to PX4	Compatibility with Ardupilot
MAVProxy	Advanced Users, developers	Win, Linux, OS X, Android	Partial	Yes
Mission Planner	Basic and Advanced Users, developers	Win only, runs on Linux using Mono and OS X using VM	Partial	Yes
QGroundControl	Basic and Advanced Users, developers	Win, Mac, Linux	Yes	Partial
APM Planner 2	Basic and Advanced Users, developers	Win, Linux, OS X	Partial	Yes

2.3.4 Software Applications for Mission Computer

Middleware has become a fundamental part of any distributed environment because it offers essential features and functionalities (Mohamed et al., 2014). Mohamed et al. (2013) stated that "*it is practically impossible to build large-scale or complex distributed applications*

without involving middleware" (Mohamed et al., 2013, p. 6). Middleware is found all over the system regarding distributed computing, and it plays different roles. For instance, it can be used for the underlying communication features, for translation and data integration from various sources, as a tool used to clarify application development, and many other functionalities. In addition, middleware connects any set of components that are part of distributed systems, such as operating systems, applications, network stacks, and hardware (Mohamed et al., 2013).

Since middleware is a crucial part of developing a UAV architecture, the most common middleware solutions used by the community will be addressed. The literature reviewed on low-cost UAVs from Chapter 2.1 identified that ROS, DroneKit, FlyOS, APSync, Maverick and Rpanion-server are the most widely used middleware (ArduPilot Dev Team, 2021b).

ROS is open-source software that develops libraries, tools, and an accessible environment to develop robotic applications (Alsalam et al., 2017). This middleware was developed to facilitate and expedite the prototyping of a robotic system (Bernardeschi et al., 2019). For example, Day et al. (2015), for multiple UAVs, uses ROS to communicate with UAVs in the swarm. It enables particular swarm members to autonomously perform navigation decisions based on the states of their neighbours (Alsalam et al., 2017). ROS organises its processes into nodes that communicate with each other concurrently by exchanging messages in a publisher/subscriber scheme. Those messages are published in topics (Quigley et al., 2009).

DroneKit Python is an open-source and community-driven project that allows the community to create powerful apps for UAVs. For example, for their UAV Agricultural applications, Psirofonia et al. (2017) used DroneKit to enhance the autopilot, adding greater intelligence to the UAV behaviour and performing computationally intense tasks, in particular, computer vision, path planning, and 3D modelling. Dronekit library enables sending MAVLink commands from an onboard computer to a flight controller, expanding different applications. However, ROS has its specific ROS package of MAVLink, MAVROS (ROS.org, 2018). This package provides communication between several systems using MAVLink. Moreover, ROS can use the MAVROS package to gain the same functionality as DroneKit while at the same time opening doors to several other ROS tools and utilities (Kim & Lee, 2019).

Kim & Lee (2019) agree that ROS is more accessible than DroneKit because ROS, supports, natively, C++ and Python (among others through non-native processes), whereas DroneKit supports only Python. Another concern regarding DroneKit is that it has a problem with extending to a heterogeneous collaboration system. In contrast, ROS enables the communication between multiple processes using message passing, solving the communication of heterogeneous UAVs issue (Kim & Lee, 2019).

It is essential to state that ROS has two different versions, namely ROS 1 and ROS 2. ROS 1 was developed in 2007, and due to many years of experience, it was possible to establish important features missing which could be improved. In fact, ROS 1 lacks crucial real-time safety, certification, security, and increased distributed processing requirements. For those reasons, one of the goals of ROS 2 is to be compatible with industrial applications (Yuhong Lin, 2018). ROS 1 itself does not permit more than one node in a process and has no standard framework for writing about nodes, which means that each developer has the freedom to develop a unique product. Conversely, ROS 2 enables various nodes in the same process and provides a predefined convention that saves users time and allows cleaner and more structured programs (The Robotics Back-End, n.d.).

The choice between both ROS versions will depend on the application and the level of ROS understanding. For example, ROS 1 is recommended for beginner developers, students, and Robotics Algorithm Developers. On the other hand, ROS 2 is advised for professional computer learners and developers who value real-time applications (Yuhong Lin, 2018).

2.3.5 Middleware Interaction with Pixhawk

Since this dissertation will use the Pixhawk autopilot, it is vital to understand how the interaction between the mission computer software application and Pixhawk firmware is carried out. Since we are addressing two types of middleware, Dronekit and ROS, the features of each are presented.

Firstly, even though DroneKit is working towards full compatibility with PX4 firmware, it is not entirely working with this firmware because it is not formally tested (PX4 User Guide, n.d.-b). Moreover, the community that tested DroneKit with PX4 raised some issues related to parameter handling due to DroneKit implementing the Ardupilot variant that is not compatible with PX4. Additionally, the waypoint missions upload shows errors related to the planned home location, and some PX4 features are not supported. On the other hand, for Ardupilot, Dronekit can be a great choice because this middleware was built with Ardupilot in mind (PX4 User Guide, n.d.-b). PX4 supports both ROS versions with different configurations. In particular, ROS 2 communicates with PX4 over the PX4-ROS 2 bridge (PX4 User Guide, 2021e). This interface directly connects PX4 uOBR messages and ROS 2 DDS messages and provides direct access to PX4 from ROS workflows and nodes in real-time. Apart from this configuration, there are other two, ROS 1 via ROS 2 bridge and ROS 1 via MAVROS. In this last configuration, the communication is engaged through MAVLink (PX4 User Guide, 2021b).

2.3.6 Communication through MAVLink

MAVLink is a messaging protocol that provides communication with UAVs, specifically onboard drone components. In addition, it allows up to 256 UAVs to communicate at the same time in the same frequency bandwidth (Fuller et al., 2014). This protocol is an open-source library and is free for public users (Erdelj et al., 2019). MAVLink is also compatible with Pixhawk Autopilot and both firmware Ardupilot and PX4 (ArduPilot Dev Team, 2021d; PX4 User Guide, 2022a). Furthermore, MAVLink can be used for both onboard and offboard communications. For example, it is possible to communicate between the autopilot and a GCS and between the autopilot and mission computer (ArduPilot Dev Team, 2021d).

2.3.7 Simulator

Since the testing of the software architecture is usually performed in a simulated environment, it is necessary to address different simulator options in order to recreate the aircraft and a real environment. For this reason, the simulator must guarantee compatibility with one of the flight stacks, Ardupilot or PX4, and provide a simulating flight with a fixed-wing aircraft. Gazebo, Mission Planner Simulator, JSBSim, and Flight Gear (ArduPilot Dev Team, 2021g) are the simulators that meet the two requirements mentioned above and which are the most used by the community.

Gazebo simulator is highly recommended by the PX4 User Guide (2022b) and is a powerful 3D simulation for object avoidance and computer vision. One the other hand, FlightGear is a more complex simulator regarding physical and visual realistic environments. It also provides several weather conditions and types of atmospheric flows, which require better graphical performance from the computer. In the same line of performance requirements, the JSBSim is an advanced flight simulator that provides realistic flight dynamics based on wind tunnel data (PX4 User Guide, 2022b). Finally, the Mission Planner simulator allows a simpler interface of the aircraft behaviour but is only supported by the Ardupilot (ArduPilot Dev Team, 2021g).

2.4 Guidance and Control Techniques

Different methods have been proposed in the literature for tracking ground targets. In the early days of target tracking, Dickmanns & Mysliwetz (1992) developed an algorithm to detect roads that employed a control strategy based on total state feedback and used an extended Kalman filter. Additionally, Taylor et al. (1999) developed a simple edge detection algorithm

and compared different control strategies, for instance, lead-lag control, full state feedback, and input-output linearization, to solve the road tracking issue.

One line of research defined periodic reference flight trajectories for a UAV to navigate via waypoint and maintain proximity with the target. For example, Lee et al. (2003) introduced a path-planning algorithm for a UAV to track a ground vehicle. The ground target changes its heading and varies its velocity from a standstill position up to the UAV speed. Moreover, Husby (2005) proposed a path algorithm with three possible patterns (square wave, circular and bow tie-shaped standoff), assuming that the target's position and velocity are known and the UAV has a visual range of the target. Finally, R. W. Beard (2007) developed two classes of flight trajectories for tracking ground-based objects. The first one is called longitudinal trajectories, which are more suitable for targets with the same speed as the UAV. The other trajectory is an orbital, considered to have more advantages for slow targets.

Quintero & Hespanha (2014) recognize that the greatest amount of research regards the standoff tracking issue in the target tracking area. In order to solve this problem, Frew et al. (2007) used a Lyapunov guidance vector field (LGVF) to enable a UAV orbiting a moving target at a fixed, planar standoff distance, assuming no wind speeds. However, this method can lead to oscillatory behaviour when there is a proximity between the target speed and the UAV speed or wind speed (Chen et al., 2009; Wise & Rysdyk, 2006). Therefore, Chen et al. (2009) proposed a tangent and Lyapunov field vectors that enable the UAV to converge to the standoff circle faster than the previous LGVF method. In addition, through differential geometry, Oh et al. (2013) proposed a rendezvous and standoff tracking guidance. This method permits the UAV to track accurate information without the target's noticing. Finally, Zhu et al. (2013) presented a saturated heading rate controller based on a vector field to regulate the relative course angle to the target while the velocity remains constant. This approach guarantees that the UAV, initiating with an arbitrary state, can converge to a desired circular orbit around the target.

Another line of research uses vision sensors based on nonlinear feedback control of the UAV heading rate to track a ground vehicle. Dobrokhodov et al. (2008) proposed a vision-based control algorithm with a gimbaled camera onboard the UAV to maintain the target in the centre of the image frame. Once the target position is unknown, the authors rely on the information attained by the image processing software for feedback. Along the same lines, Quintero & Hespanha (2014) adopted a vision-based target tracking which maintains the target in the scope of an onboard camera with enough resolution for visual detection. This research proposed two different optimization-based control policies, the game theoretic approach and the stochastic optimal control approach, which enable the UAV to maintain visibility and

proximity to the target. In the same line of work, Li et al. (2010) used a gimbaled camera in the UAV to follow a target. In particular, the UAV was equipped with a single gimbaled pan/tilt camera and a high bandwidth wireless link video to accomplish the vision-based target tracking and motion estimation. This system allowed the UAV to maintain a horizontal circular orbit around the target to estimate its position, speed, and heading (Li et al., 2010).

Still, on tracking through sensors, Ariyur & Fregene (2008) described a system for tracking a ground target from a UAV through delayed estimation of target motion from noisy sensors. This algorithm estimates the vehicle's acceleration and uses a point mass model to estimate its velocity and location. Afterwards, these estimations are used to produce waypoints for the UAV (Ariyur & Fregene, 2008).

Kuwata & How (2004) proposed a receding horizon control (RHC) to design trajectories for aerial vehicles in three-dimensional terrain. More recently, Yao et al. (2017) presented a three-layer distributed control structure to create an optimal multiple UAV trajectory to seek a target in a complex environment based on RHC and Gaussian mixture model (GMM).

On another perspective, X. Wang et al. (2014) proposed a vision-based detection and tracking of a mobile ground target using a fixed-wing UAV through an improved Zhu & Wang (2012) bang-bang heading rate controller. This research proposes to achieve a desired circular tracking of a ground target, trying to avoid exposure to the UAV.

Other authors used the moving path following (MPF) technique to address target tracking problems. Initially, Oliveira et al. (2013) presented a control method that relies on a path following control strategy to track a fixed radius circumference that moves together with the target that emits its position through a GPS beacon. In this research, the fixed-wing UAV is assumed to maintain its airspeed constant by the inner loop controller. Moreover, Oliveira et al. (2013) approach showed no oscillating issues when the speed of the UAV is close to the target speed. More recently, Oliveira et al. (2016b) introduced the MPF control law while control was derived using Lyapunov methods. This new method has overcome the problems of monitoring moving objects using classical path following algorithms. In the MPF, the authors assume that the UAV flies at constant altitudes and speeds. In this research, the UAV was required to converge to and follow a specific moving path without temporal specification, resulting from a generalization of the classical path following.

In this dissertation, the MPF was the algorithm implemented as it allowed operating in every conditions disregarding the relative initial position between the target and the UAV. Moreover, the oscillating behaviour is eliminated due to the target speed approximating the UAV speed. Finally, it allows using any geometric path shape (Oliveira et al., 2016b).

2.5 Ground Vehicle Detection in Airborne Images using Computer Vision

In the current work, the main goal is to have the UAV to follow a target, completing the control loop based on a sensor. For the sensor to detect an object, several methods can be adopted, such as radar (Schwartz, 1990), an operator (van Breda, 1995), and the method adopted in this research, computer vision (Borji et al., 2019).

Larry Roberts first introduced computer vision at the Massachusetts Institute of Technology (MIT) in the 1960s through his Ph. D thesis, which discussed the possibility of recovering "*3-D geometrical information from a 2-D perspective view of blocks*" (Huang, 1996, p. 21). This task can be represented as a pipeline of sequential tasks. The first one tries to extract information from an image, such as the location of an object or its size. This task consists mainly of applying image processing techniques, which vary according to the state-of-the-art. In a subsequent task, some assumptions about geometry and the image formation process extrapolate information from a 2D signal to a 3D representation.

In parallel, since the 1950s, a small subsection of Artificial Intelligence (AI) known as Machine Learning (ML) has revolutionized different fields. Neural Networks (NN) is a set of techniques that integrates ML, and it was the field that spawned Deep Learning (DL). Since 2012, DL has shown outstanding success in image processing tasks, such as image classification, object detection and object segmentation. This makes DL a strong candidate to be used in the first task of the computer vision pipeline (Alom, 2018).

Object detection can be accomplished through several approaches that can be divided into traditional and deep-learning-based (Nguyen et al., 2020). Alom (2018) stated in his research that for detection problems, DL-based regression approaches such as You Only Look Once (YOLO) and Single Short Multiple-box Detector (SSD) provide superior performance facing traditional methods. Since DL has been dominant in image classification, segmentation, and detection (Alom, 2018), several ways introduced by the literature will reviewed.

There are two types of DL methods of object detection, the two-stage approaches, which are the ones based on region proposal algorithms (R-CNNs), and one-stage approaches based on regression or classification recognized (YOLO and SSD) (Nguyen et al., 2020). On the one hand, two-stage methods generally achieve high accuracy but are slower. On the other hand, one-stage methods are much faster than two-step methods, but have reduced accuracy (Dorrer et al., 2019).

2.5.1 Region-Based Conventional Neural Network (R-CNN)

Region-Based Conventional Neural Network (R-CNN) is a method proposed by Girshick et al. (2014) that provides more than 30% mean average precision (mAP) than related to previous best results on PASCAL VOC 2012 (Girshick et al., 2014). The R-CNN architecture includes four stages, firstly, it resizes an image to 227×227 pixels and takes it as an input. Secondly, it applies a selective search algorithm and generates 2000 candidate region proposals that, in the next phase, are wrapped into a square and inserted into a CNN that creates a 4096-dimensional feature vector. Afterwards, using a Support Vector Machine (SVM), the features extracted from the image are classified (Kim et al., 2019).

Nguyen et al. (2020) describe that R-CNN's main advantage is the use of CNN features learned from real images. This makes them more discriminative representations than the low-level image features, which are hand-crafted. On the other hand, the R-CNN is considered to be extremely costly in space and time (Girshick, 2015) and wasteful because it has to apply CNN 2000 times (Girshick, 2015; Kim et al., 2019; Nguyen et al., 2020). Moreover, once R-CNN resizes the image to 227×227 pixels, it constitutes an issue to small object detection (Nguyen et al., 2020).

2.5.2 Fast R-CNN

Fast R-CNN is a method developed by Girshick (2015) that includes innovations to improve the drawbacks of R-CNN, mainly the speed and accuracy (Girshick, 2015). Although Fast R-CNN is similar to the R-CNN approach, instead of providing the region proposals to the CNN, the CNN takes an image of any size as input and several Region of Interest (ROIs) and generates a convolutional feature map (Nguyen et al., 2020).

As a result, it has higher average precision of detection than R-CNN. Moreover, the training stage is a single-phase, using a multitask loss, and can update the entire network layers (Gandhi, 2018).

2.5.3 Faster R-CNN

Ren et al. (2017) introduced a Region Proposal Network (RPN) that "*shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals*" (Ren et al., 2017, p. 1). This research is an improvement of Fast R-CNN and is called Faster R-CNN. The methods mentioned above (R-CNN and Fast R-CNN) use selective search to designate the region proposals. Consequently, those techniques affect the network due to being a slow and time-consuming process (Gandhi, 2018). Moreover, Ren et al. (2017) stated

that the Faster R-CNN accomplishes near real-time rates by avoiding the time used on region proposals.

The overall object detection accuracy and region proposal quality are enhanced with Faster R-CNN (Ren et al., 2017). In addition, Nguyen et al. (2020) considered the RPN to improve the running time because it generates fewer proposal boxes due to sharing computation on convolutional features. However, although RPN and Fast R-CNN combination enable improved accuracy, it leads to a processing speed reduction (Nguyen et al., 2020).

2.5.4 YOLO

YOLO, as the name suggests, "*you only look once at the image to predict what objects are present and where they are*" (Redmon et al., 2016, p. 1). This method was developed to create a one-step process through a single convolutional network that predicts multiple bounding boxes and class probabilities for those boxes (Pedoeem & Huang, 2018; Redmon et al., 2016). Contrary to the R-CNNs process, YOLO trains on entire images and optimizes detection performance directly (Redmon et al., 2016). The authors considered that YOLO is an extremely fast detection algorithm (45 FPS) due to its test time to predict detections. Moreover, YOLO's capacity to analyze the whole image reduces to half the background errors compared to Fast R-CNN, meaning high accuracy. Finally, this method learns generalizable representations of objects, enhancing the possibility of detecting in new domains or unexpected inputs (Redmon et al., 2016).

One of the main drawbacks of YOLO is its accuracy compared to other state-of-art detection systems. Furthermore, this method lacks precisely localizing some objects due to its rapid object detection capability (Redmon et al., 2016). In view of these limitations presented, the authors introduced the YOLO v2 method that improved the accuracy but lacked small object detection (Redmon & Farhadi, 2016). Apart from these three versions of YOLO, Pedoeem & Huang (2018) suggested the YOLO-LITE method. This technique solved the real-time problem in a non-GPU computer once both YOLO and R-CNN frames per second (FPS) in a non-GPU computer were impractical for real-time detection (Pedoeem & Huang, 2018).

Later YOLO v4 was developed by Bochkovskiy et al. (2020). This method is influenced by the state-of-art Bag of Freebies (BoF) and Bag of Specials (BoS) (Bochkovskiy et al., 2020). The first method improves the detector's accuracy without affecting the inference time but increments the training computational cost. On the other hand, the second method increases object detection accuracy significantly while the inference cost rises by a small margin

(Supeshala, 2021). Regarding mAP and FPS, YOLO v4 has increased 10 percent and 12 percent of those parameters compared to YOLO v3 (Bochkovskiy et al., 2020).

Until this moment, PP-YOLO is the latest version of YOLO and was introduced by Long et al. (2020). The researchers aimed to develop an effective and efficient technique that could be utilized in actual application scenarios. Since YOLO v3 has been successfully used in practice, PP-YOLO is based on this method. In contrast to YOLO v4, this detector uses a specific backbone, ResNet (He et al., 2015) and data augmentation method, MixUp (Zhang et al., 2018). In Long et al. (2020) research, an experiment was conducted and demonstrated that PP-YOLO is faster (FPS) and more accurate (COCO mAP) than YOLO v4. Long et al., 2020, compared three different versions of YOLO and PP-YOLO is faster and more accurate than YOLO v4 and YOLO v3 (detailed comparison displayed in Table A-1 from Annex A).

2.5.5 Single shot multi-box detector (SSD)

Single shot multi-box detector is a method that detects objects in real-time in images using a single and one-stage deep neural network (Liu et al., 2016). This method comprises two parts: extraction of feature maps and convolution filters to detect objects (Nguyen et al., 2020).

Compared to the state-of-art two-stage processing, for instance, Fast R-CNN, which needs two shots (one for generating regional proposals and the other to detect the object of each proposal), SSD presents an improvement in speed of running time faster. This increase is a consequence of eliminating bounding box proposals and the subsequent pixel or feature resampling stage (Liu et al., 2016; Nguyen et al., 2020). On one hand, eliminating the proposal network causes a reduction in mAP, which is compensated by applying improvements such as multiscale features and default boxes. It permits the SSD method to use lower resolution images, therefore increasing the speed of processing (Nguyen et al., 2020). On the other hand, this makes SSD easy to train and straightforward to integrate into systems with limited resources. (Liu et al., 2016).

Experiments conducted by Liu et al. (2016) proved that SSD has significantly improved speed for high-accuracy detection compared to Faster R-CNN and YOLO v1 in the dataset VOC 2007 (detailed comparison displayed in Table A-2 from Annex A).

Even though the improvements mentioned above regarding the SSD are significant, this method presents difficulty detecting small objects (Nguyen et al., 2020). Therefore, Fu et al. (2017) introduced the Deconvolutional Single Shot Detector (DSSD), capable of enhancing the accuracy for detecting small objects and the additional large-scale context in object detection.

2.5.6 UAV Vehicle Detections Applications

Vehicle detection in aerial images is challenging due to three main factors: i) the variable vehicle size, ii) the high/low density of vehicles and iii) the complex background in the camera's field of view (Mandal et al., 2019). In addition, the variety of object types and their similarities lead to difficulties distinguishing the vehicle and non-vehicles objects in aerial images. Dorrer et al. (2019) consider vehicle detection a difficult task due to the small size of vehicles, the monotone appearance and the complex background.

Mandal et al. (2019) proposed a lightweight network for vehicle detection in aerial scenes called SSSDet. This method is a simple and shallow convolutional network optimized for fast detections and high accuracy in object detection and classification in a single step. Mandal et al. (2019) measured the mAP of the proposed SSSDNet, YOLO v2, YOLO v3, YOLO v3 tiny, Faster R-CNN and RetinaNet performances in three different datasets (VEDAI, DLR-3K, DOTA), concluding that their network outperforms the other state-of-art detectors (detailed comparison displayed in Table A-3 from Annex A). Furthermore, an analysis for real-time applications was conducted by Mandal et al. (2019), showing worst performance of SSSDNet in terms of inference speed (FPS) than YOLO v3 tiny (detailed comparison displayed in Table A-4 from Annex A).

Dorrer et al. (2019) proposed an external region method, RetinaNet, for vehicle detection and a focal loss function to distinguish complex negative samples from actual vehicles. In this research, since the authors considered aerial images that presented small-scale objects, one-stage detectors, such as YOLO and SSD, were used to negatively affect vehicle detection performance.

Although many existing datasets contain different types of vehicles, for instance, ImageNet (Krizhevsky et al., 2017), those are not applicable for vehicle detection in airborne images (Dorrer et al., 2019). From the datasets available for aerial images, Dorrer et al. (2019) do not recommend the use of VEDAI (Razakarivony & Jurie, 2016) and DLR 3K (Kang Liu & Mattyus, 2015) due to their low offer of vehicle images to train the CNN model. Afterwards, a large-scale dataset was developed to provide 1200 images with 75567 vehicle images for training (Dorrer et al., 2019).

Regarding Dorrer et al. (2019) research, experiments evaluating the precision rate and recall rate of RetinaNet, YOLOv3 and SSD512 with several Intersection over Union (IoU) and the proposed dataset were conducted. As a result, Dorrer et al. (2019) concluded that RetinaNet outperformed both state-of-art detectors (detailed comparison displayed in Table A-5 from Annex A).

Mandal et al. (2020) detect small-scale vehicles in aerial scenes through a proposed one-stage vehicle detection network (AVDNet). However, once the authors stated that YOLOv2, YOLOv3 and RetinaNet were more suitable for images captured from canonical perpective and required high memory space, those detectors were not an option for their research.

The effectiveness of AVDNet was performed on several datasets, namely, VEDAI, DLR-3K and DOTA and combined dataset (VEDAI, DLR-3K, DOTA and ABD). In addition, the airborne image data set (ADB) was the dataset developed by Mandal et al. (2020) for further experiments. The performance measured by AVDNet, YOLOv2, YOLOv3 and RetinaNet proved that the proposed network outperforms, in terms of mAP, the other state-of-art methods in all four datasets (detailed comparison displayed in Table A-6 from Annex A).

Lu et al. (2018), Ju et al. (2020) and Cepni et al. (2020) used YOLO as vehicle detection methods in their research. Lu et al. (2018) processed and integrated three public datasets of aerial images, VEDAI, COWC and DOTA, for testing YOLO. The authors concluded that the model was suitable for small, rotating, compact and dense objects, showing good test results (mAP =76.7%) and meeting the real-time requirements.

In addition, Ju et al. (2020) proposed an improved YOLO v3 method to detect vehicles through aerial images. To accomplish that, the authors redesigned the backbone of YOLO v3 for resizing the aerial images, improved the loss function of the original YOLO v3 and verified the performance with experiments on VEDAI dataset. This improved method showed better results in detecting small vehicles than the original YOLO v3 since the mAP of the improved YOLO v3 was 70.3% and in the original was 62.8%.

Cepni et al. (2020) trained three YOLO v3 versions, original, v3-spp and v3-tiny with COCO data and were tested through a video obtained by a UAV. The results showed that YOLO v3-spp outperformed compared to the other versions with 63.53% mAP.

3 System Architecture

The hardware and software architecture proposed in this thesis is depicted in Figure 5. The proposed hardware based on Silva et al. (2020) is displayed on the left side of the image. On the right side, it is presented the software components selected for the architecture based on the requirements imposed by the hardware, as well as the need to run the computational component of trajectory control and computer vision on board the aircraft. Therefore, a discussion on the choice of each software component is held.

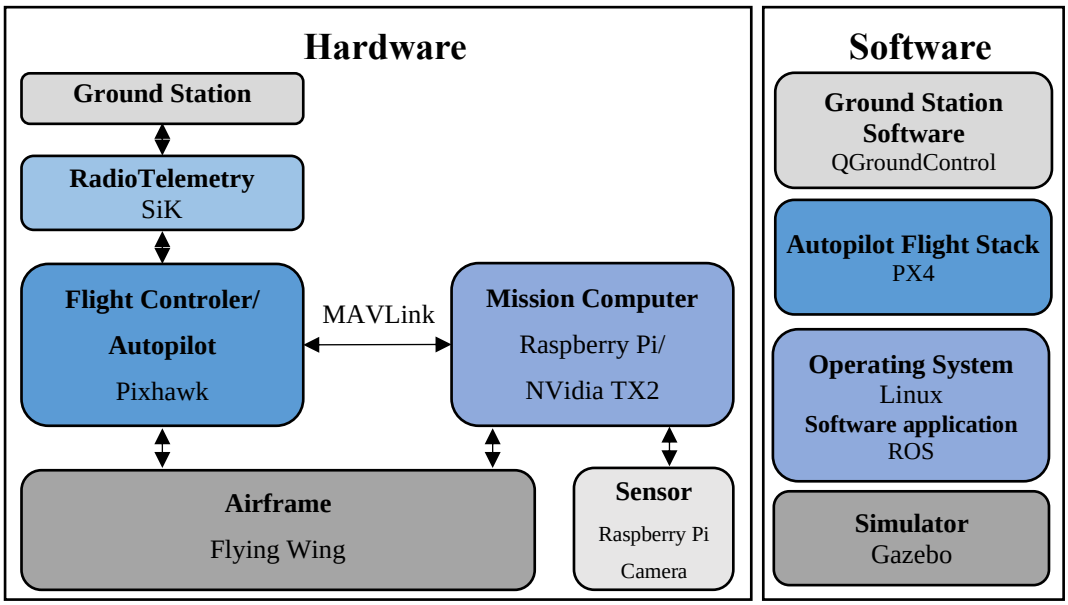


Figure 5. Hardware and Software architecture.

Concerning the autopilot, the Pixhawk has two options for the flight stack: the Ardupilot or the PX4. Despite advantages of the Ardupilot regarding the broader and more active community and information sharing and progress over the PX4, the Ardupilot lacks a main requirement for this thesis, i.e., the Offboard mode (ArduPilot Dev Team, 2021c). The Offboard mode is the method that provides attitude and thrust control to the fixed-wing aircraft, and the PX4 flight stack is the one to have it (PX4 User Guide, 2021a). In addition, choosing PX4 over the Ardupilot was driven by easy understanding for beginners, which is decisive in this research (PX4 User Guide, n.d.-a). Furthermore, it allows easy connectivity with software like ROS, QGC and Gazebo, all used in this dissertation.

As for the ground station software, several options could be adopted, such as QGC, MAVProxy, Mission Planner and APM Planner 2. However, among these ground stations, only the QGC provides full compatibility with the chosen flight stack, PX4, so it is the software used in this thesis (LambDrive, 2016b).

Regarding the mission computer software, the proposed software application to be used is ROS library due to its compatibility with PX4, which was not the case with the Dronekit. The use of ROS thus limits the operating system of the mission computer that is Linux (PX4 User Guide, 2021b). The decision to choose ROS 1 over ROS 2 was based on four factors: i) ease of support for installing the development tool chain (PX4, Gazebo and ROS Melodic), ii) compatibility with CIAFA software applications, which are based on ROS1, iii) CIAFA's greater experience using ROS1, and iv) the complexity of the present work doesn't require the real-time guarantees and certification provided by ROS2.

Lastly, the choice of the Gazebo 9 simulator was based on the recommendations made by the PX4 community and its scalability, which permits friendly usage and easy and fast access to other systems such as PX4 or ROS. Another reason was the availability of the PX4 website with the installation of the complete toolchain of PX4 flight stack, ROS and Gazebo 9 (PX4 User Guide, 2022b).

3.1 Software Architecture

The software architecture developed in this research is presented in Figure 6. The architecture can be divided into four major parts, the PX4 SILT interface, the ROS environment, the QGroundControl and the Gazebo simulator.

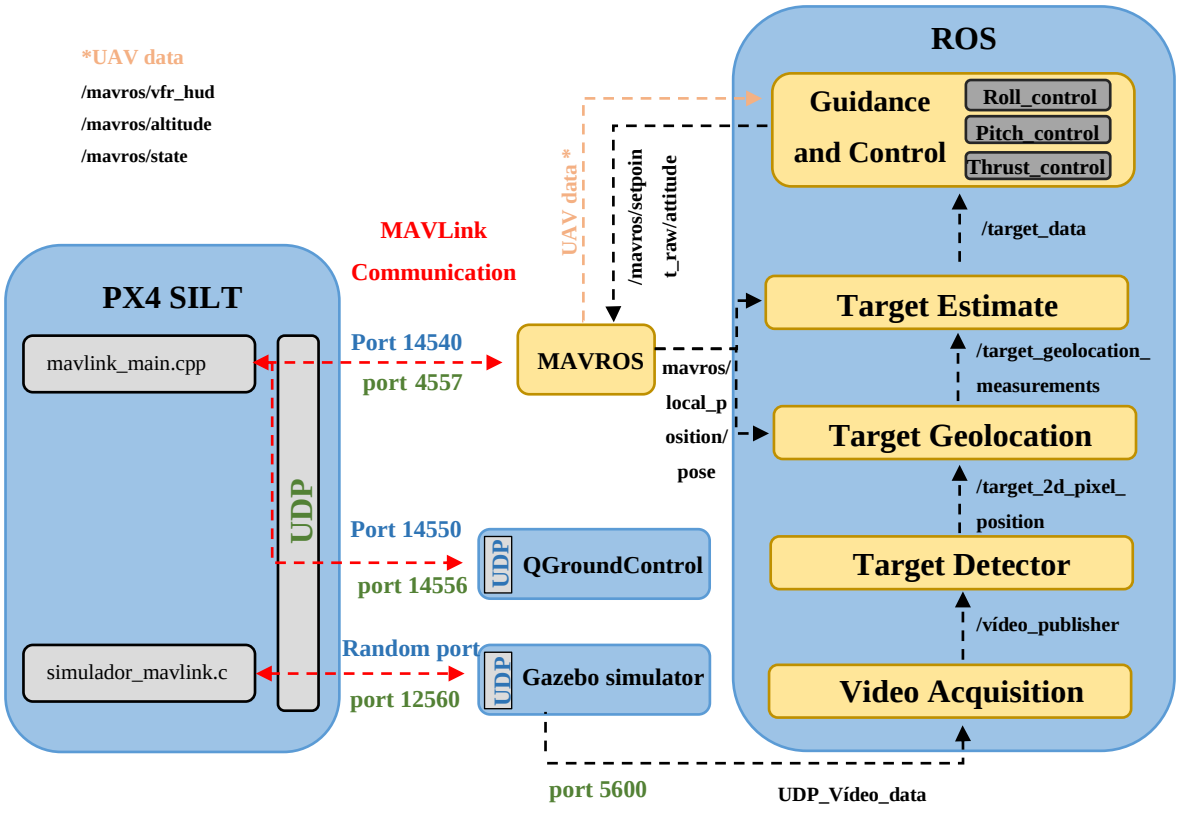


Figure 6. System Software architecture.

The module PX4 SILT represents the PX4 flight stack and is the component that will receive information via MAVROS from the ROS Environment module, as well as from QGroundControl and Gazebo simulator, through a UDP port. The PX4 will use the information from the ROS environment module (through the MAVROS node) to command the UAV aerodynamic surfaces deflection and thrust settings.

The Gazebo simulator module provides a realistic environment for the SILT simulations. From the received information, the simulator reproduces the commands in the aircraft. In addition, there is a connection through UDP port from the simulator to the Video Acquisition node sharing the aircraft camera video displayed in the Gazebo.

The QGroundControl block represents the software from the ground station. The link between the QGC and PX4 autopilot, allows the exchange of information between the simulation interface and ground control. As a result, it is possible to follow the flight and monitor several parameters from the aircraft through the QGC, for instance, airspeed, heading, altitude, flight mode, etc. In addition, it is also possible to file a flight plan from the QGC to the PX4 autopilot, change the flight mode and a handful of other functionalities.

Finally, the ROS environment block contains the following sub-modules: Guidance and Control, Target Estimate, Target Geolocation, Target Detector and Video Acquisition. Each sub-module corresponds to a ROS node. The Target Detector node is divided into two parts; initially, a colour filter is implemented to detect a red box to validate the Guidance and Control algorithm. Afterwards, this detection module is replaced by a neural network-based approach to detect a ground vehicle.

In the ROS environment, the data is transferred over ROS topics. Regarding Figure 6, it displays several ROS topics, namely, UAV data published by the MAVROS and subscribed by the Guidance and Control node and *local_position_pose* message subscribed by Target estimate node and Target Geolocation. Both these ROS topics correspond to telemetry data regarding altitude, airspeed, flight mode, etc.

Inside the ROS module, five ROS topics are used: i) *video_publisher*, ii) *target_2d_pixel_position*, iii) *target_geolocation_measurements*, iv) *target_data* and v) *setpoint_raw/attitude*. First, the *video_publisher* is the topic where the images from Gazebo are published for the Target Detector node. Next, the *target_2d_pixel_position* is a topic that publishes the 2D coordinates from the target's position in the camera frame in pixels and will subsequently be used by the Target Geolocation position estimation. Afterwards, the *target_geolocation_measurements* topic represents the NED coordinates calculated by target geolocation and is subscribed by the Target Estimate node.

Subsequently, the *target_data* topic represents the output from the Kalman Filter estimation. Having two separate ROS nodes, i.e., one for position measurement through the Target Geolocation node and the other node for parameters estimation by the Target Estimate, it enables the *target_geolocation_measurements* rate to be distinct from the *target_data* computation rate (which includes information regarding the target position, velocity and heading). This makes it possible to use filter estimation and prediction in the case of no measurements by the Target Geolocation node.

Finally, the Guidance and Control node subscribes to the *target_data* topic and publishes the *setpoint_raw/attitude* topic. The *setpoint_raw/attitude* topic includes a command message (*mavros_msgs/AttitudeTarget.msg*, which includes commands to yaw, pitch, roll and thrust) sent to the PX4 autopilot. The Guidance and Control node comprises the PI lateral (Roll_control) and longitudinal (Pitch_control and Thrust_control) controllers.

Implementing a modular architecture allows various algorithms to be developed and tested for later implementation in UAVs. Each of these modules will be described in the following sections.

3.2 Gazebo Simulator

The Gazebo simulator is the module responsible for creating complex worlds (Koenig & Howard, 2004) that will be encountered in reality by both UAVs and unmanned ground vehicles (UGV) (Shin, 2019). Additionally, its diverse library of air vehicles, open-source community, high fidelity and precise control makes this simulator a crucial piece between "*the drawing board and the real hardware*" (Koenig & Howard, 2004, p. 1).

3.3 PX4 Autopilot

PX4 autopilot architecture can be divided into two layers, the flight stack and the middleware. The first one represents the software of the flight control system and estimation; on the other hand, the middleware supports all types of UAVs and UGVs, providing hardware integration and communication (PX4 User Guide, 2021f).

The software architecture from PX4 is displayed in detail in Figure B-1 (see Annex B). Generally, the upper part of the diagram consists of the middleware blocks, and the lower layer contains the flight stack's components. The source code is broken down into self-contained modules. Each building block is considered a module, and the arrows between them represent the flow of information and connections. There are many more links than the ones defined in the diagram, and some data is shared by most of the modules. The communication between

modules is conducted by publish-subscribe messages known as uORB. This type of communication can be reactive, fully parallelized, and open to consume data from anywhere inside the system (PX4 User Guide, 2021f; Shin, 2019).

The flight stack, shown in Figure 7, is composed of guidance, navigation and control algorithms for drones and controllers, attitude and position estimation for fixed wing, multirotor and VTOL. The flight stack architecture includes several elements, from sensors, RC input and outer-loop flight control laws (referred to as Navigator in Figure 7) to the motor and servo control (actuators). The estimator combines one or more sensor inputs and computes a vehicle state. For example, an IMU sensor can measure the estimated attitude.

Generally, a controller is an element that takes a setpoint and an estimation state, also known as a process variable. Afterwards, the process's variable is adjusted to match the setpoint and finally outputs the correction to reach the setpoint. For example, in Figure 7, the input is the desired position (setpoint); the process variable is the estimated position; the output is the attitude and thrust that converges the vehicle to the predetermined setpoint. Finally, the mixer takes force commands and converts them into actuator inputs. Those actuator commands will vary from aircraft type (PX4 User Guide, 2021f).

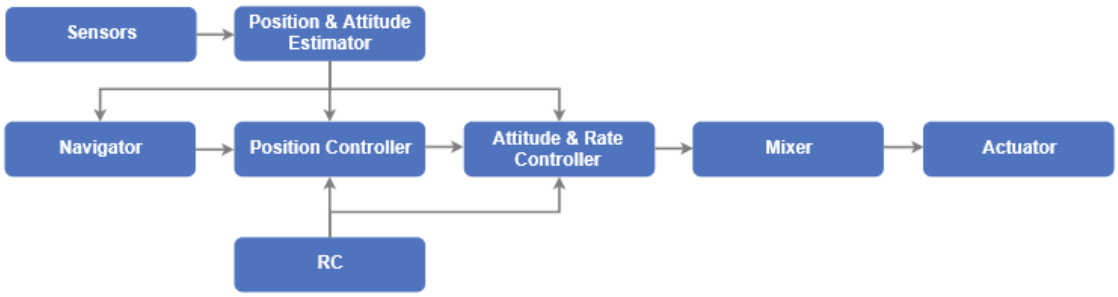


Figure 7. Flight stack architecture (Adapted from 'PX4 Architectural Overview', 2021).

The middleware includes drivers of systems devices for installed sensors, communication with the external environment and the PX4 internal communication mechanisms, the uORB publish-subscribe message bus. Moreover, the middleware allows PX4 code to be executed on the desktop operating system to control a vehicle in the simulated environment (Shin, 2019).

3.4 ROS Environment

The ROS environment corresponds to the module that incorporates the Python code related to the Guidance and Control, Target Estimate, Target Geolocation, Target Detector and Video Acquisition.

3.4.1 Guidance and Control

The Guidance and Control node comprises the sub-blocks related to longitudinal, lateral and attitude control. The longitudinal control was computed through PID implementation, while the lateral control was done by implementing MPF algorithm developed by Oliveira & Encarnação (2013). Finally, the attitude control is performed by the flight stack of the PX4 displayed in Figure 7. Next, each one of these sub-modules are presented in detail.

3.4.1.1 Outer Loop Longitudinal Controller

The PID control is a feedback controller that provides excellent control performance despite the varied dynamic aspects of the process plant. This control method is extensively used in flight control and industrial control (Shin, 2019). Once PID stands for proportional, integral and derivative modes, choosing which modes will be used as well as associating them to specific parameters or settings is a key point. Willis (1999) states that the three basic algorithms used are P, PI, and PID.

Since this control system is simple to design, to implement, is response-based and requires no background knowledge in system modelling (Shin, 2019), it was used in this dissertation. In this research, the outer loop controller, including altitude and velocity, was developed as a PI control process to make it possible to command a specific altitude and speed to be followed by the UAV (Shin, 2019).

The outer loop controller used in this study is shown in Figure 8. Variables h_{cmd} and V_{cmd} are the altitude and velocity setpoints of the UAV, respectively. The longitudinal outer-loop controller computes the pitch and thrust used in the UAV to reach or maintain the altitude and velocity. Then, the outer loop lateral controller takes the target data and the UAV dynamics as input and determines the roll command (Φ_{cmd}) that allows the UAV to perform a loiter around the moving target. The target data list the position, velocity, acceleration, angular acceleration and Ψ of the target.

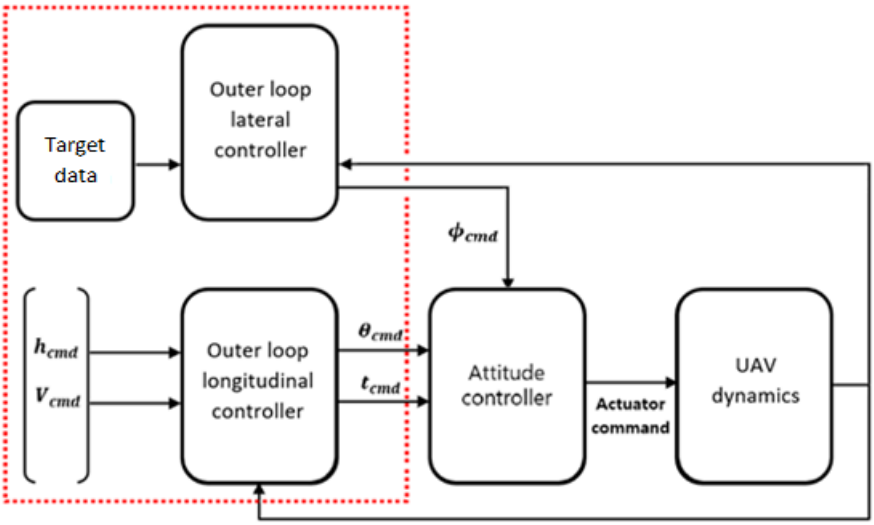


Figure 8. Outer loop controller.

The outer loop begins by computing the altitude and velocity errors through the actual altitude and velocity provided by the autopilot telemetry and the altitude and velocity setpoint. Variables V_{error} and h_{error} are the velocity and altitude error, respectively, and are described according to

$$V_{error} = V_{cmd} - V_{measured} \tag{3.1}$$

$$h_{error} = h_{cmd} - h_{measured}. \tag{3.2}$$

The K_p^V , K_p^h, K_i^V and K_i^h variables represent the proportional velocity gain, proportional altitude gain, integral velocity gain and integral altitude gain outer loop controllers, respectively. In this research, these parameters were set to $K_i^V = 0.3$ and $K_i^h = K_p^V = K_p^h = 0.2$. Variable T_{ss} stands for throttle steady-state and is set equal to 0.5. Variables t_{cmd} and θ_{cmd} are commanded thrust and commanded pitch, correspondingly, as displayed in equations (3.3) and (3.4)

$$t_{cmd} = T_{ss} + V_{error}\, K_p^V + \int V_{error}\, K_i^V\, dt \tag{3.3}$$

$$\theta_{cmd} = h_{error}\, K_p^h + \int h_{error}\, K_i^h\, dt\,. \tag{3.4}$$

Outer Loop longitudinal Controller Validation

This section presents two Gazebo simulations that validate both altitude and thrust controller in a climb, descent, and cruise scenarios. In the first simulation, the aircraft was initially standstill on the ground and then initiated a take-off to an altitude of 100 m at 15 m/s airspeed. After establishing this altitude, the aircraft started a loiter with a radius of 150 m at an airspeed of 15 m/s. This simulation was performed in 265.06 s.

Figure 9 to Figure 13 show the graphics with the parameters that sustain whether the PIs are valid. In all four graphs, two vertical lines are drawn. The green line represents the time instant when the aircraft reaches 100 m for the first time. Between the two vertical lines, the controllers converge to the setpoint values for airspeed and altitude and adjust the pitch and thrust for these values. The red vertical line corresponds to the moment from which the UAV has achieved a steady state.

Regarding Figure 9 and Figure 10, which correspond to the altitude PI controller, it can be observed that the aircraft, after establishing the 100 m height, can maintain it steadily. The pitch angle is constant during the climb and converges to -0.05 radians to maintain the altitude setpoint. Concerning Figure 11 and Figure 12, the thrust percentage oscillates between 25.18% and 100%; the airspeed has an average of 15.23 m/s during the climb. After the aircraft reaches 100 meters, the thrust converges to 23.55%, and the airspeed reaches the 15.33 m/s average. The settling time for the controllers to converge to the setpoints was 39.80s.

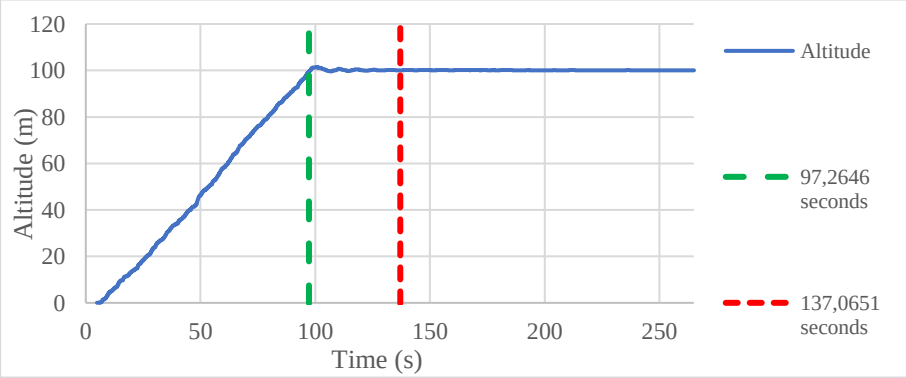


Figure 9. UAV climb to 100 m.

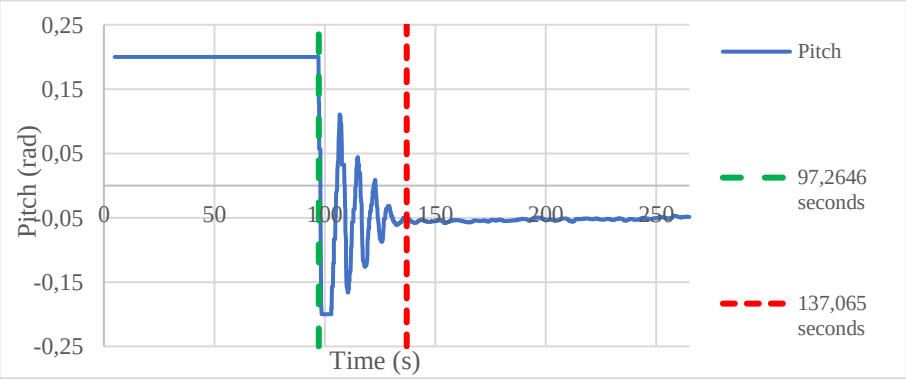


Figure 10. UAV pitch variation during climb to 100 m.

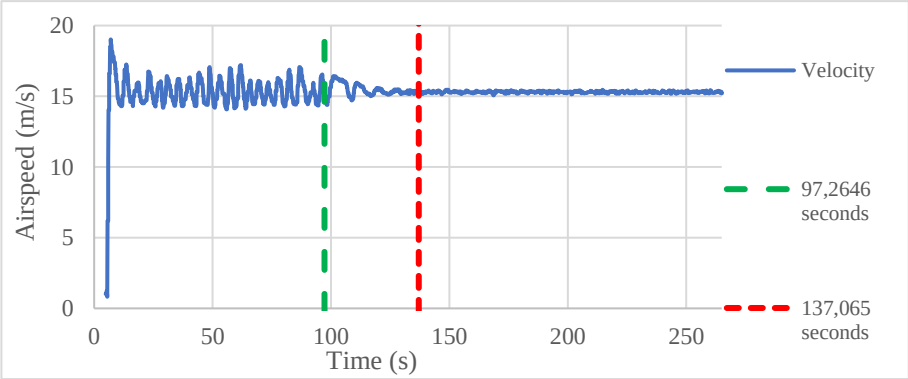


Figure 11. UAV airspeed variation during climb to 100 m.

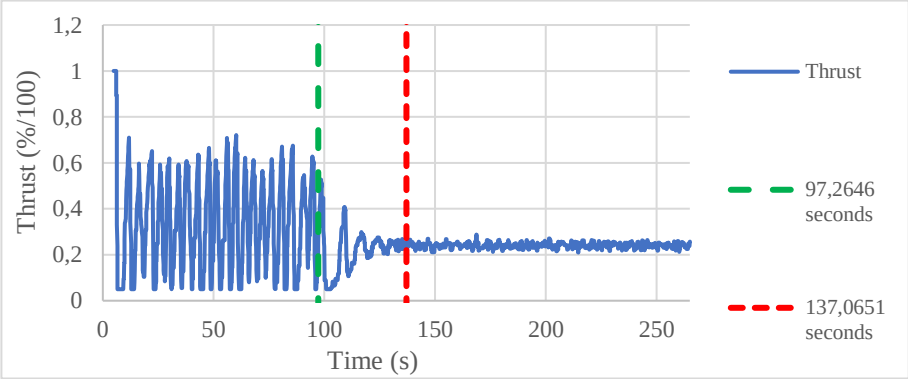


Figure 12. UAV thrust during climb to 100 m.

In the second simulation, the aircraft flew at an altitude of 100 m at 15 m/s and started a descent to 50 m at a constant airspeed. After establishing the predetermined altitude, the aircraft began a loiter with a radius of 150 m at an airspeed of 15 m/s. This simulation took 265.09 s.

Regarding Figures 13 to 16, the graphics show the parameters that sustain whether the PIs are valid during descent. In all four graphs, two vertical lines are drawn. The green line corresponds to when the aircraft reaches an altitude of 50 m for the first time. The area between these vertical lines presents the flight stage when the altitude and airspeed variables reach the setpoint values for airspeed and altitude and adjust the pitch and thrust for those values. The red vertical line determines the moment from which the controller is stable.

Figures 13 and 14 correspond to the altitude PI controller's input (altitude) and output (pitch). Those graphs show that the aircraft, when established at 50 m height, can maintain the altitude steadily. The pitch angle is constant during the descent and converges to -0.04 radians to maintain the altitude setpoint.

Regarding Figures 15 and 16, the thrust percentage oscillates between 27.97% and 5%, and the airspeed has an average of 16.14 m/s during the descent. The airspeed during the climb has exceeded the setpoint value due to the 5% minimum thrust predefined in the controller, to ensure safety of flight. After the aircraft reaches 50 m, the thrust converges to 23.56%, and the

airspeed reaches the 15.35 m/s average. This PI controller has a settling time of 32.89 s to be completely stable.

Concerning both simulations, i.e., climb and descent, the altitude and velocity are achieved and maintained correctly during the flight. Therefore, the PI controllers are validated to maintain the altitude and velocity in further simulations. More detailed information about both simulations is exposed in Table C-1 (see Annex C).

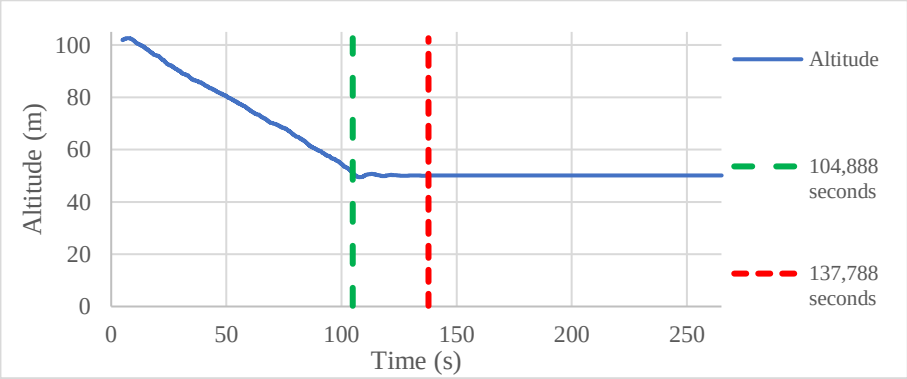


Figure 14. UAV altitude during descent to 50 m.

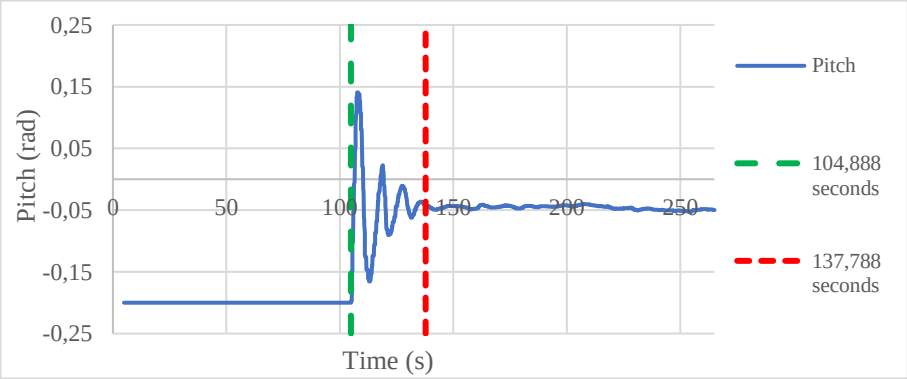


Figure 13. UAV pitch variation during descent to 50 m.

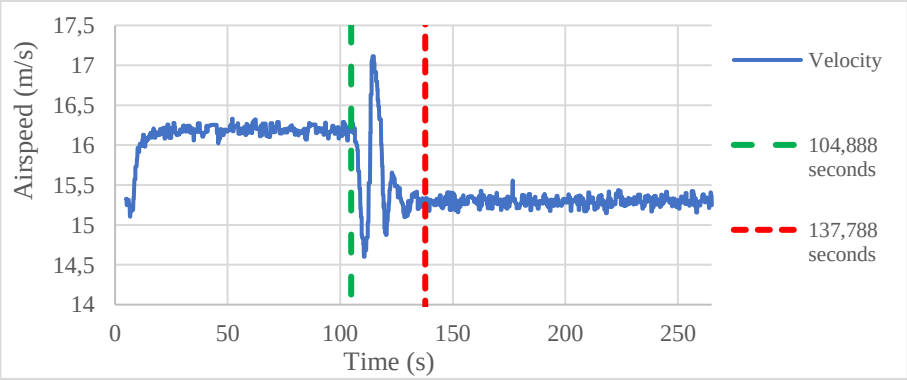


Figure 15. UAV airspeed during descent to 50 m.

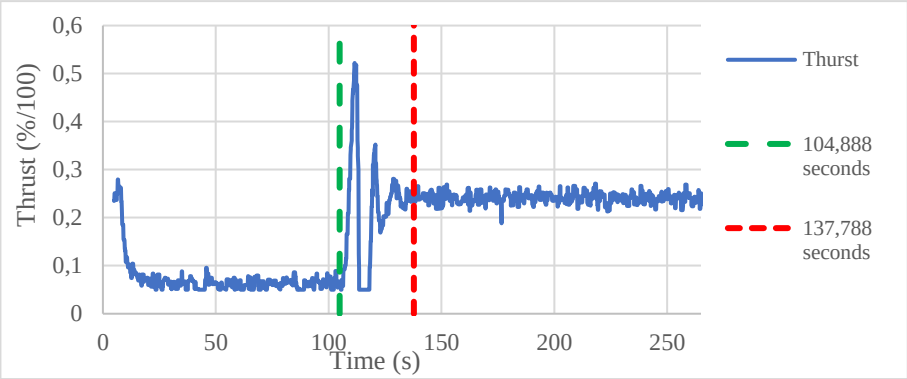


Figure 16. UAV thrust during descent to 50 m.

3.4.1.2 Outer Loop Lateral Controller

The control method used in this dissertation is the moving path following control method developed by Oliveira & Encarnação (2013). This method allows an autonomous vehicle to follow a circumference with a constant radius that moves jointly with a reference on the ground, as displayed in Figure 17.

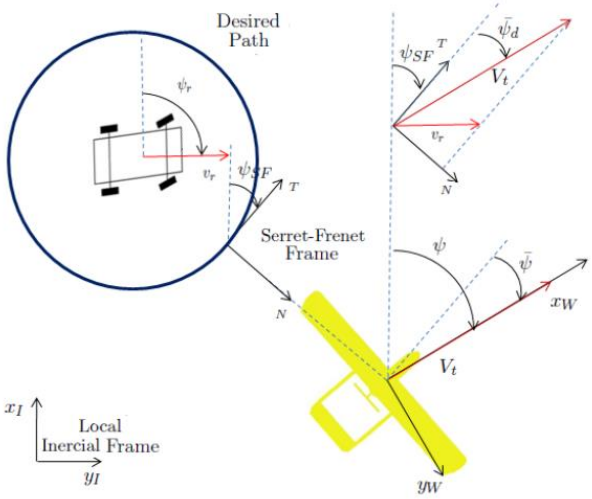


Figure 17. Moving-path following: relevant variables (Retrieved from Oliveira & Encarnação, 2013).

The control law was used in all conditions regardless of the relative position and velocity between the UAV and target. For this reason, the UAV performs a loiter when the target velocity is lower than the UAV. Conversely, if the target velocity approximates the UAV, the UAV behaves similarly to a controller that tracks a particular point on the path(Oliveira et al., 2013).

The control law was derived from Lyapunov methods, and the kinematic model for path following was written by considering a Serret-Frenet frame attached to the reference path. The Serre-Frenet frame is presented as $\{SF\} = [\vec{T}, \vec{N}, \vec{B}]$ in Figure 18. The x-axis is represented

along the tangent to the path, the y-axis is the normal vector to the x-axis and points to the observer's right that moves along the path positively to the tangent axis. Lastly, the z-axis points down.

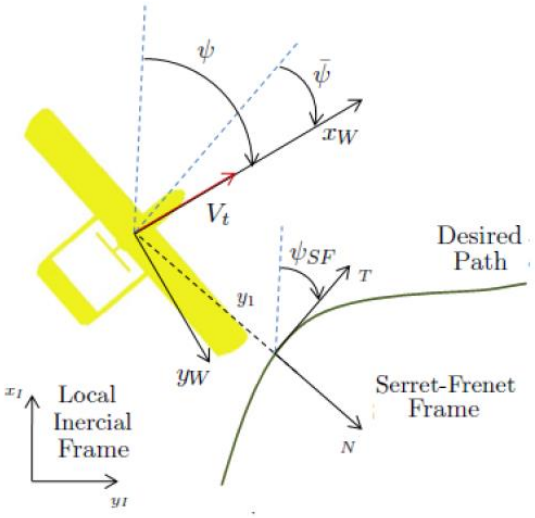


Figure 18. Path following lateral dynamics relevant variables (Retrieved from Oliveira & Encarnação, 2013).

The MPF considers a local inertial frame, which has its origin in the Ground Station. In addition, the local inertial frame follows the North-East-Down (NED) reference, where x_I axis pointing North, y_I axis pointing East and z_I axis pointing Down, as displayed in Figure 18. Furthermore, the wind frame has its origin in the centre of mass of the UAV. The x_W axis is defined along the direction of the UAV ground airspeed vector, the y_W axis is normal to the x_W axis, parallel to the x_I-y_I plane and points to the right of the observer that is moving with the path, the z_W axis points down according to the NED reference.

Equation (3.5) corresponds to the control law proposed by Oliveira & Encarnação (2013) and implemented in this thesis.

$$\dot{\psi} = (-g_1\tilde{\psi} + k\,\dot{s} - g_2\,y_1(v_x \sin\psi_{sf} - v_y \cos\psi_{sf})\frac{1-\cos\tilde{\psi}}{\tilde{\psi}} - g_2 V_t y_1 \cos\bar{\psi}_d \frac{\sin\tilde{\psi}}{\tilde{\psi}}). \qquad (3.5)$$

The control law variables can be divided into five major parts: path geometry, UAV data, tracking errors, controller parameters and parameters of the space error that mathematically characterize the problem.

The first part, regarding path geometry, includes variables k and ψ_{sf}, representing the path curvature and the yaw angles of the Serret–Frenet frame, respectively. Variable $\dot{s}$ is the velocity vector of the UAV along the path, written in the referential that parameterizes the path and V_t is the UAV ground speed, as displayed in Figure 18. Variable y_1 represents the linear distance between the UAV and the path to be followed, and $\tilde{\psi}$ refers to orientation tracking

errors, as depicted in Figure 18. The controller parameters g_1 and g_2 were set to 0.1 and 0.00012, respectively. Finally, ψ_d and $\dot{\psi}$ are the parameters that mathematically characterize the problem.

To determine the bank reference for the outer loop controller, we used the coordinated turn relation (McLean, 1990), where U_0 representes the aircraft airspeed and g is the gravitational acceleration, as shown in

$$\Phi_{cmd} = \arctan\left(\frac{\dot{\psi}\, U_0}{g}\right). \tag{3.6}$$

Further details about the control law can be found in Oliveira & Encarnação (2013) and Oliveira et al. (2016) papers.

Guidance and Control Module Validation

One of the first blocks implemented in this architecture was the MPF adopted from Oliveira & Encarnação (2013) study. Since this control law takes as input the position, velocity, acceleration, angular acceleration, and orientation of the target, the first step to validate the implementation of this law was to develop a dummy target whose coordinates were artificially generated by the Target Estimate node (refer to Figure 19 and equations 3.15).

After implementing this target, the goal was to conduct several simulations to verify that the UAV was following the target accurately. Moreover, the fictitious target's coordinates are sent to the Guidance and Control node through a ROS node with a customized target message, as displayed in Figure 19.

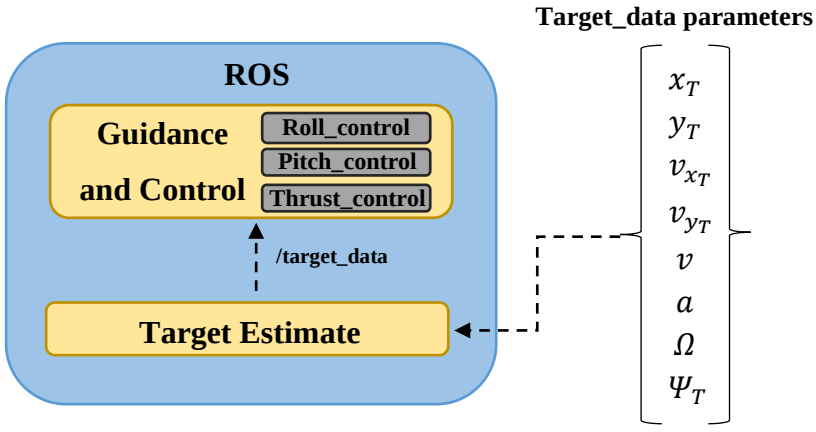


Figure 19. Architecture for the simulation of MPF with fictitious target.

Since the fictitious target had to generate several random parameters, a script had to be created to generate these values. Equations (3.5) present the considered target kinematics that were used to provide the Guidance and Control node with the target telemetry

$$x_T = \int v_{x_T}\, dt$$

$$y_T = \int v_{y_T}\, dt$$

$$v = \int a\, dt$$

$$v_{x_T} = v\ cos\ (\Psi_T) \tag{3.15}$$

$$v_{y_T} = v\ sen\ (\Psi_T)$$

$$\Omega = 0{,}03\ cos(0{,}03\ )$$

$$a = 0{,}011\ sen(0{,}015\ )$$

$$\Psi_T = \int \Omega\, dt$$

where,

x_T is the position in the Cartesian axis that corresponds to north in NED axis

y_T is the position in the Cartesian axis that corresponds to East in NED axis

v_{x_T} is the velocity in the north-axis

v_{y_T} is the velocity in the east-axis

v is the linear velocity of the target

a is the acceleration

Ω is the angular acceleration

Ψ_T is the target's orientation.

The simulation was conducted with the aircraft taking off and climbing to 100 m at 15 m/s airspeed. Then, the UAV started to loiter counterclockwise with a radius of 150 m, and the fictitious target was launched.

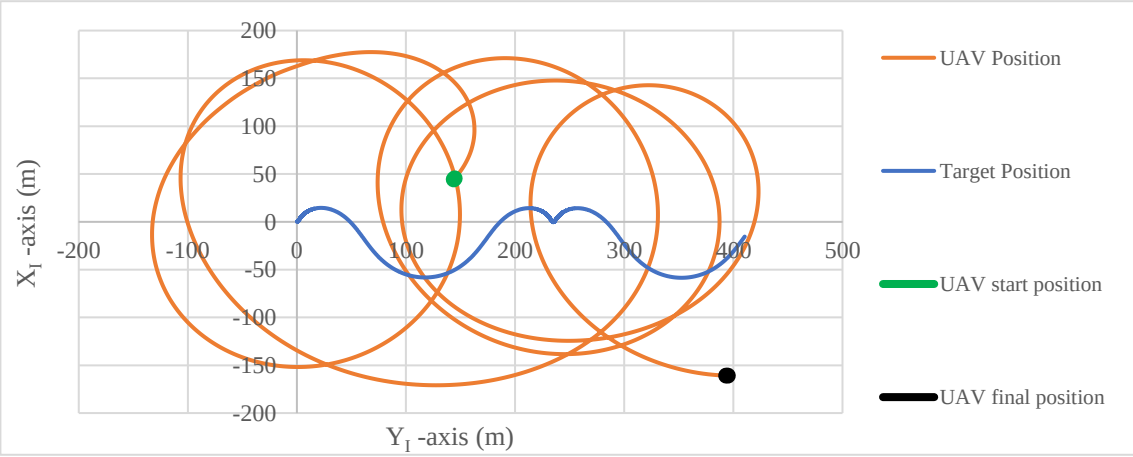


Figure 20. 2-dimensional display of the UAV and target during the simulation.

Figure 20 displays the position of the UAV and fictitious target during the whole simulation. The orange line corresponds to the UAV position, and the blue line represents the target's motion corresponding to the sinusoidal trajectory. It shows an expectable behaviour with various turns around the target with an average lateral error of 1.79 m, a maximum of 55.67 m and a minimum of -44.36 m.

Figure 21 displays the lateral error throughout the simulation. In this figure, the lateral error oscillates with low amplitude around zero throughout the simulation.

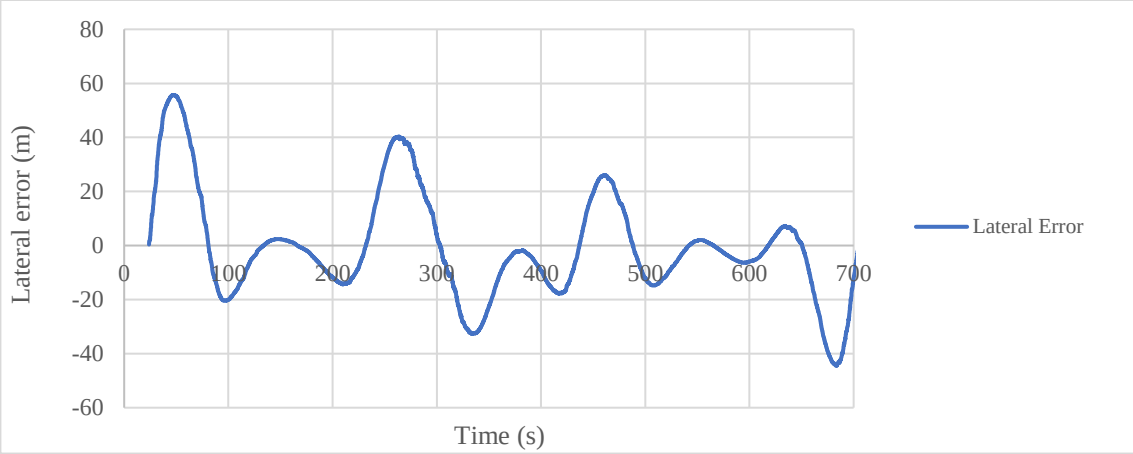


Figure 21. Lateral error.

The evolution of the variables concerning the target motion is shown in Figure 22. More detailed information regarding the simulation is presented in Table 2. As the UAV can follow the fictitious target and the lateral error decreases over time, we can consider that this module is validated, given the results presented. Furthermore, the results presented are consistent with the Oliveira & Encarnação (2013) paper.

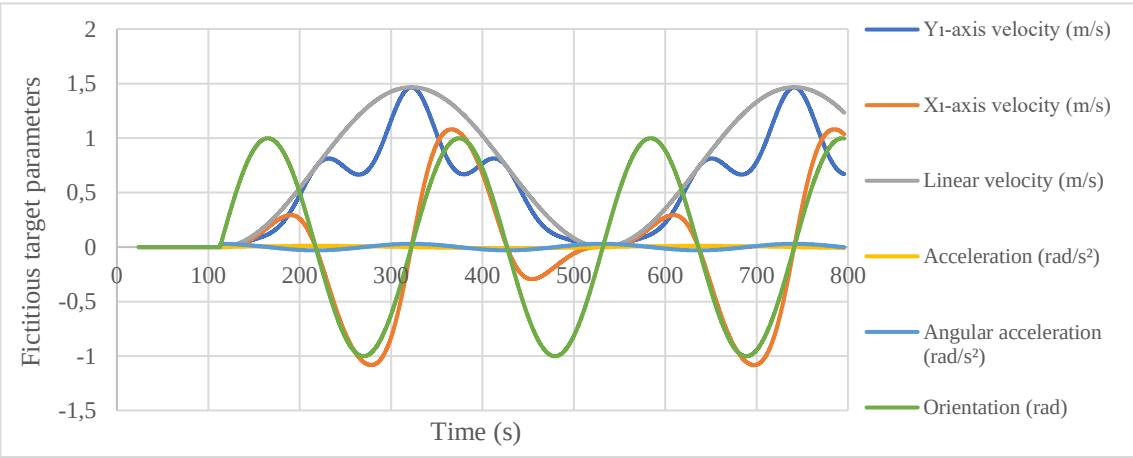


Figure 22. Fictitious target Parameters.

Table 2. Detailed information regarding the fictitious target parameters and UAV lateral error.

		Results		
		Maximum	**Minimum**	**Average**
Fictitious target parameters	Velocity east-axis (m/s)	1.466	0	0.53
	Velocity north-axis (m/s)	1.08	-1.08	-0.02
	Linear velocity (m/s)	1.47	0	0.65
	Acceleration (m/s²)	0.011	-0.011	0.002
	Angular Acceleration (rad/s²)	0.03	-0.03	0.0013
	Angular orientation (rad)	0.99	-1	0.05
Lateral error	Lateral Error (m)	55.67	-44.36	1.79

3.4.2 Target Detector

Once the UAV is equipped with a camera observing the ground, the next step is to implement a method that allows the UAV to detect and extract relevant information from a target to be used by the Guidance and Control node.

Since the initial aim is to validate the Control and Guidance module, a simple colour filter was implemented to detect a red block. The OpenCV library was used for image processing due to its extensive list of implemented algorithms and ease of use (OpenCV team, 2022). The first step was to import the image through a UDP port from the Gazebo simulator. The image has three channels RGB, each channel with 256 levels and a resolution of 8 bits per pixel.

Since the target was a red object, the colour upper and lower boundaries in the RGB colour space needed to be defined. In this research, the boundaries [Blue,Green,Red] used for the red colour were: lower bound=[0,0,100] and upper bound=[50,50,255]. These boundaries mark the lightest red and darkest red colour to be detected.

Then Open CV's image masking was applied to highlight only the portion of the image of interest, based on the colour boundaries mentioned above. Afterwards, a mask was implemented to show only the red object in any image. After applying the mask, the red block becomes white and the rest of the image becomes black (Rosebrock, 2021).

Finally, this filtered image intersects with the original image obtaining the red object detected. Subsequently, the OpenCV counter function applied an outline to the object. Next, through the boundingRect function (OpenCV team, 2022), we determined the height, width and coordinates of the top-left vertex of the red block. Then the width and height were summed to the corresponding coordinate and determined the centre of the rectangle. In the last phase, the position of the centre of the red block was written in a ROS message to publish these two coordinates in a ROS topic.

Target Detector Validation

In order to validate the Target Detector module, a continuous loiter was performed by the UAV with a red object standing still on the ground. The simulation purpose was to verify that the aircraft could assertively detect and estimate a ground target's position.

The simulation verified that the aircraft camera could detect the red object and indicate the pixel coordinates of its centre of mass in the camera frame. The sensor width and height used were 640 pixels, and the focal distance was 320 pixels.

Figure 23 displays, on the left side, the red object detected in the Gazebo simulator with a green cross representing its centre of mass which corresponds to the pixel coordinates to be provided to the Target Geolocation node (described in the sequel). This image helped confirming if the object was being detected properly and if it needed any modification in the thresholds that specifies the red colour. On the right side of Figure 23, a binary mask indicated the presence of the desired object and verified if another object was being detected.

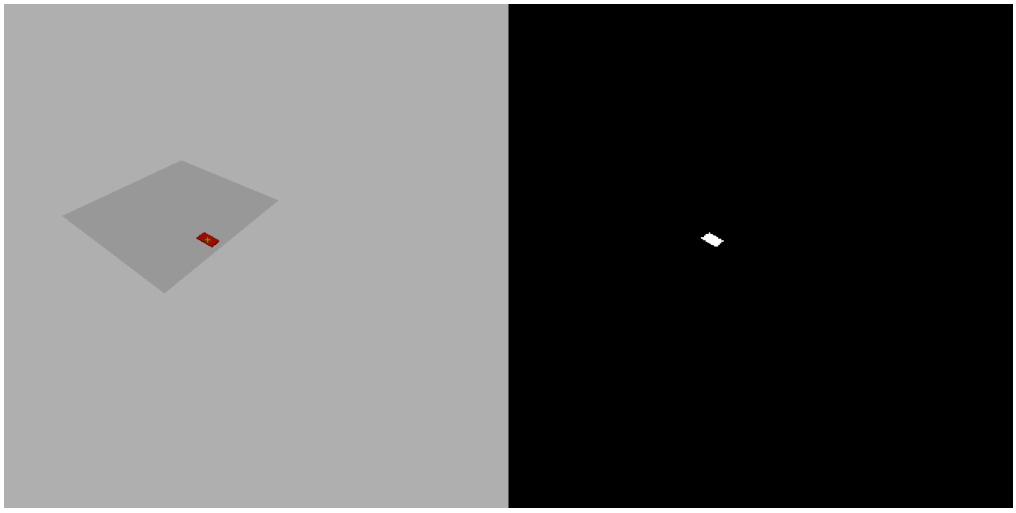


Figure 23. Picture from the onboard camera with the object detection in colors and the binary mask indicating the position of the detected object.

3.4.3 Target Geolocation

Since a low-cost small UAV with a camera embedded is being used, the solution found for navigation, guidance, and control purposes was to use the geolocation algorithm from Barber et al. (2006). This method was motivated by how birds and insects use their vision as primary guidance sensors.

Generally, the geolocation algorithm computes the position of a ground target through its location and motion streamed in the video sequence. This approach assumes that another algorithm is being used to track the features of the target in the video. Moreover, the geolocation

approach considers that the object moves around a flat terrain (flat-earth assumption), and the UAV can calculate its location and other state variables. The image plane is determined by the target movement and the translational and rotational motion of the aircraft (R. W. Beard & McLain, 2012).

The method is based on five different frames, the inertial frame, the gimbal frame, the camera frame, the body frame and the vehicle frame, as depicted in Figures 24 and 25. The inertial frame is a fixed frame denoted by (X_I, Y_I, Z_I), which corresponds to the NED notation and corresponds to the local inertial frame addressed in the MPF algorithm with coordinates (x_I, y_I, z_I), which are displayed in Figure 18. The vehicle frame has its origin at the vehicle centre of mass and is oriented identically to the local inertial frame (X_v, Y_v, Z_v). The body frame, as well as the vehicle frame, has its origin in the vehicle's centre of mass. However, the orientation is different, the X_b points to the nose, Y_b points to the right-wing and Z_b pointing out to the belly. Note that the body frame from Figure 24 and the wind frame presented in Figure 18 in this thesis are coincident since the wind is neglected, and the angles of attack and slip in relation to the air mass are also negligible (Oliveira et al., 2016b).

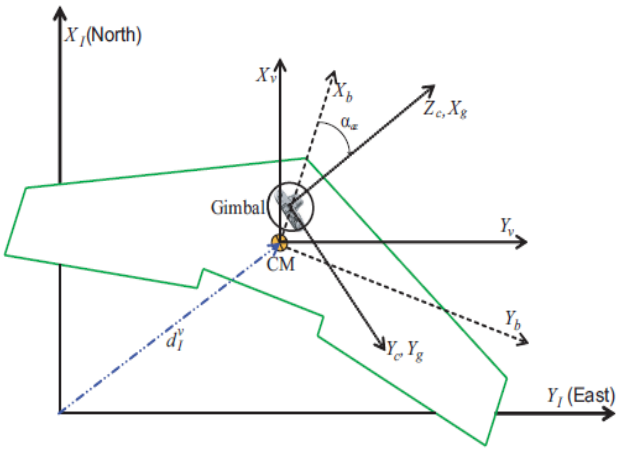


Figure 24. Longitudinal perspective of the coordinate frame (Retrieved from Barber et al. (2006)).

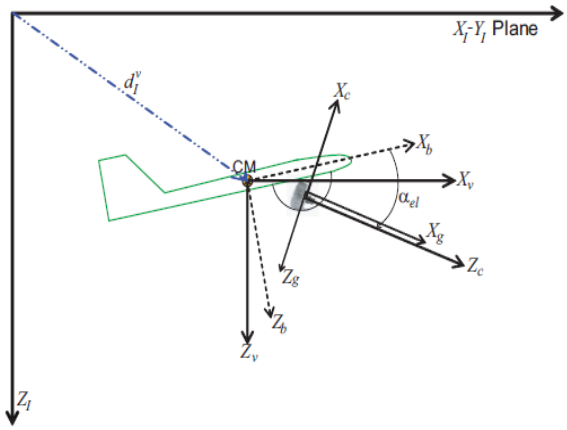


Figure 25. Lateral perspective of the coordinate's frames (Retrieved from Barber et al. (2006)).

As displayed in Figures 24 and 25, the gimbal frame originates at the gimble rotation centre and is represented by X_g pointing out along the optical axis, Y_g pointing out to the right in the image plane and Z_g pointing down in the image plane. Finally, the camera frame denoted by (X_c, Y_c, Z_c) has its origin in the optical centre. The X_c points towards the upper part of the image, Y_c points to the right of the image plane and Z_c is aligned with the optical axis (Barber et al., 2006).

Given the previously presented notation, the position of target in the inertial frame can be computed by the target's pixel coordinates in the camera frame through equation (Barber et al., 2006)

$$p_{obj}^{i} = \begin{pmatrix} p_n \\ p_e \\ p_d \end{pmatrix} + h\frac{\mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \tilde{\ell}^c}{\mathrm{k}^i \mathcal{R}_b^i \mathcal{R}_g^b \mathcal{R}_c^g \tilde{\ell}^c}, \tag{3.7}$$

where,

$\mathcal{R}_g^b$ is the rotation matrix from the gimbal to the body

$\mathcal{R}_c^g$ is the rotation from camera frame to gimbal frame

$\mathcal{R}_b^i$ is the rotation from body to inertial

h is the UAV altitude

$\begin{pmatrix} p_n \\ p_e \\ p_d \end{pmatrix}$ are the UAV coordinates

k^i represents the Z axis of the inertial frame

$\tilde{\ell}^c$ represents the normalized target's line-of-sight vector from the camera frame.

Target Geolocation Validation with the Target at a Constant Position

Once proven that the object was detected effectively and could publish the pixel coordinates, several simulations were conducted, feeding those target's pixel coordinates to the Target Geolocation node to verify its estimated position in the inertial frame. In order to achieve as accurate validations as possible, the following parameters were changed between simulations: object position, camera orientation, and aircraft loiter turn direction. As a result, the target position was simulated at four different places, each corresponding to a different quadrant considering the Cartesian Axis. In addition, the aircraft camera orientation was changed given the direction of the turn, which altered the camera pan and the desired elevation (tilt).

One out of the four simulations results is presented in detail in the following. In this test, the UAV flew at 100 m, 15 m/s, loitering clockwise, with a pan equal to $\alpha_{az} = 68.75^{\circ}$ and a tilt of $\alpha_{el} = -44.12^{\circ}$,as displayed in Figure 26. The camera frame (provided by the gazebo simulator) had a height and width of 640 pixels with a focal length of 320 pixels. The object was placed in the fourth quadrant with the following coordinates, $X_I = -10\ m$ and $Y_I = 50\ m$.

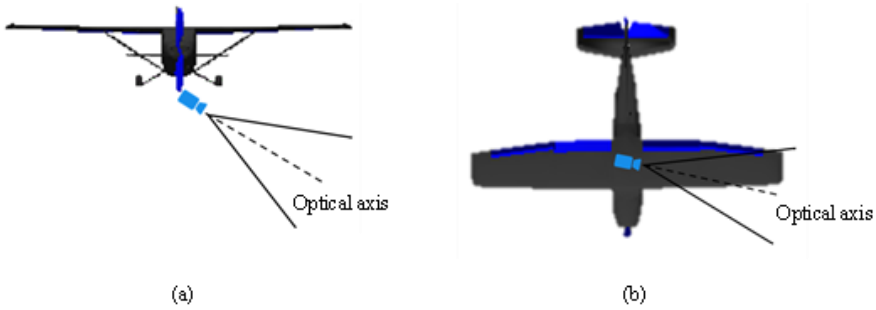


Figure 27. Front (a) and top (b) views of the camera's frame in relation to the aircraft.

In Figure 27, a 2-Dimensional view of the simulation is displayed. The target position had an average error of 14.04 m, but regarding the blue line in the graph, the target position is around the real position. Therefore, it is possible to confirm the accuracy of the implemented method, since the average $Y_I = 48.87\ m$ and the average $X_I = 14.03\ m$.

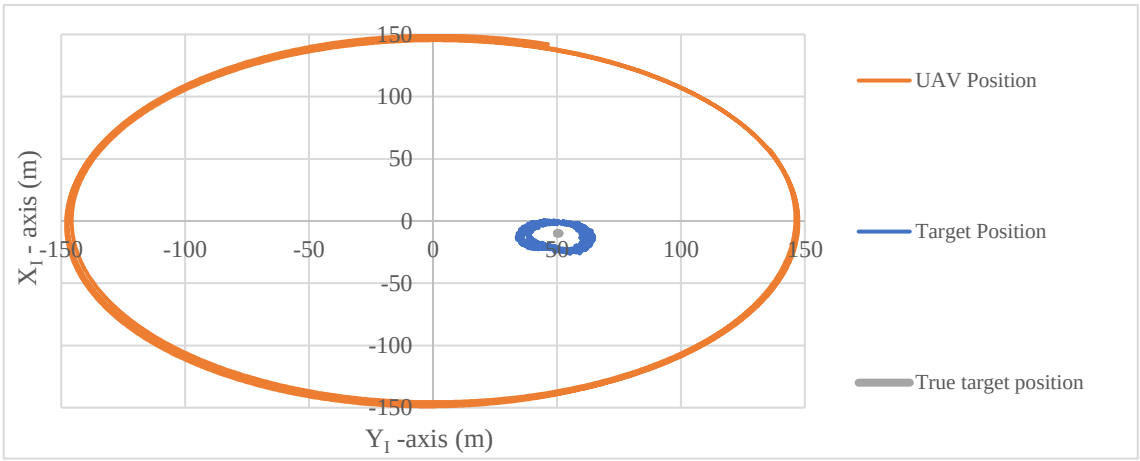


Figure 26. 2-dimensional display of the UAV and target during the simulation.

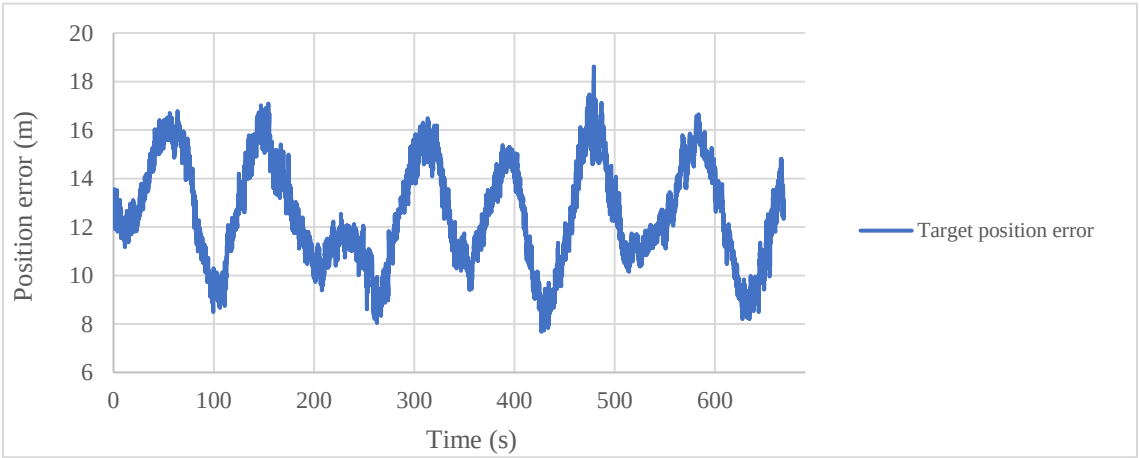


Figure 28. Target position error.

Figure 28 shows the position error over simulation time with a maximum error of 25.60 m and a minimum error of 7.67 m. In view of the results for the four simulations it is possible

to validate the Geolocation module since the results are within the desirable error bound (detailed information regarding the four simulations is presented in Table 3).

Table 3. Detailed information regarding the Geolocation validation in four simulations with the target standing at a constant position.

Right Turn

Pan	Tilt	Target Position in Gazebo World (m)	Geolocation average position error (m)	Geolocation position error (m)	
				Maximum	Minimum
68.75°	-44.12°	X_I=-10 Y_I=50	14.04	25.60	7.67
90°	-45°	X_I=60 Y_I=20	12.38	25.38	3.89

Left Turn

Pan	Tilt	Target Position in Gazebo World (m)	Geolocation average position error (m)	Geolocation position error (m)	
				Maximum	Minimum
-90°	-45°	X_I=-30 Y_I=-10	13.09	23.63	5.58
-90°	-45°	X_I=20 Y_I=-70	17.69	30.97	5.25

Target Geolocation Validation with the Target Moving

Since the ultimate goal of this research is to have the UAV locating a moving car, it is important to validate the geolocation algorithm with a moving target. In order to validate this movement condition, an actor plugin from Gazebo was used to give movement to the red block. This plugin works through a predefined set of waypoints and time between waypoints to build a closed path.

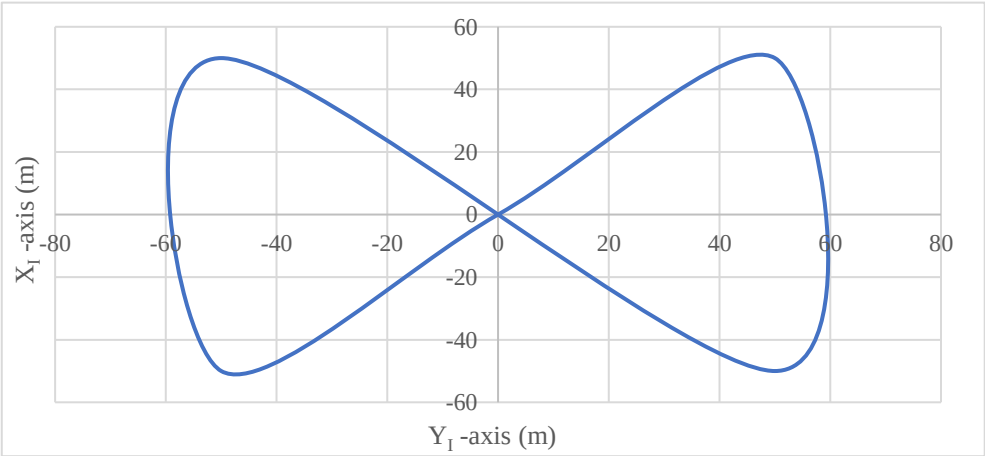


Figure 29. 2-Dimensional view of the target's path.

The target's path used in this simulation, shown in Figure 29, was divided into five sections. The target started at position $X_I = 0\ m$ and $Y_I = 0\ m$ and then moved along the four vertices. In the first section of the track, the object moved to the point $X_I = 50m$ and $Y_I = 50\ m$. Then, in the second section, the target maneuvered along the north axis to $Y_I = 50\ m$ and $X_I = -50\ m$. After that, it drew a diagonal through the graph to $X_I = 50\ m$ and $Y_I = -50\ m$

and completed the third section. In the fourth branch, it went down to the third quadrant and coordinates of to $X_I = -50\ m$ and $Y_I = -50\ m$, and finally headed to the starting point, thus completing the last part of the path (detailed information about the target's path is displayed in Table D-1 from Annex D).

In order to make the simulation even more realistic, the velocity of the object was changed by modifying the time that the target had to travel each part of the path. In the first, second and fifth sections, the target moved at 2.36 m/s, and in the remaining two parts, the target moved at 3.33 meters/second.

The simulation was performed with the UAV loitering clockwise with the centre in $X_I = 0$ and $Y_I = 0$, at the height of 100 m and 15 m/s. The detailed information regarding the camera, UAV and target settings used in this simulation is depicted in Table D-2 from Annex D. The tilt used in the simulation was computed based on the aircraft's average roll angle on 150 m loiter. In order to maintain the 45◦ angle between the horizontal plane of the aircraft and the camera's optical axis, the tilt had to be -36.31◦ considering that the UAV's average roll angle was 8.69◦.

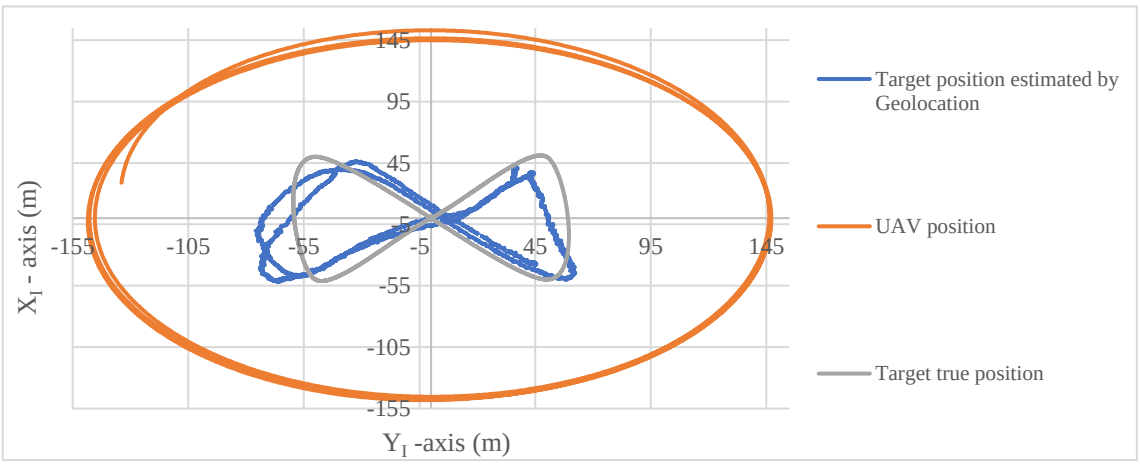


Figure 30. 2-Dimensional view of the UAV, target estimated position and target's true position.

Figure 30 displays the 2-Dimensional view of the whole simulation, where the orange line represents the loitering from the UAV, the grey line corresponds to the real target path. Lastly, the blue line is the target position measured by the Geolocation algorithm.

Based on the blue and grey lines, it can be stated that globally, the geolocation estimate is close to the actual path of the target. On average, the computed position error is 11.64 m, with a standard deviation of 2.81 m. Furthermore, from Figure 31, the estimate of the target's position (solid line) follows the real path (discontinuous line) uniformly throughout the simulation. Therefore, according to the results obtained from this simulation, it is possible to

validate the geolocation module (detailed information about the Geolocation position measurement is presented in Table 4).

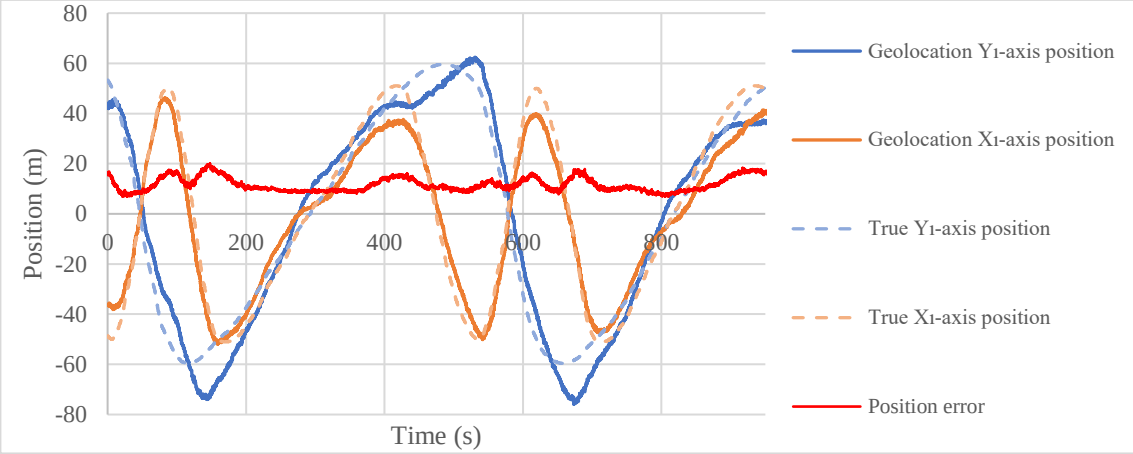


Figure 31. Comparison between the east and north position measured by the Geolocation algorithm and the actual position of the target.

Table 4. Results from the validation of Geolocation with the target moving.

Results	
Average position error (m)	11.64
Standard Deviation (m)	2.81
Maximum position error (m)	19.99
Minimum position error (m)	6.86

3.4.4 Target Estimate

Due to the specific nature of UAVs, several errors affect the geolocation estimation, with some high-frequency peaks. In addition, the Geolocation only calculates the position, and the MPF needs additional parameters as input (including, among others, velocity and heading of the target). For these two reasons, a Kalman Filter was implemented.

The Kalman Filter is an algorithm that "*provides estimates of unknown variables given the measurements observed over time*" (Y. Kim & Bang, 2019). This filter is used to estimate states through linear dynamical systems. In simpler words, the goal of the Kalman filter is to provide an estimative of x_k at a time k, knowing the first estimate of x_0 and the other estimations until x_k, and the information described by different matrices *P*, *F*, *Q*, *R* and *H*.

The system state or x_k corresponds to the parameters that the algorithm wants to estimate; in this dissertation, the system state is $X_{state} = [x, y, v_x, v_y, a_x, a_y]$. In addition to provide an estimate of the system's state, the Kalman filter also tracks an estimate of the uncertainty associated with that state. That is related to the fact that the position, velocity and acceleration estimates are never perfectly known (since they depend, among others on the sensor

measurements, which are affected by noise). This uncertainty can be represented by a matrix known as the state covariance matrix P. In the implemented Kalman Filter, the covariance matrix was the following one

$$P = \begin{bmatrix} 0.01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix}. \tag{3.8}$$

Matrix (3.8) is the first estimate of the covariance matrix used in this study. The covariance matrix (3.8) uses small values since the system's first estimates the system state based on the target's first measurement position.

The Dynamic model or matrix *F* describes the relationship between the current state (input) and the next state, corresponding to the target parameters in the next interval (output). This matrix is based on the physical context of the problem. In this research, since the Geolocation only measures the target's position, the goal is to estimate the velocity and acceleration based on a dynamic model. The equation that describes an object's dynamics motion with constant acceleration is Newton's motion equation and displayed as follows

$$\begin{gathered} x = x_0 + v_0\, dt + \frac{1}{2}\, a\, dt^2 \\ v = v_0 + a\, dt \\ a = a. \end{gathered} \tag{3.9}$$

The simulation is in three dimensions, but since it is considered that the target is lying on a flat ground, it discarded the motions on the z-axis, so the system of equations is

$$\begin{cases} x = x_0 + v_{x0}\, dt + \frac{1}{2}\, a\, dt^2 \\ y = y_0 + v_{y0}\, dt + \frac{1}{2}\, a\, dt^2 \\ v_x = v_{x0} + a_x\, dt \\ v_y = v_{y0} + a_y\, dt \\ a_x = a_x \\ a_y = a_y \end{cases}. \tag{3.10}$$

The Dynamic model can be described as the following matrix (3.11), assuming constant acceleration

$$F = \begin{bmatrix} 1 & dt & \frac{dt^2}{2} & 0 & 0 & 0 \\ 0 & 1 & dt & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & dt & \frac{dt^2}{2} \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.11)$$

The process noise covariance matrix *Q* is the error related to external factors that accounts for possible miss-match between the considered target kinematics and its actual motion. In the implemented Kalman Filter, the Q matrix (3.12) was the following one

$$Q = \begin{bmatrix} 0.0001 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.00001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.00001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.00001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.00001 \end{bmatrix}. \quad (3.12)$$

Since Geolocation estimates the position state, its confidence (10^{-4}) is slightly higher than the other states (10^{-5}).

The measurement noise matrix *R* represents the sum of errors in the measurements from the system. Therefore, this matrix indicates the level of thrust of the measured values from the sensor. Similar to the matrix (3.8), the higher the sensor's accuracy, the lower the values entered in the matrix, and vice versa. The next Matrix (3.13) represents the matrix *R* used in this research

$$R = \begin{bmatrix} 30 & 0 \\ 0 & 30 \end{bmatrix}. \quad (3.13)$$

The uncertainty that best fits this problem is 30 because, through several simulations, this value was the one that most closely matched the filter estimate to the actual target parameters.

Finally, the matrix H or observation matrix represents the measured variables and relates to the state vector. Since the only variable measured by the system is the x and y position, the following matrix (3.14) has a unit value at the position of these same variables

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \quad (3.14)$$

The Kalman filter comprises two stages, the prediction process, and the measurement process, as shown in Figure 32. As the name suggests, the first stage predicts the next state of the system. The filter makes the first prediction through the system state estimate, using the

first measurement received by the system, and over the system model matrix *F* and matrix *Q* (Lim et al., 2016).

The next step is when the Kalman filter computes the Kalman Gain for each incoming measurement and determines how the input measurements influence the system state estimate. The Kalman Gain is used by the system to estimate the system state and error covariance matrix for the time of the input measurement (Welch & Bishop, 1995).

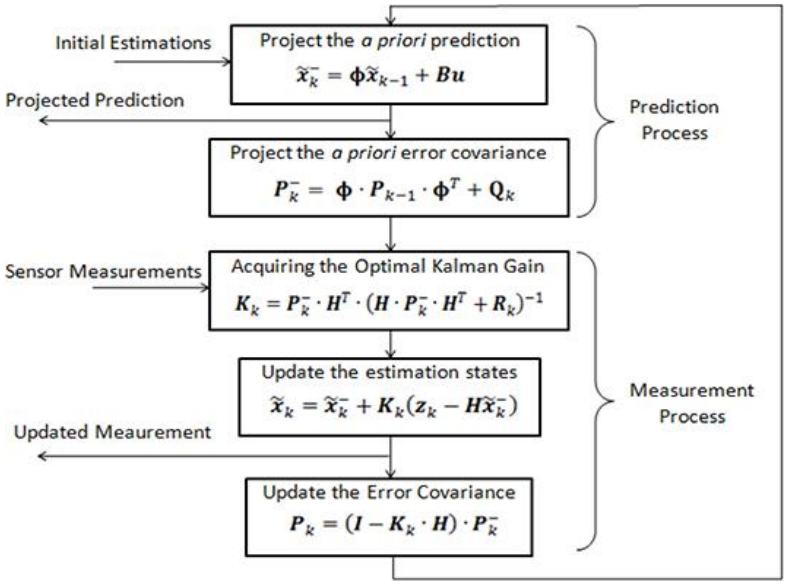


Figure 32. Block diagram of the Kalman filtering algorithm (Retrieved from Lim et al. (2016)).

Since the filter can only estimate the parameters of position, velocity and acceleration, it was necessary to use the equation (3.16) and (3.17) to determine the value of orientation and angular acceleration. Therefore, the orientation of the object is calculated using the following equation

$$\Psi = \operatorname{atan}\left(\frac{v_{north}}{v_{east}}\right). \tag{3.16}$$

The angular acceleration is deduced from Ψ by the following equation

$$\Omega = \tanh\left(\frac{d\,\Psi}{dt}\right). \tag{3.17}$$

Target Estimate Validation

Since the data coming from the Target Geolocation node depends on the "instantaneous" reading of the color filtering system, this measurement is sometimes too noisy to be sent directly to the Guidance and Control node. Additionally, the Target Geolocation node only provides estimated target position information. Since the Kalman Filter is one of the most common methods of estimating parameters not directly observable/measured by a system's sensors, it is

used in this research. In the case of this thesis, it is intended to estimate the velocity and acceleration based on the measured position. After reviewing how the Kalman filter works and subsequently adapting the matrices to the problem at hand, the filter's validation was started.

The validation of this filter was performed with the target position measurements from the Geolocation. Since the Geolocation measures position, it was possible to feed the filter with the position and estimate the other two parameters with the filter's estimation, as Figure 33 shows.

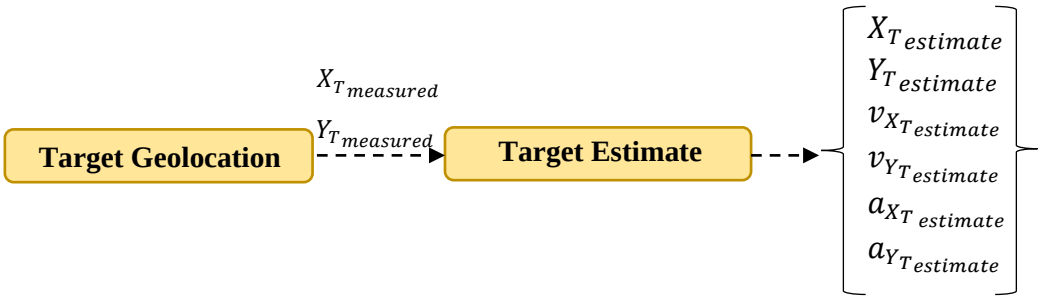


Figure 33. Architecture for the simulation of target measured by the Geolocation with Kalman Filter estimation.

Kalman Filter Validation with the Object Standing at a Constant Position

The first simulation was conducted with the target standing on the ground at a constant position. The detailed information regarding the camera, UAV and target settings used in this simulation is depicted in Table E-1 from Annex E.

Figure 34 displays a 2-Dimensional view of the simulation where both Kalman Filter and Geolocation estimates are close to the target's true position.

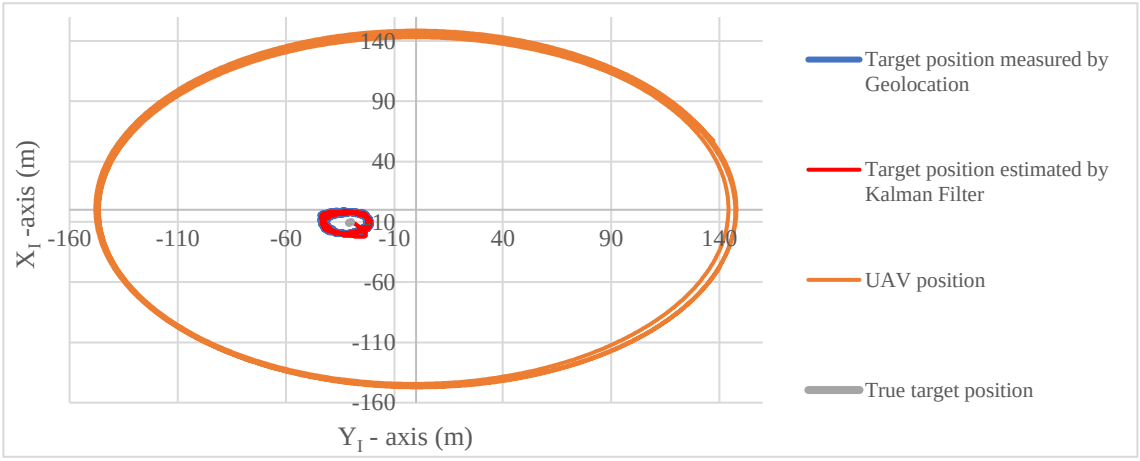


Figure 34. 2-dimensional display of the UAV, target measured by Kalman Filter, Geolocation and true location of the target during the simulation.

Apart from the similarity in the position, the main difference between the filter and Geolocation is exposed in Figure 35. It is possible to notice that the Kalman filter measurements

are less noisy than the Geolocation data. Moreover, the output from the filter eliminates the high-frequency peaks from the Geolocation.

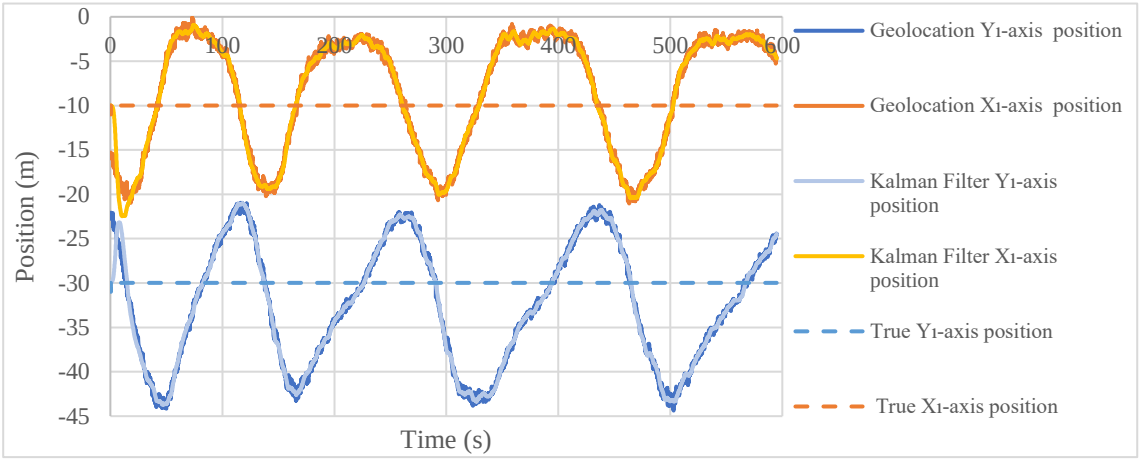


Figure 35. Target position estimated by Geolocation and Kalman Filter and true target position.

The additional parameters estimated by the Kalman filter are displayed in Figure 36. Both estimated velocities slightly oscillate in around zero, with averages near zero.

In addition, the acceleration also reached values close to zero. It is possible to get a more detailed information of the data estimated by the Kalman Filter through Table 5.

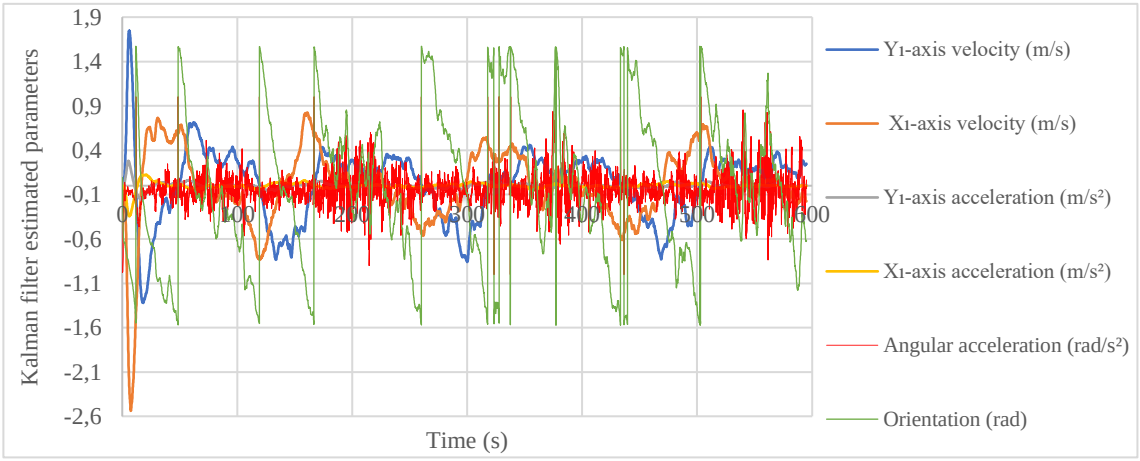


Figure 36. Target parameters estimated by Kalman Filter during the simulation.

Table 5. Parameters estimated by the Kalman Filter.

Kalman Filter estimated parameters

	Average	**Maximum**	**Minimun**
Velocity X_I-axis (m/s)	-0.0016	0.8255	-2.536
Velocity Y_I-axis (m/s)	0.0168	1.7477	-1.323
Acceleration X_I-axis (m/s ²)	-0.0019	0.1238	-0.344
Acceleration Y_I-axis (m/s ²)	0.0017	0.2806	-0.155
Target Orientation (rad)	-0.0450	1.5703	-1.57
Angular acceleration (rad/s²)	-0.0412	1	-1

Kalman Filter Validation with a Moving Object

In the simulation, to validate the Kalman filter with the object moving, we used the previously defined path that validated the geolocation algorithm, represented in Figure 29. The detailed information regarding the camera, UAV and target settings used in this simulation is depicted in Table E-1 from Annex E.

The position estimated by the Kalman Filter is similar to the one measured from Geolocation, as shown in Figure 37 (detailed information about the position measured by Geolocation is displayed in Table 6). As it was possible to state in the previous section, the filter outputs are less noisy and reduce high-frequency spikes, as shown in Figure 38.

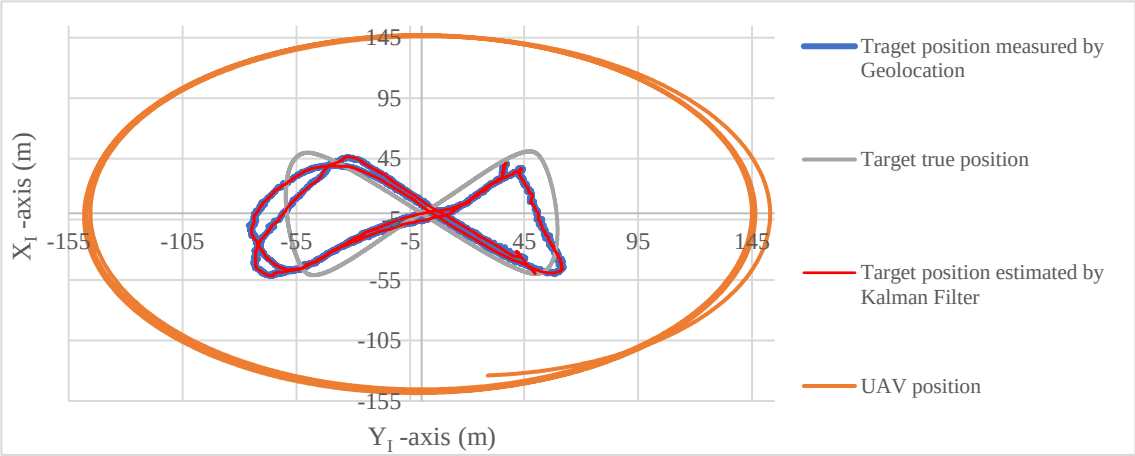


Figure 37. 2-dimensional display of the UAV, target measured by Kalman Filter, Geolocation and true location of the target during the simulation.

Table 6. Results from Kalman Filter position estimation.

Results

Average position error (m)	11.619
Standard Deviation (m)	2.9000
Maximum position error (m)	19.253
Minimum position error (m)	2.6833

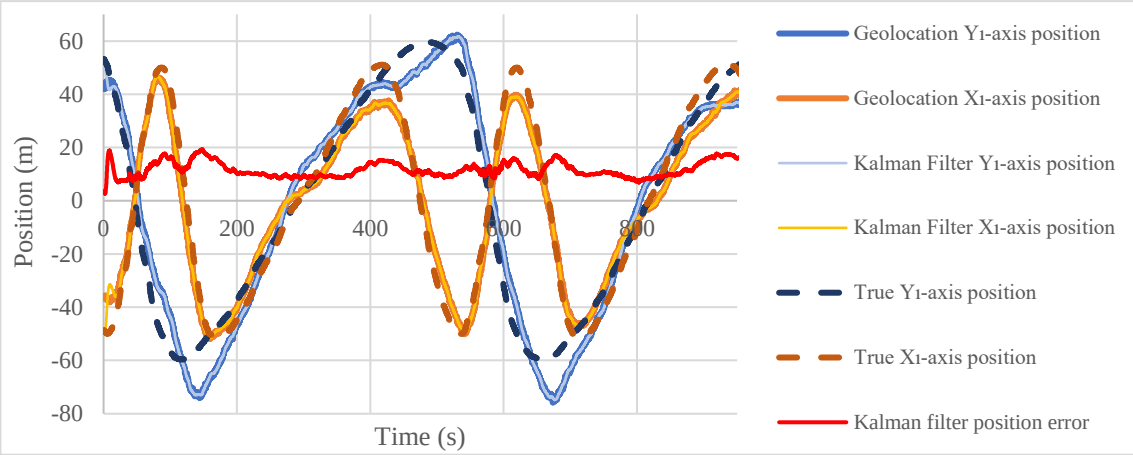


Figure 38. East and north positions from Kalman Filter, Geolocation and true target position.

Figure 39 displays the east and north velocities estimated by the Kalman Filter and measured from the Gazebo telemetry. Since the red object is generated by a Gazebo plugin, the telemetry for that target is only available in a specific simulator window. Given this setback, it was necessary to create a new ROS topic to access the target (true) position. Afterwards, the velocity was calculated for each instant of time in both axis.

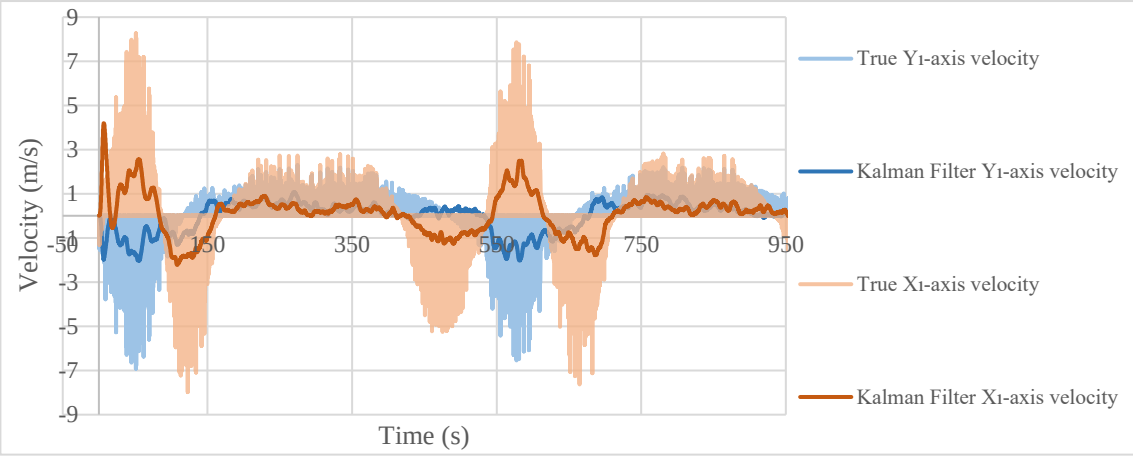


Figure 39. Comparison between the target velocity estimated by the Kalman Filter and true velocity.

It is possible to verify that the velocity coming from the Gazebo's telemetry is noisy due to a generation issue of the target's position. Since the Gazebo's world is coarsely pixelated, the object moves between consecutive (discrete) positions instantaneously and not continuously, causing a noisy output. Even so, it is possible to confirm that the velocity estimated by the Kalman filter follows the low-frequency component of the target's actual speed. In terms of acceleration, it was not calculated for later comparison to the Kalman Filter outputs due to the noisy measurements. Figure 40 displays the acceleration, angular acceleration, and orientation resulting from Kalman Filter. The error from the velocity estimated by the Kalman filter are displayed in Table 7, among the other parameters.

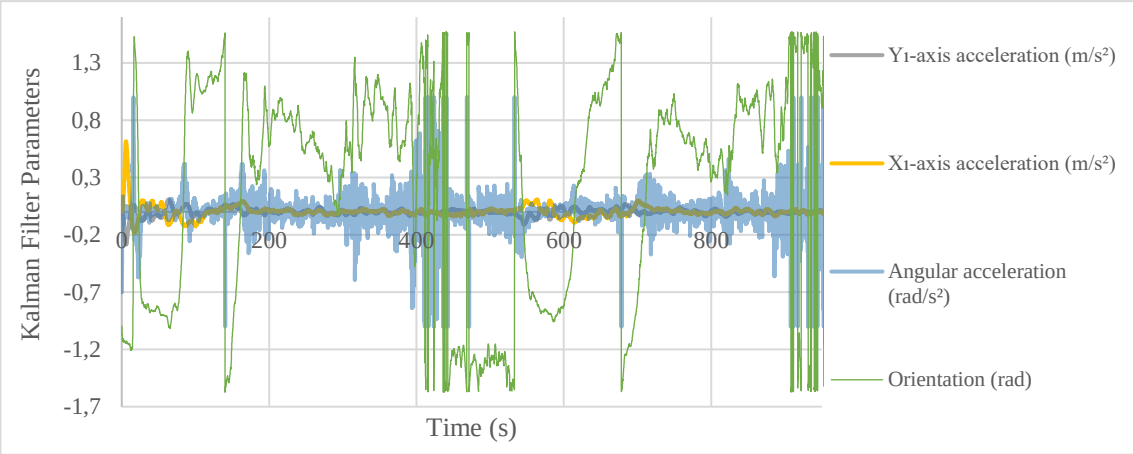


Figure 40. Acceleration, angular acceleration and orientation from Kalman Filter estimate.

Table 7. Results from Kalman filter parameters estimation.

Results	
X_I axis velocity (m/s)	
Average error (m)	0.0076
Maximum estimated (m)	4.1947
Minimum estimated (m)	-2.234
Y_I axis velocity (m/s)	
Average error (m)	-0.0175
Maximum estimated (m)	1.0807
Minimum estimated (m)	-2.029
X_I axis acceleration (m/s²)	
Maximum estimated (m)	0.6163
Minimum estimated (m)	-0.188
Y_I axis acceleration (m/s²)	
Maximum estimated (m)	0.1084
Minimum estimated (m)	-0.286
Angular acceleration (rad/s²)	
Maximum estimated (m)	1
Minimum estimated (m)	-1
Orientation (rad)	
Maximum estimated (m)	1.5706
Minimum estimated (m)	-1.5711

3.5 Closed-loop Validation - Intermediate Results

One of the valences of the architecture used in this dissertation is the possibility of easily replacing blocks of the chain and thus changing the scope or adding other capabilities to the

system. However, to make these changes successfully, all the system modules have to be tested and validated individually to ensure that each block works as it is supposed to.

Full validation of the entire loop encompasses connecting all the blocks of the architecture shown in Figure 6 and verifying that the aircraft reacts in the correct and expected way by putting them together. In this simulation, the object is moving along a predefined path.

First, the UAV performs a loiter and start detecting the target. After detecting it, the UAV measures its position through Geolocation and estimate with Kalman Filter the remaining target variables. Finally, the position, velocity, acceleration, angular acceleration, and orientation parameters are fed to the Guidance and Control node. Then, the UAV should be able to continuously detect and track a moving target, by loitering around its estimated position. The detailed information regarding the camera, UAV and target settings used in this simulation is depicted in Table F-1 from Annex F.

The target's path used in this simulation is drawn in Figure 41, and the different sections of the track are specified in Table F-2 from Annes F. The course design was based on the path once implemented in geolocation validation, but in this case, it was divided into seven-parts. In order to move the object as similar as possible to a car, several changes in direction, speed and stops at a given point were implemented.

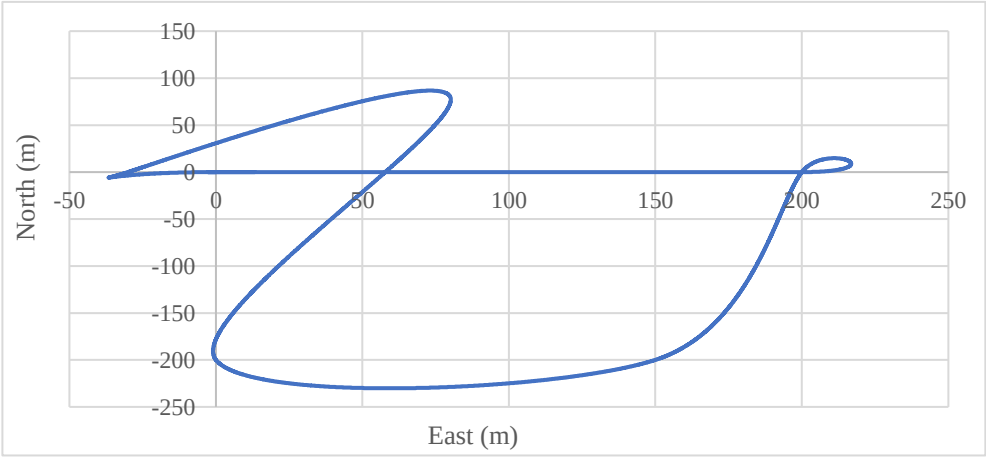


Figure 41. 2-Dimensional view of the target path.

After the target path was defined, the simulation followed. With the UAV flying at an altitude of 100 m and 15 m/s speed, the Target Geolocation and Target Estimate nodes were started, and the UAV began following the target. Figure 42 shows the 2-dimensional path taken by the object and the UAV. The figure shows an accurate estimate of the target position by the Kalman filter compared to the actual position of the object.

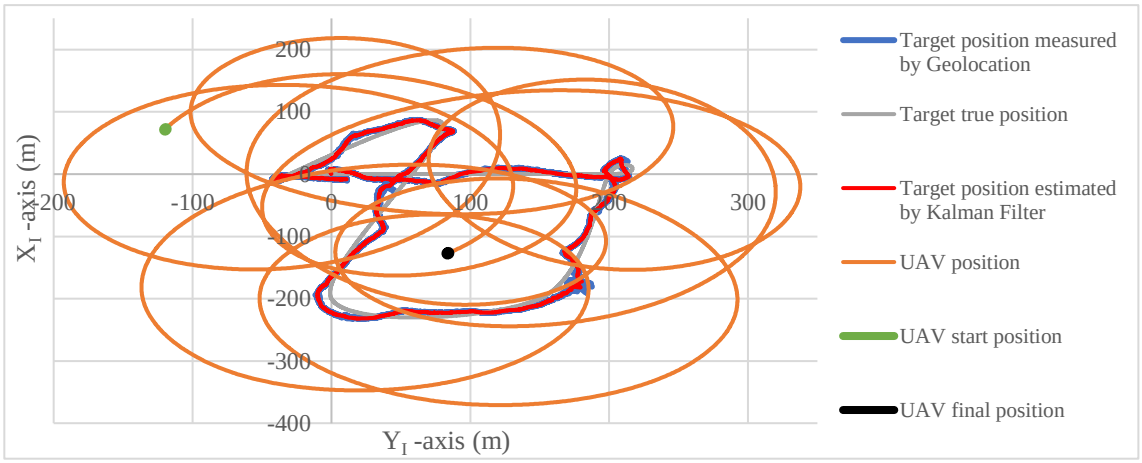


Figure 42. 2-Dimensional view of the simulation.

In conjunction with Figure 43, which shows the position discriminated by each axis, it is possible to verify the proximity between the real position and estimated measurements by the filter. Moreover, the average error of the filter position compared to the actual target position is 8.7953 m, with a standard deviation of 2.56 m (detailed information about the position estimated by Kalman Filter is displayed in Table 8).

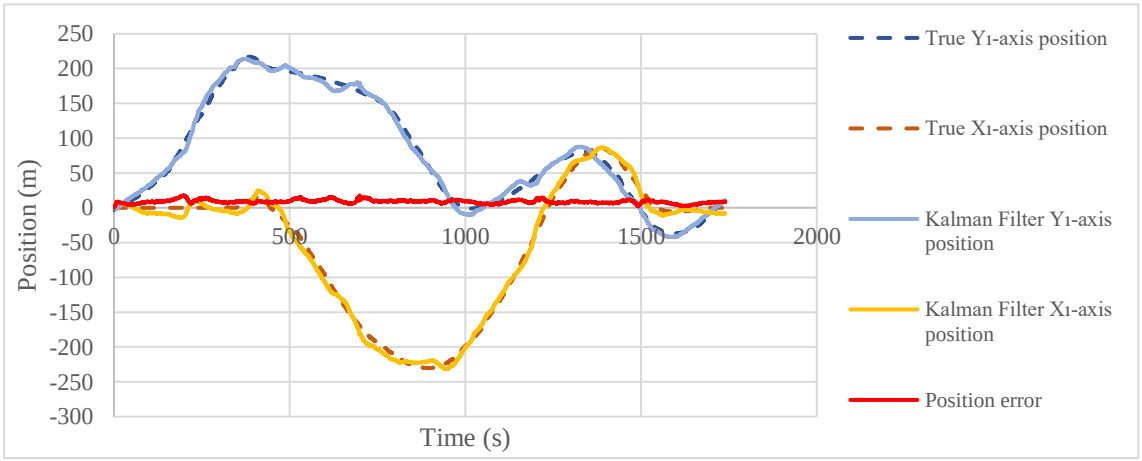


Figure 43. Target position estimated by Kalman Filter and true target position.

Table 8. Information regarding Kalman Filter target position.

Results from Kalman Filter

Average position error (m)	8.79
Standard Deviation (m)	2.56
Maximum position error (m)	17.46
Minimum position error (m)	0

Figure 44 displays the velocity estimated by the Kalman filter and the velocity calculated from the true target position given by Gazebo. For the same reasons stated in section 3.4.4, despite the calculated speed being noisy, the estimated velocity follows the course of the actual speed, with an average error on the east axis of 0.0039 m/s and the north axis of 0.0017 m/s.

So far, it has been possible to verify that the target location is accurate and thus validated. Finally, Figure 45 shows that the lateral error tends to zero and oscillates around zero. Although the oscillation of lateral error is maintained during the simulation, its amplitude is reduced and allows the UAV to track within an acceptable error for this research (detailed information about the lateral error is presented in Table 9). The results presented make it possible to validate the simulation with the complete loop with all the blocks foreseen in the architecture proposed in this thesis.

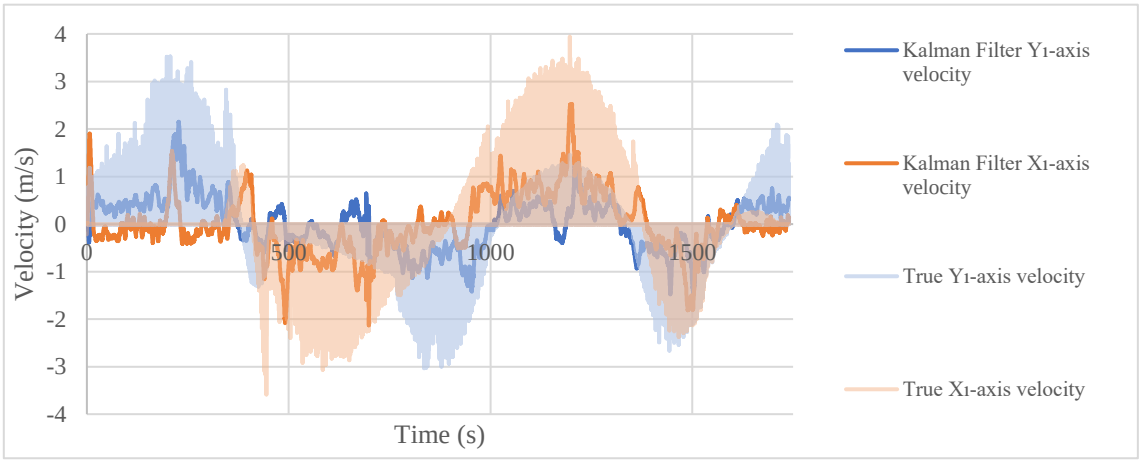


Figure 44. Target velocity estimated by Kalman Filter and true target velocity.

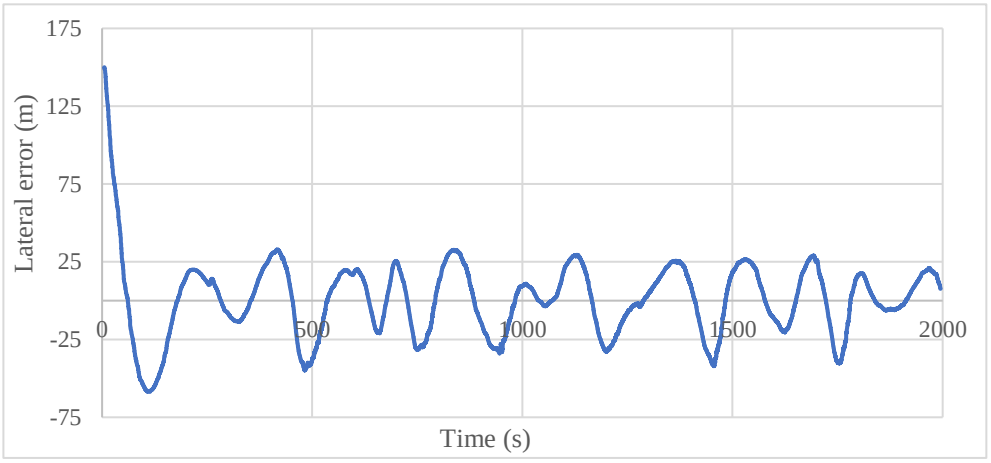


Figure 45. Lateral error from UAV.

Table 9. Information about the lateral error from the UAV.

Results UAV lateral error

Average position error (m)	1.22
Standard Deviation (m)	24.91
Maximum position error (m)	149.99
Minimum position error (m)	-58.71

4 Ground Vehicle Detector

The detection of a ground vehicle is challenging due to the complex background and small size of the cars (Ju et al., 2020). The detection of objects can follow different approaches, as stated in the literature review. However, the Deep Convolutional Neural Network (DCNN) proved to have outstanding success in classifying, detecting and segmenting an object compared to traditional object detectors (Alom, 2018).

Detecting an object through aerial images is different from general object detection once the pictures "*lose their 'faces' because they are captured from a distance to the ground*" (Ju et al., 2020. p. 1). In addition, the UAV's flight changes the background colour, tone, and shade continually, which makes it more challenging for the detector to operate in different scenarios. The aerial images also have several perspectives and dimensions depending on the position and height of the UAV when capturing the image (Dorrer et al., 2019).

This research implements a YOLOv3 detector that follows a data-driven approach, which justifies using a dataset. Regarding the literature review on computer vision, the YOLO detector presented key advantages for this research, such as performing rapid detection, and ability to analyse the whole image, which decreased background errors (Redmon et al., 2016) and improved small object detection for YOLOv3 (Redmon & Farhadi, 2018; Lu et al., 2018; Ju et al., 2020; Cepni et al., 2020). These characteristics are essential for detecting in the simulation environment where the vehicle is consistently changing position in the image plane; thus, rapid detection is required.

In addition, the images captured by the UAV have great latency and poor definition, which sometimes makes it difficult to identify the car, hence the importance of reducing the background error. Lastly, the detector will identify small vehicles in the images. Beyond these aspects, YOLOs have a wide range of repositories available from the community, allowing greater flexibility of choice and even a means of comparison between different projects proposed online (Redmon et al., 2016).

In contrast, one of the main drawbacks of the choice of this detector was the compromise of lower accuracy against the higher detection speed that, as stated previously, is a more significant point for the problem in question (Redmon et al., 2016). Detailed information about the YOLOv3 architecture, loss function and detection process are described in Annex L.

4.1 System Setup

This section describes the aircraft, the camera used to capture the images for the dataset, and the different Gazebo Worlds considered to modify the background environment.

4.1.1 Aircraft

The aircraft used in this research was the fixed-wing standard plane (PX4 User Guide, 2021d), a UAV developed by PX4 autopilot, depicted in Figure 46. This UAV is a small-scale replica of a Cessna 172 with a wingspan of 0.47 m, length of 0.47 m, and 0.11 m height. In addition. this plane model can carry an onboard camera which was a key requirement for this research.

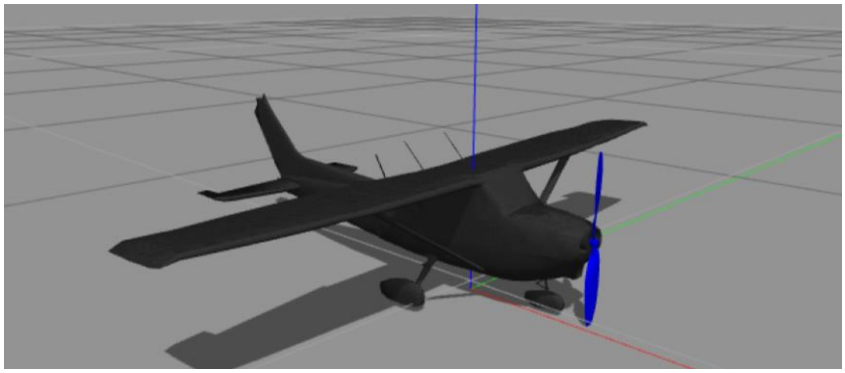


Figure 46. Standard Plane from PX4.

4.1.2 Camera

While choosing the plane that would be integrated into the simulations, one of the main concerns was whether this plane would be able to carry a camera. Therefore, the camera used in this research was a camera plugin provided by PX4 that operates in the visible spectrum.

Various differentiation parameters used in the camera were modified to increase the dataset's diversity. Changes to the camera included adjusting the pan, tilt, field of view and resolution in order to get different perspectives of the vehicles.

In this research, a fixed camera was adopted, so any changes in its characteristics required a new simulation to be started. As a result, the camera was mounted on the airframe in several setups (detailed information about the different dataset setups is presented in Table G-1). Figures 47 and 48 display two examples of those setups.

In particular, Figure 47 presents the tilt and pan used on the camera to take the images from a vertical perspective; hence the camera is pointed downwards with its optical axis aligned with the vertical body frame axis. Figure 48 shows the camera setup aimed to be used in the further simulations conducted in this research. In this figure, the camera was pointing at 68.76º

to the left of the aircraft and approximately 31.78º from the horizontal plane. The choice of 31.78º for the tilt was based on maintaining a 45º angle between the camera and the vertical earth axis. Since the aircraft, on average, banks 13.23º degrees with a radius of 100 m, a tilt of 31.78º permits reaching 45º on the camera.

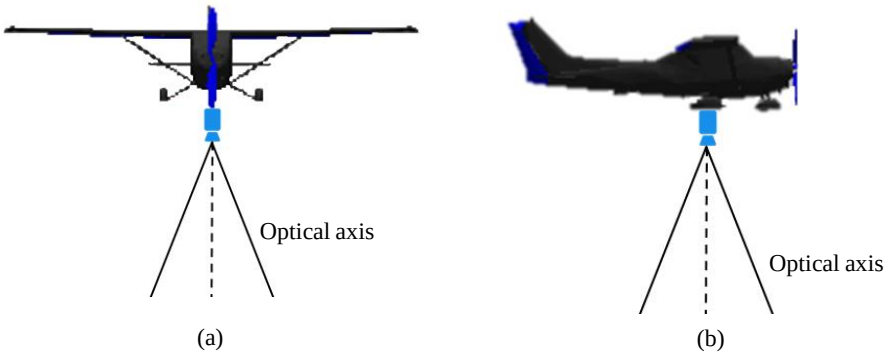


Figure 47. Front (a) and side (b) views of the camera's orientation. This configuration was used to capture images from a vertical perspective.

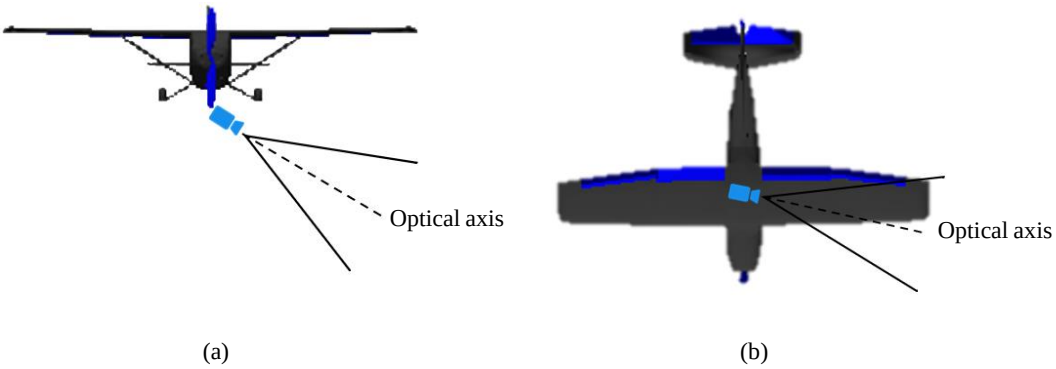


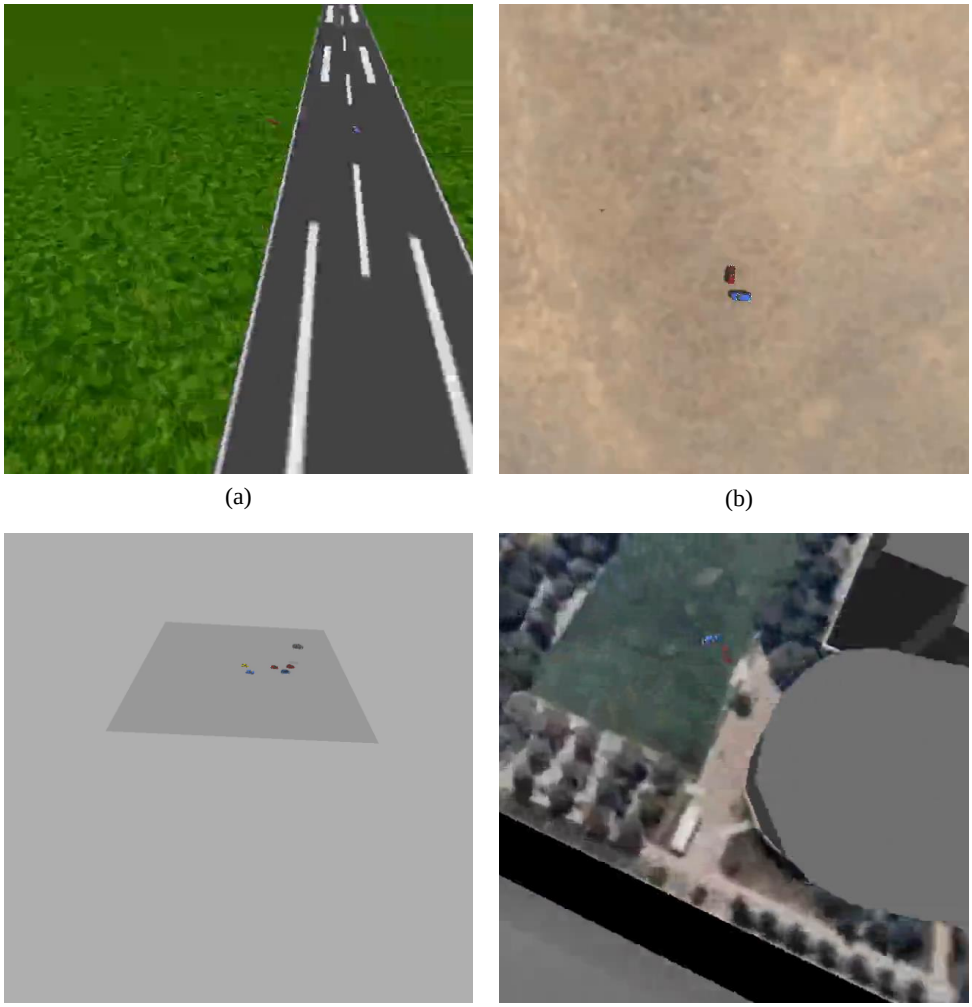
Figure 48. Front (a) and top (b) views of the camera's orientation. This configuration was used to capture images from a side view and is the setup used for further simulations.

4.1.3 Different Scenarios considered for Image Capture

In order to ensure diversity in the dataset, four scenarios were used for image capture, as shown in Figure 49. Six cars were placed with different models, sizes, and colours, along with these scenarios. All the vehicles occupied two Gazebo units in length and height, and one Gazebo unit in width, as represented in Figure 50. In addition, the colours of the vehicles vary from black, yellow, blue and red.

Although the aircraft is predetermined to perform the surveillance mission flying at an altitude of 40 meters with a turn radius of 75 meters, images were captured at different heights and turn radius so that, there was greater flexibility in changing one of these parameters during

the mission. That said, images were collected at 30, 40, 50 and 60 m altitude and with 75, 90, and 100 meters turn radius.



(a) (b)

(c) (d)

Figure 49. Example images of the four scenarios used for the dataset development. Image (a) corresponds to the Runway scenario, image (b) to the Mcmillan Airfield, image (c) to the Ground Plane scenario and image (d) to KSQL Airport. Images (a). (b) and (c) show two ground vehicles and image (c) displays six ground vehicles.

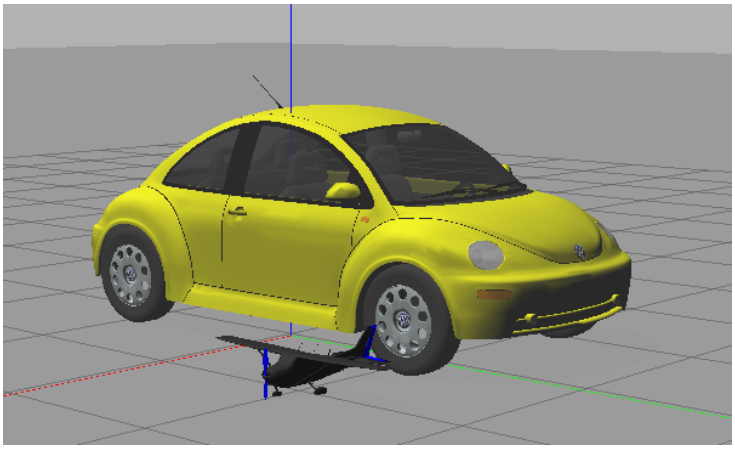


Figure 50. Yellow Beetle car from Gazebo.

4.2 Dataset

Several datasets of aerial car images were proposed in the literature, such as Vehicle Detection in Aerial Imagery (VEDAI), introduced by Razakarivony & Jurie (2016). and Vehicle Aerial Image Dataset (VAID), developed by Lin et al. (2020). However, those datasets mainly use pictures taken from the upper perspective of a car, which is not ideal for the problem at hand since the UAV will capture the cars from a side perspective, as shown in Figure 49 (a) and Figure 49 (c).

Moreover, the detection is carried out in a simulated environment, so having a dataset close to that environment will improve the detector's performance. Given those reasons, a dataset was developed to be more representative of the scenario, improving its accuracy.

The developed dataset contains 1153 images of cars from different models and sizes. As stated in the previous section, the pictures were taken from a UAV at different heights, perspectives, and camera settings in several Gazebo worlds, making a total of eight tests- The tests from 1 to 6 were performed with the exact parameter assigned for each test. In contrast, tests 7 and 8 were conducted with different parameters that range according to Table G-1 from Annex G.

In each image, the individual labelling of the objects of interest was performed by a human operator through a labelling tool called Label-Studio (Label Studio, 2022). The bounding boxes' width and height in pixels varied from X=[0.4218, 80.8155] and Y=[0.2109, 72.0788], respectively.

4.3 Detector Implementation

Since the goal of this dissertation regarding computer vision is to detect a car, an existing YOLOv3 repository within the community, duly validated, has been used. The repository used was TrainYourOwnYOLO, available on GitHub and developed by Muehlemann (2022). This detector was previously trained on a database, so a new learning will not be started. Instead, it will be retrained to detect one class of objects: cars.

Keras and TensorFlow were the libraries used to implement the deep learning techniques. In addition. the Adaptative Moment Estimation Optimizer (Adam) was used as an optimization algorithm to train the model (Ezat et al., 2021). This optimizer was developed by Kingma & Ba (2017) and it is an extension to stochastic descent that recently has broader adoption for deep learning applications in computer vision. The authors describe the optimizer as straightforward to implement, computationally efficient, little memory requirements, among other advantages (Brownlee, 2018).

The classical stochastic gradient descent maintains a single learning rate for all weights updates, which means that the learning rate is not modified during training. Nevertheless, the Adam optimizer "*computes individual adaptative learning rates for different parameters from estimates of first and second moments of the gradients*" (Kingma & Ba, 2017. p. 1). In addition, an advantage of Adam is that it integrates two other extensions of stochastic gradient descent, particularly the adaptative gradient algorithm (AdaGrad) and the root mean square propagation (RMSProp) (Brownlee, 2018).

The dataset proposed for this study was divided into 90% for the training set and 10% for the validation set. Later, forty more images were captured, ten for each world of the Gazebo, to test the detector after the network training. Figure I-1 from Annex I displays four examples of an original image from the test set and respective detection.

Different batch sizes and learning rates were used during the neural network training. Initially, for the first training stage, the learning rate was set on $l_{rate} = e^{-3}$ with a batch size of 32 for 51 epochs, and 249 of 252 layers were frozen. In the next stage, the learning rate parameter was set on $l_{rate} = e^{-4}$ with a batch size of 16 for 51 epochs, and all the layers were unfrozen.

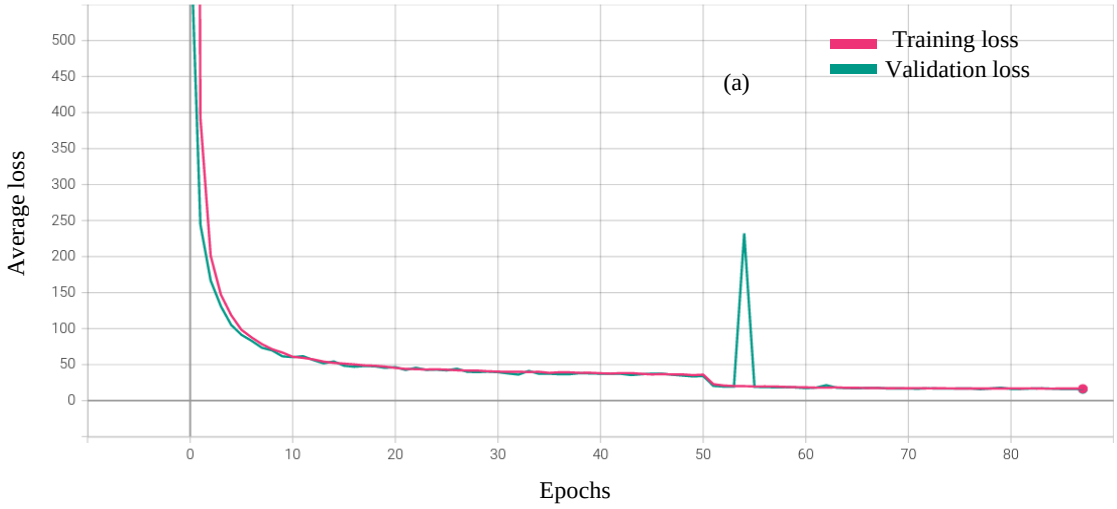


Figure 51. Average loss function regarding the trainning of the neural network.

Since the train begins using frozen layers, only a small part of the model is trained; thus, the loss is larger. In contrast, when the model unfreezes all layers, more parameters are available to learn, reducing the loss (Brock et al., 2017). The change from frozen to unfrozen layers can be noticed in Figure 51 at epoch 50, where the loss reached 36.09 and subsequently declined with a greater gradient.

After this drop, a high peak (a) from validation loss is noticed in epoch 54, with a loss of 232.2 due to incorrectly adjusted weight in that training epoch. Subsequently, the loss function value declines until an early stop in the 87th epoch with a training loss of 16.53 and validation loss of 15.89, as displayed in Figure 51.

Bengio et al. (2016) suggest two main factors determining if the machine learning algorithm will execute well: the capacity to make the training error short and the gap between training and validation error small. Both factors are considered the two central challenges in machine learning: underfitting and overfitting. Briefly, underfitting appears when the model can not achieve a low error value in the validation set; overfitting results from a large difference between the training and validation error (Bengio et al., 2016).

Regarding the graphic in Figure 51 and considering the underfitting and overfitting explained above, it is possible to conclude that the model has a good fit learning curve. A good fit is defined as the training and validation loss decreasing to the stability point with a minimal gap among the two final loss values (Brownlee, 2019), which means that the model does not suffer from a significant amount of underfitting or overfitting (Bengio et al., 2016). The training loss is expected to be lower than the validation loss. The gap between both losses is called the generalization gap (Brownlee, 2019).

4.4 Neural Network Evaluation

Mean Average Precision (mAP) is a metric commonly used to evaluate the performance of object detectors and its values range from 0 to 1. This metric indicates the performance of the detectors over different classes and although there is only one class in the current problem, the mAP was still used for easier comparison with other implementations. The mAP depends in other metrics like the confusion matrix, intersection over union, recall and precision, which are described in the following subsection (Rizzoli, 2022).

4.4.1 Mean Average Precision Metrics

The confusion matrix focuses on four elements: true positives, true negatives, false positives, and false negatives. The attribute true occurs when the prediction from the model is considered correct, and it is considered false when there is a mismatch between the model prediction and the ground truth labels (Gad, 2020).

Furthermore, the IoU measures the performance of an object detector by measuring the overlap of the ground truth bounding box and the predicted box of the model. A higher IoU

means that the predicted box is close to the ground truth box (Rizzoli, 2022). The following equation describes how to calculate the IoU,

$$IuO = \frac{Area\ of\ Overlap}{Area\ of\ Union}. \tag{4.8}$$

Afterwards, the precision measures the ratio between the number of true positives and all positive predictions. as displayed in the following

$$Precision = \frac{True_{positive}}{True_{positive} + False_{positive}}. \tag{4.9}$$

The precision level determines the model's confidence in classifying a sample as positive. The higher the precision, the more confident the model is when classifying a sample as positive (Gad, 2020).

The recall measures how well it is possible to find true positive samples correctly classified as positive out of the whole positive samples, which means that this parameter focuses only on the positive samples and how they are classified, as shown in

$$Recall = \frac{True_{positive}}{True_{positive} + False_{negative}}. \tag{4.10}$$

The higher the recall, the more positive samples the model correctly classified as positive (Gad, 2020).

4.4.2 Mean Average Precision Computation

The Average Precision (AP) can summarize the precision-recall curve into a single value representing all precisions' average using the following equation

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k), \tag{4.12}$$

where $Recalls(n) = 0$, $Precision(n) = 1$ and $n = number\ of\ thresholds$ (Gad. 2020). The mean average precision corresponds to the area under the recall-precision curve.

The network evaluation was performed with 125 images with several backgrounds and a different number of ground vehicles. The threshold of the evaluation was changed (see Table 10) in order to understand how it affected the mAP. In Figure 52, it is observed that the mAP increases approximately linearly with decreasing IoU threshold. This result is expected since the system accepts more detections by the network as true by decreasing the threshold

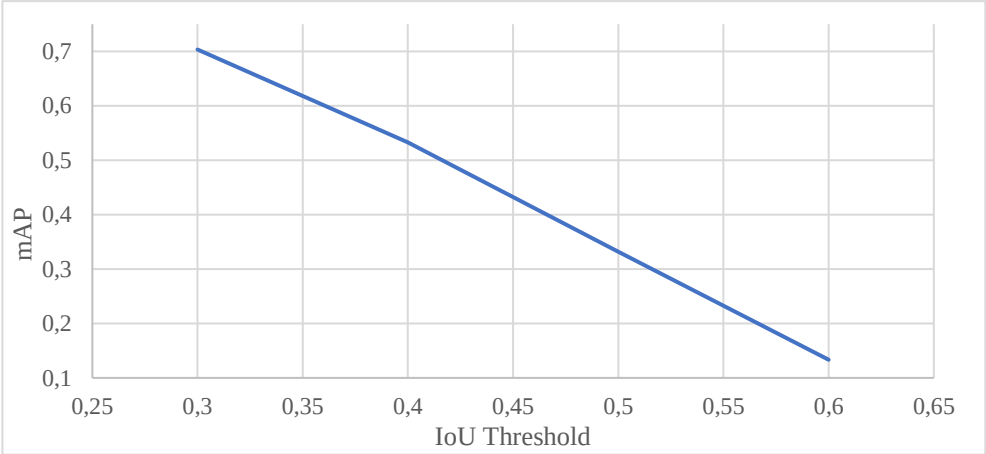


Figure 52. Variation of mAP as a function of IoU threshold.

Table 10. Network evaluation results for different IoU Thresholds.

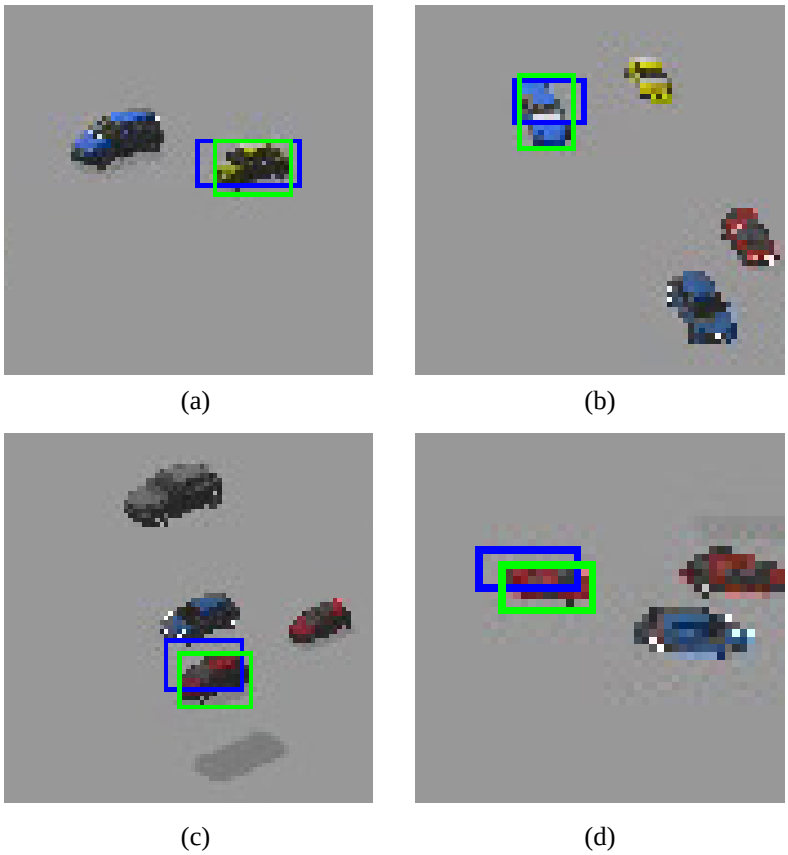
IoU Threshold	mAP
0,3	0,703
0,4	0,533
0,5	0,331
0,6	0,133

Figure 53 displays four example images with a detection in blue and the corresponding ground truth in green. Each image has a different IoU threshold.

Regarding Figure 53 (d), which corresponds to a 30% IoU threshold, the system will be more permissive on detections since it only requires a 30% intersection between detection and ground truth to consider the detection true. As a result, the mAP will increase against lower thresholds.

Applying this detection to the case study context from this thesis, the difference in coordinates would be insignificant when calculating the ground truth and detection coordinates, considering that the UAV is flying at 40 meters with a turning radius of 100 meters. So, even though 30% is a low threshold, it still allows obtaining the car's coordinates with an accuracy of +- 1.5 meters.

In function of these conditions, it is verified that it is not necessary a threshold so high to obtain the desired relative position since the variation of the squares ends up being practically insignificant for the proposed mission.



(a) (b)

(c) (d)

Figure 53. Images with the detection bounding box in blue and the ground truth bounding box in green. Image (a) corresponds to a threshold of 60%, (b) to a threshold of 50%, (c) to a threshold of 40% and (d) to a threshold of 30%.

5 Closed-loop Validation – Final Results

In order to validate the close-loop, the architecture of Figure 6 is implemented. Until this chapter, the Guidance and Control, Target Geolocation and Target Estimate modules with the colour filter have been validated. In the previous chapter, computer vision was validated in a realistic environment. Consequently, the Target Detection block is now be replaced with the vehicle detector described in Chapter 4.

Previously, the detection of the red block was performed at a rate of 10 Hz, enabling the rate of 2D coordinates publication to be equal to the rate of measurement of the Geolocation. Therefore, each Kalman Filter update had a new Geolocation measurement because the filter ran at a 10Hz rate. Since the MPF also runs at a rate of 10 Hz, the information from the target to feed the Guidance and Control is always given by the Kalman Filter update estimate.

Conversely, when introducing the YOLO detector, the frequency between detections is approximately 0.15 Hz which is considerably slower than the rate of the MPF. To solve this issue, the Target Geolocation rate was changed to match the rate of detections, and the Target Estimate maintained a rate of 10 Hz, equals to the Guidance and Control node.

So, when detection occurs, the Target Geolocation calculates the estimated position, and until the next detection, the Target Geolocation does not send position measurement to the Target Estimate. As a result, the prediction phase of the Kalman Filter from the Target Estimate block in these periods is used to feed the MPF. Furthermore, since there are different rates between the Geolocation and the Kalman Filter, the Target Geolocation and Target Estimate run in distinct nodes.

Initially, we test the closed-loop with a vehicle standing in a constant position in the Runway Gazebo world displayed in Figure 49 (a), chosen to be the realistic environment for the final simulation. Next, the vehicle moves in a pre-determined path, changing its orientation and velocity similar to a car in a real environment.

5.1 Closed-loop Validation with the Object Standing at a Constant Position

The first simulation was conducted with the target standing on the ground at a constant position, $X_I = -30\,m$ and $Y_I = -10\,m$. The simulation started with the UAV loitering clockwise with a 100 m radius and centre in $X_I = 0\,m$ and $Y_I = 0\,m$ at an altitude of 40 m and 15 m/s.

The change in height and turn radius to the previous validation values from Chapter 3 increased the quality of the images captured from the camera for further detection. As a result,

the new turn radius changed the aircraft's camera tilt value as the roll angle depended on that radius, as mentioned in Chapter 4. As a result, the tilt angle used was -31.81º. The detailed settings used in this simulation regarding the object, the UAV and the camera are mentioned in Table J-1 from Annex J.

Figure 54 displays the 2-Dimensional perspective of the simulation regarding the UAV position, target estimated position and target true position. The orange line represents the UAV trajectory around the target position; the green point corresponds to the start position, and the black dot to the final location. In addition, the position estimated by the Kalman Filter is in red, and the dark blue corresponds to the true target position.

Based on the estimated and true target position, it can be stated that visually the Kalman Filter estimate is close to the actual path of the target. On average, the estimated Kalman Filter position error is 5.66 m, and the standard deviation is 3.24 m (detailed information regarding the Kalman Filter position estimation presented in Table 11).

Regarding Figure 55, the target's estimated position (solid line) oscillates between the discontinuous dashed line representing the true position, with a maximum error of 14.62 m. This behaviour, by the estimation, is consistent with the previous Kalman Filter validations.

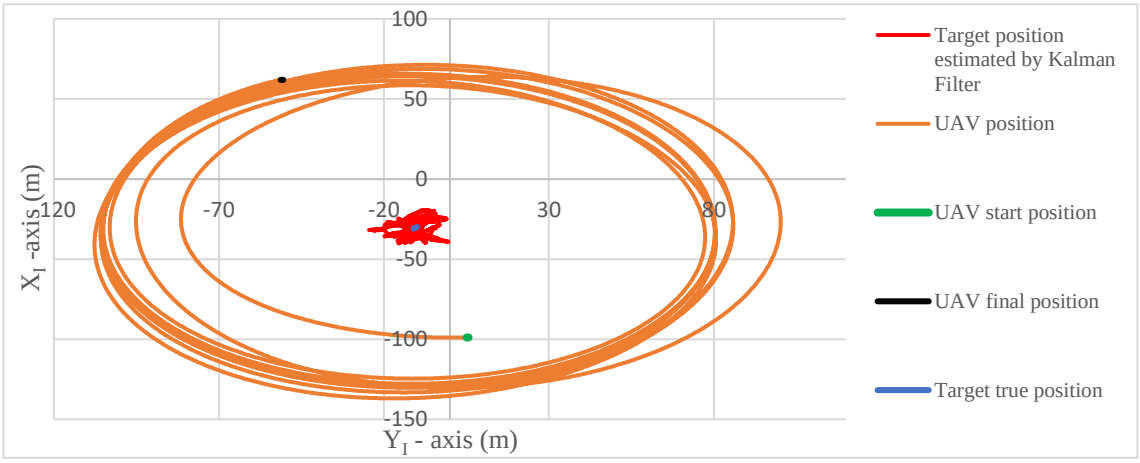


Figure 54. 2-Dimensional view of the simulation.

Table 11. Results from Kalman Filter estimation.

Results from Kalman Filter

Average position error (m)	5.66
Standard Deviation (m)	3.24
Maximum position error (m)	14.62
Minimum position error (m)	0

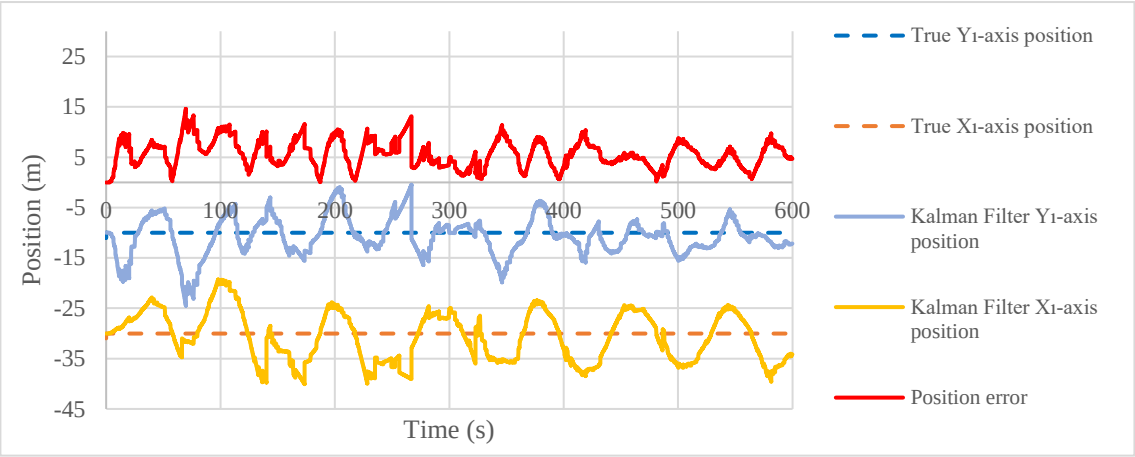


Figure 55. Position estimate and error from Kalman Filter and true target position.

The additional parameters estimated by the Kalman Filter are displayed in Figure 56. Since the target is standing in a constant position, the velocity and acceleration values are expected to be close to zero as show in Figure 56.

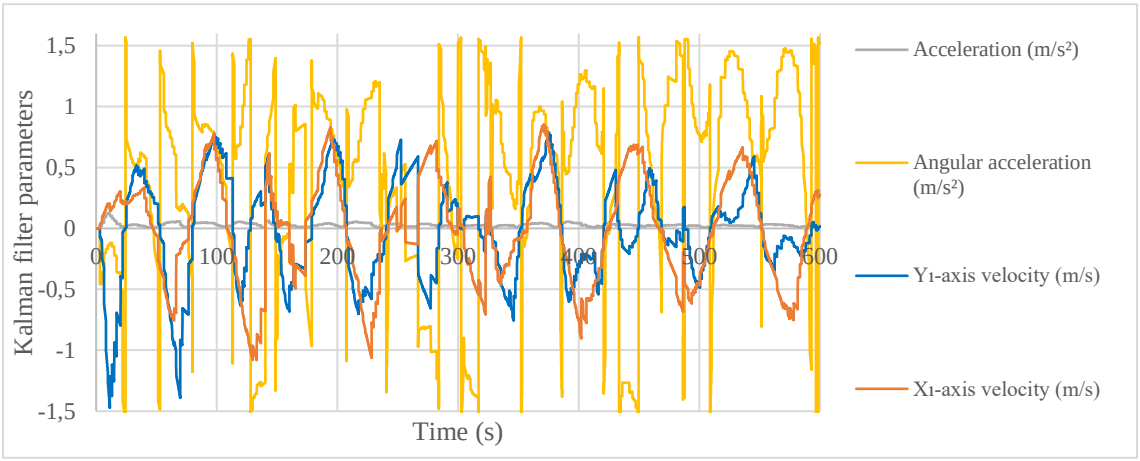


Figure 56. East and north axis velocity, acceleration, angular acceleration and orientation estimated by the Kalman Filter.

The values in Table 12 are calculated after the UAV has converged to the path, disregarding the initial approach phase as this depends on the relative initial position between the UAV and the path itself. The lateral error oscillated between a maximum of 38.74 m and a minimum of -22.78 m. From that moment on, the error decreases with a tendency to zero, as displayed in Figure 57. The average lateral error is 4.24 m, and the standard deviation is 13.81 m, both being expected and acceptable values for validation, taking into account the validation done to MPF in Chapter 3 (see Table 12).

Regarding Figure 57, there is a discontinuity in the system in the 157.11^{th} second since the Kalman filter only started estimating the UAV parameters after the first detection. This causes the UAV to instantly change it loiter from the centre $X_I = 0\ m$ and $Y_I = 0\ m$ to the first estimate coordinates, $X_I = -30\ m$ and $Y_I = -10\ m$.

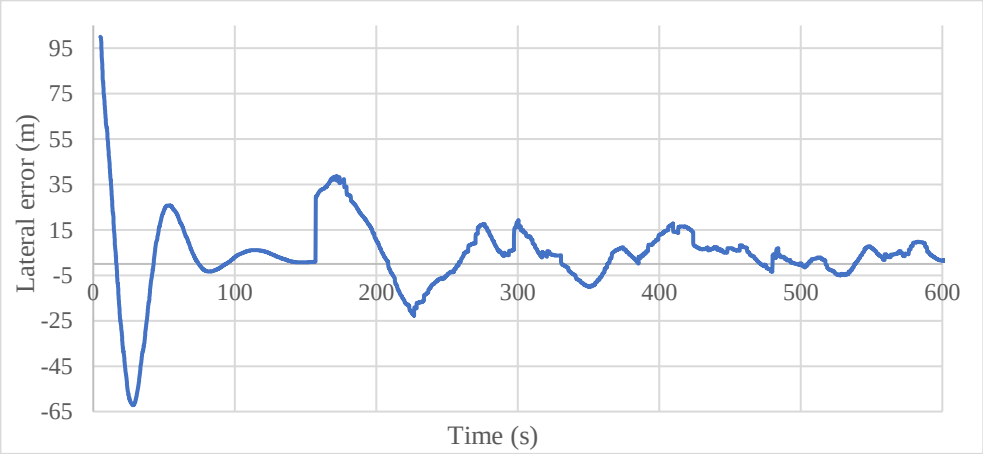


Figure 57. Lateral error during simulation.

Table 12. Results regarding the MPF lateral error.

Results UAV lateral error

Average position error (m)	4.24
Standard Deviation (m)	13.81
Maximum position error (m)	38.74
Minimum position error (m)	-22.78

Focusing on the Computer Vision part, its validation will be qualitative since its quantitative validation was carried out in Chapter 4. In a first approach to the detections in this simulation, all objects detected as cars were considered. Consequently, when the aircraft was converging to the loiter with the centre at the target, sometimes the UAV stopped having the vehicle in sight, detected targets in the distance with low confidence and considered them as accurate detection. This behaviour impaired the simulation since high velocities and accelerations were generated that did not allow the aircraft to converge to the loiter.

In order to solve this problem, a threshold value of 60% confidence was restricted for the detection to be considered. Thus, whenever the UAV did not have the car in sight and made a false detection, it would not be considered, and the UAV would have to be guided by the output of the Kalman Filter prediction from the Target Estimate block.

In the case of obtaining several detections that were above 60%, the one with the highest confidence was considered. During several simulations, when more than one car was detected, the true positive detection was always the one with the highest confidence and, therefore, the one chosen to perform the tracking.

During the simulation, 415 frames of the video were analysed, of which 314 obtained detections and 99 had no detections, as displayed in Table 13. Of the 314 frames detected, 27 detected more than one car. Since each image had only one car placed, we can conclude that the detector presented false positives.

The importance of having a separated block to the Kalman Filter (Target Estimate node) with an independent ROS node was confirmed when the average detection time was 1.57s. Since the Target Geolocation has a rate of 10 Hz or 0.01s per iteration, the detection rate was low to feed the Geolocation algorithm.

Table 13. Detection results.

Detection Results

Analysed frames	414
Frames with detections	315
Frames without detections	99
Total detections in the simulation	342
Frames with two detections	27
Average time between detections (s)	1.57

Concerning the results previously presented, the closed-loop architecture with the object stopped in a fixed position was successfully validated. Afterwards, the architecture with the ground vehicle in movement is tested.

5.2 Closed-loop Validation with the Object Moving

The simulation with the vehicle moving was conducted through the path displayed in Figure 58. The course design was based on a possible path followed by the target. In order to increase the reality of the simulation and make the path as similar as possible to a potential car movement, several changes of direction, velocities and stops were used and are described in detail in Table K-1 from Annex K.

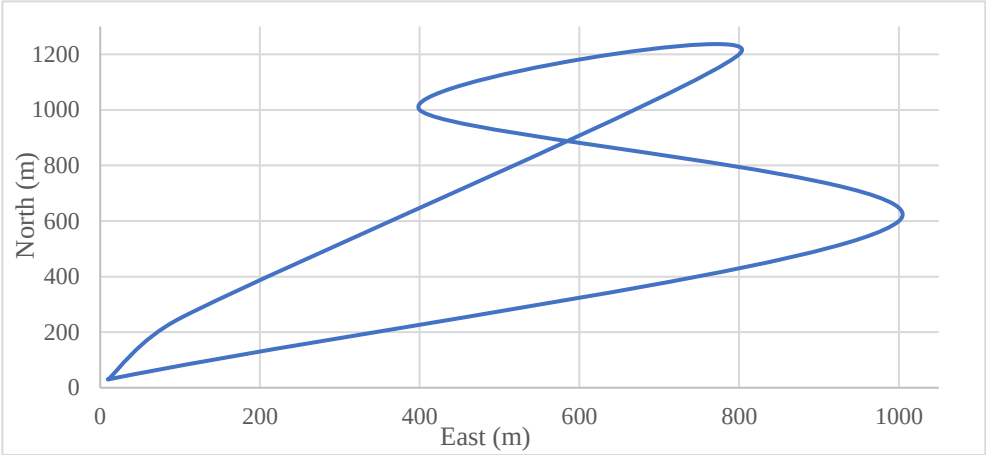


Figure 58. 2-Dimensional view of the target path.

After performing several simulations with the proposed path, the time between detections and the ground vehicle movement provided poor estimations of acceleration by the Kalman Filter. Furthermore, those acceleration estimations impaired the MPF tracking, leading the simulation to have false positive detections, which generated high lateral errors.

As a result, the Kalman filter's model was updated in order to consider a constant speed target, as shown in equation (5.1)

$$F = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.1}$$

Matrix (5.1) follows the same equations as the matrix (3.11); however, the dynamic model considers constant speed. The matrix Q and H are identical to the matrix (3.12) and (3.14), yet, the acceleration parameters are not considered. Lastly, matrix R remains equal to the matrix (3.13).

The simulation was performed with the UAV loitering with a radius of 100 meters centred in $X_I = 0\,m$ and $Y_I = 0\,m$ and at an altitude of 40 m (detailed information regarding the simulation settings is presented in Table K-2 from Annex K). After establishing the altitude, the UAV started detecting the vehicle, estimating the coordinates and feeding the MPF for the target tracking.

Figure 59 shows the upper perspective of the simulation, where the orange line represents the UAV path, the red line corresponds to the target's estimated coordinates, and the blue line resembles the target's true path, as displayed in Figure 59. The green and black dots mark the start and final position of the UAV, respectively.

Visually analysing the graph in Figure 59, it is possible to state that the estimation of the target's coordinates allows for obtaining a general drawing of the real path of the car. Indeed,

the mean error, 8.18 m, and the standard deviation, 3.27 m, are within the expected range for this simulation and verified that the filter position estimate is accurate enough for the MPF to follow (detailed information regarding the Kalman Filter position estimation is displayed in Table 14). Moreover, the UAV can efficiently maintain continuous loiters around the target, both in straight lines and curved segments, which tend to be the most difficult segments for tracking once the car changes direction and speed continuously.

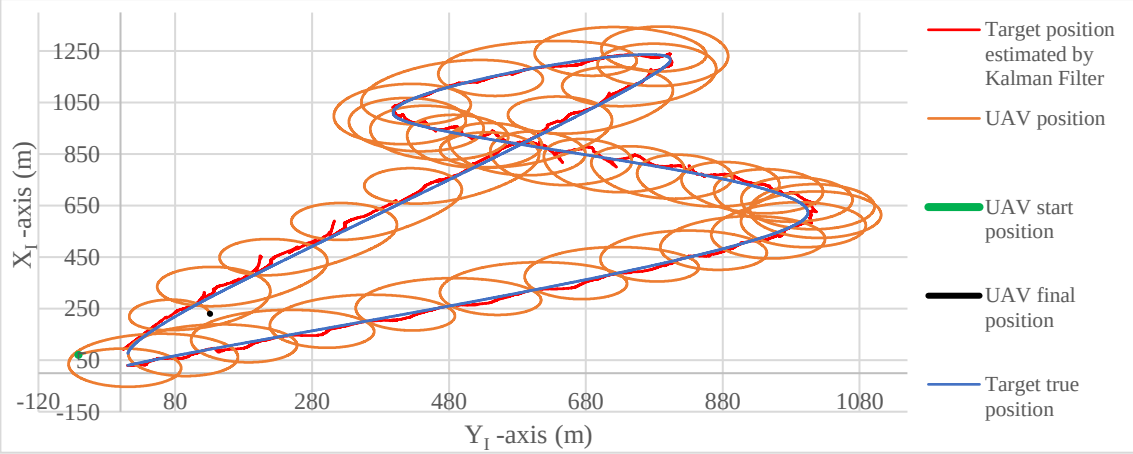


Figure 59. 2-Dimensional view of the UAV, target estimated position and target's true position.

Table 14. Results from the Kalman Filter position estimation.

Results from Kalman Filter

Average position error (m)	8.18
Standard Deviation (m)	3.27
Maximum position error (m)	71.24
Minimum position error (m)	0

Regarding Figure 60, the target's estimated position (solid line) oscillates between the discontinuous dashed line representing the true position. This behaviour, by the estimation, is consistent with the previous Kalman Filter validations.

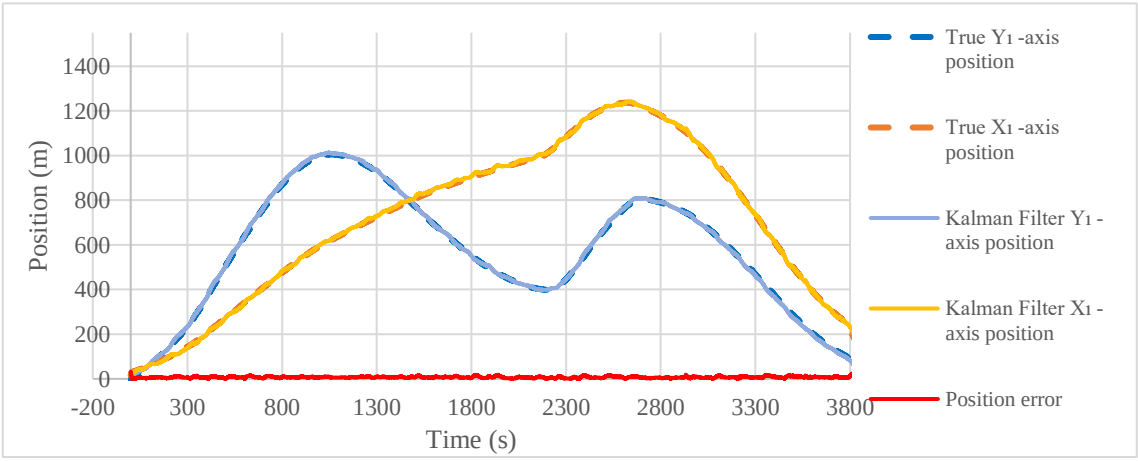


Figure 60. Position estimate and error from Kalman Filter and true target position.

The velocity of the target by the Kalman Filter and from Gazebo telemetry are displayed in Figure 61. For the same reasons presented in Section 3.4.4, the target speed provided by Gazebo is noisy yet allows an analysis to be made of the behaviour of the estimated speed.

It is also possible to verify through the graph below that the velocity behaviour follows the low frequency component of true velocity. Moreover, it is possible to state that the east and north velocities vary according to the speeds defined in the path (detailed information regarding the target path is presented in Table K-1).

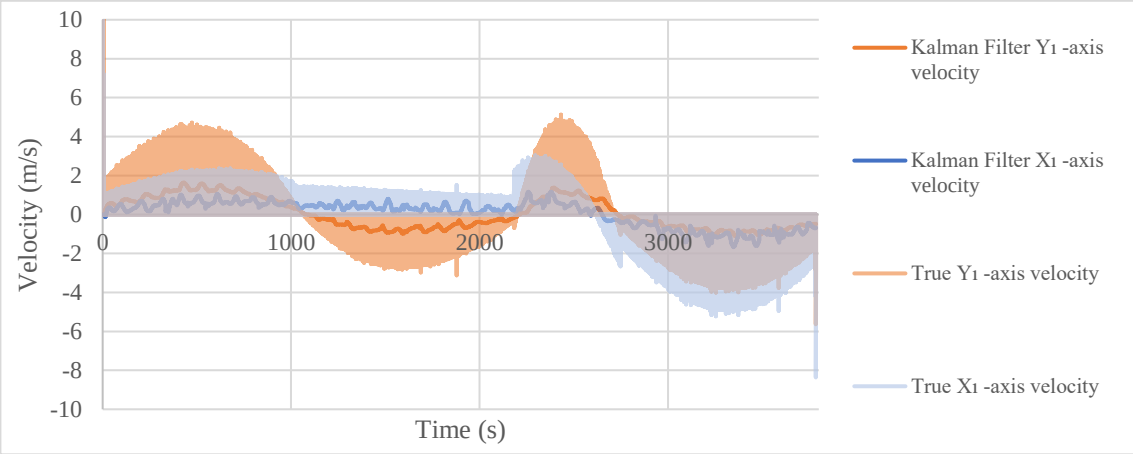


Figure 61. Velocity estimated from Kalman Filter and true velocity of the target.

The MPF gains changed compared to those used until this simulation (consult Table K-2). Initially, the orientation and position error gains, $g_1 = 0.1$ and $g_2 = 0.00012$, respectively, were causing high amplitude oscillation of the lateral error. However, through an iterative gain adjustment process based on previous experience, it was possible to realize that the proposed MPF gains, shown in Table 15, enable obtaining the best lateral error results compared to the other gain combination results. Furthermore, improvements to the results presented in Figure 62 would only be obtained by modifying the controller inner loop.

The values in Table 15 are calculated after the UAV has converged to the path, disregarding the initial approach phase as this depends on the relative initial position between the UAV and the path itself. Although the lateral error does not continuously tend towards zero in Figure 62, it makes a harmonic effect centred on zero and peaks at -20.78 m and 34.90 m, as displayed in Table 15, and are acceptable results for this research.

As mentioned in section 5.1, Figure 62 has a discontinuity point in the 144.2^{th} second, representing the moment when the centre of the UAV loiter changed from $X_I = 0\ m$ and $Y_I = 0\ m$ to $X_I = 30\ m$ and $Y_I = 10\ m$, which corresponds to the first detection.

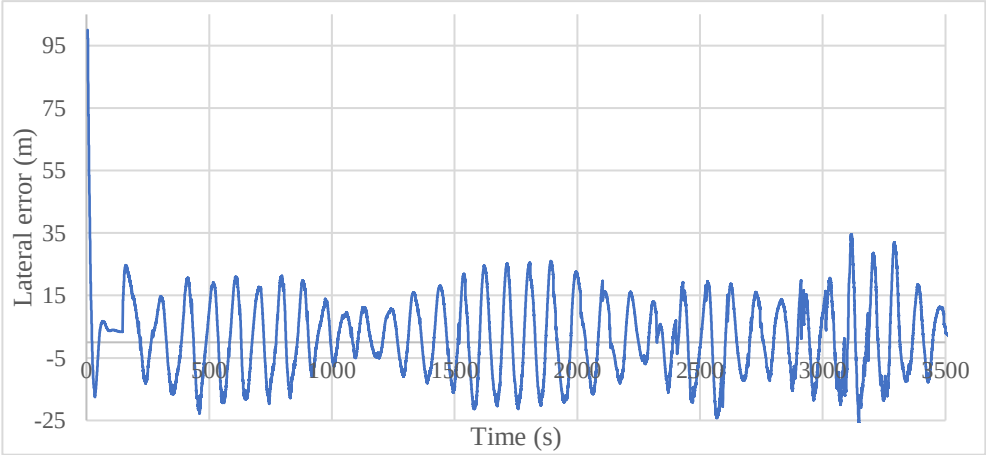


Figure 62. Lateral error during simulation.

Table 15. Results regarding the MPF lateral error.

Results UAV lateral error

Average position error (m)	8.09
Standard Deviation (m)	14.19
Maximum position error (m)	34.90
Minimum position error (m)	-20.78

As elaborated in the previous Section 5.1, the validation of the Computer Vision part will be based on reading quantitative data from the detections and qualitative analysis of the detector performance during the simulation. The threshold value used for the detection was 60%.

Regarding Table 16, of the 2754 analysed frames, just 1878 frames had detections above the established threshold, which means that the Target Estimate block with the Kalman Filter was decisive in predicting the coordinates of the target in between those measurements. In addition, among all the frames that had detection, 109 frames obtained 2 detections, which means that there have been few false positives above the denominated threshold.

Table 16. Detection results.

Detection Results

Analysed frames	2754
Frames with detections	1878
Frames without detections	876
Total detections in the simulation	1987
Frames with two detections	109
Average time between detections (s)	1,28

Considering all the results that has been presented, the software architecture is validated to detect a ground vehicle and track it. Furthermore, the modifications made to the Kalman Filter and the MPF gains allowed promising results to obtain the closed-loop validation.

6 Conclusions and Recommendations

6.1 Conclusions

The present dissertations aimed to implement an adequately validated and documented software open-architecture compliant with the hardware architecture proposed by Silva et al. (2020) and on the community available and most used software components. This software architecture's goal was to constitute a modular tool that allows rapid prototyping of control algorithms and computer vision. This goal was validated in this thesis by implementing a case study where a fixed-wing UAV is required to detect and track a moving ground target.

First, a literature review, which aggregated all the theoretical basis of this dissertation, was presented. Initially, the hardware architecture from Silva et al. (2020) was reviewed, underlining the hardware's constraints to the software architecture to be proposed in this thesis. Then the different software solutions were studied in depth. Subsequently, two areas were addressed, namely Guidance and Control Techniques and Computer Vision.

In chapter 3, the hardware/software architecture was presented, and the software tools decision was carried out. First, the PX4 flight stack was chosen for the Pixhawk autopilot due to its support for an Offboard control mode for fixed-wing UAVs. Next, the Ground Station chosen was the QGroundControl because of the full compatibility with the autopilot flight stack. The mission computer was left open to two choices, the Raspberry Pi or Nvidia TX2. Both mission computers could support the software choices in terms of the operating system, Linux, and ROS software application. Lastly, the Gazebo simulator was adopted due to the availability of a full pack installation of PX4, ROS Melodic and Gazebo 9, as well as the access to a fixed-wing aircraft model.

Afterwards, the software architecture was proposed, and each block was explained in detail. The architecture comprises four main blocks, Gazebo simulator, PX4 autopilot, QGC and ROS environment that exchange telemetry and control messages via ROS and MAVROS. The Gazebo simulator was the simulation software that created the simulated environment for the testing, and the QGC worked as the software for the Ground Station. In addition, the PX4 was the autopilot that enabled the conversion of the thrust, pitch and bank values to the deflection of the aircraft's flight surfaces and motor setting. Finally, the ROS environment contains the Control and Guidance, Video Acquisition, Target detector, Target Estimate and Target Geolocation to ensure the detection, geolocation, filtering and guidance functions.

In this thesis, a modular and incremental validation logic was followed, starting with the validation of the Guidance and Control node, followed by the Video Acquisition and Target

Detector, then Target Geolocation and Target Estimate until the control loop with vision is closed. These preliminary validations were carried out with the detection of a red object. The Target Detector module was then validated with the ground vehicle detector. Finally, the closed-loop architecture was validated.

In the Guidance and Control node, the outer loop controller uses the Offboard mode to issue roll, pitch, yaw and thrust commands, which are then used by the PX4 autopilot's inner loop controller to calculate the necessary flight surface deflections and motor settings. The trajectory control was developed separately for the lateral and longitudinal control of the UAV. The lateral control was implemented based on Oliveira & Encarnação (2013) article, using the MPF method, which uses non-linear control techniques. The longitudinal control was implemented based on PI controllers.

The longitudinal control was successfully validated in several simulations where the UAV was commanded to maintain an airspeed and altitude setpoint in climb, descent and cruise. In addition, the trajectory control was tested with a fictitious target randomly generated and reacted positively to follow the dummy target.

The Video Acquisition node imports the camera video from the simulator through a UDP port and provides it to the Target Detector node. Initially, in Chapter 3, the Target Detector uses a filter from OpenCV library to detect a red target. Later, in Chapter 4, a YOLO detector was adopted to detect a moving ground vehicle. Finally, it was validated that the target coordinates in the image frame were published by the Target Detector through a ROS message.

In the Target Geolocation block, it was implemented a Geolocation algorithm from Barber et al. (2006) to compute the position of a ground target throughout its location and motion streamed in the video sequence. This algorithm was tested by calculating the position with the target standing still and moving and comparing the results with the true target position.

Next, a Kalman Filter was used in the Target Estimate node to estimate and predict the other parameters required by the MPF through linear dynamical systems. The filter was tested to estimate the target parameters in two scenarios, where the target was at a constant position and moving on a predetermined path. Results showed that the filter could estimate the target parameters successfully, thus validating this module.

In the final phase of Chapter 3, the final validation of the closed-looped with the colour filter detector showed that the aircraft could take off, maintain a predetermined altitude, airspeed and loiter radius, then detect a red object, whether it is moving or not, and finally track it.

In Chapter 4, the implementation of the YOLOv3 detector was made by initially describing in detail the main philosophy of the detector and further advantages to sustain and enhance its choice for our problem in specific. Since the detections were performed inside the simulator, a new dataset with aerial images of cars in four Gazebo worlds has been elaborated to approximate the model's dataset to what it would detect in the simulation. The dataset was composed of 1153 images in which several parameters, in particular, the camera settings, UAV altitude, loiter radius and background Gazebo world, were changed.

The YOLOv3 implemented in this research was developed online by another author, so most training setups were maintained because the repository was validated with those settings. Furthermore, the training loss graph showed no underfitting or overfitting, meaning that the model has a good fit.

The evaluation of the neural network was conducted through the mean average precision calculation. Different thresholds were used to compute the mAP. An increase in the threshold causes a decrease in the mAP as expected since the system accepts more detections by the network as true by decreasing the threshold.

The last chapter comprised the final simulation where the closed-loop was validated. This simulation was used together with the previously tested Guidance and Control and Target Geolocation and Target Estimate nodes and the Target Detector, which uses the YOLOv3 detector. Since these sub-modules were validated individually, this final closed-loop simulation aimed to prove that the architecture could accomplish the proposed mission. In summary, the proposed architecture enabled a UAV to detect and track a car fully autonomously.

The main objective, which was developing a software architecture with low-cost tools proposed by the community, was achieved. Moreover, the Guidance and Control, Target Estimate, Target Geolocation, and Target Detector nodes were successfully validated, first individually and then as a closed-loop simulation. As a result, the surveillance mission of tracking a ground vehicle through Computer Vision proposed for this research was also attained.

6.2 Future Work

During the development of this research, several important decisions, particularly the tools used for the architecture, namely the simulator, and ROS versions, needed to be made. Some of the chosen paths led to problems due to no previous knowledge of the specific topics, thus resulting in limitations in certain areas and restricting the time available to develop this project. In the following paragraphs, some recommendations will be addressed.

In what concerns the Gazebo simulator and ROS, the versions used caused some compatibility issues when implementing the YOLOv3 detector due to the Python versions. The Gazebo simulator adopted was the 9th version recommended for use with ROS melodic. Both these versions use Python 2. In contrast, the detector repository implemented recommended Python 3 to run. That said, I would suggest implementing Gazebo 11 with ROS Noetic to run all the blocks of the simulation in the same Python version.

In addition, the adoption of Gazebo 11 could enhance the simulation for several reasons. First, in Gazebo 9, the video captured from the camera had poor resolution and plenty of latency, which degraded the images for the training phase and later in-flight detection; since version 11 is more recent, this problem could be overcome. Another advantage of adopting this version 11 would be the possibility of opening doors to all the latest projects developed by the community, particularly improving the background environment to a more realistic one.

Finally, future studies could focus on improving the YOLOv3 detector or study other state-of-the-art detectors that could perform better in more complex environments than the one used in this research. Another suggestion would be to follow up on this same work, moving forward with the hardware-in-loop phase, and test this same architecture on board a CIAFA UAV in a test flight.

One of the great advantages of the implemented architecture is that it is modular, which allows the development of various projects with different scopes and missions by replacing blocks. Having that in mind, I would suggest a study that uses the camera for a distinct mission, such as automatic landing through computer vision or use another type of sensor for a different task, in particular, an infrared sensor for fire detection.

7 References

Alom, Z. (2018). *Doctor of Philosophy in Engineering* [Doctoral Dissertation, University of Dayton]. file:///C:/Users/HP/AppData/Local/Temp/Alom%20dissertation%20draft4__final%20format%20approved%20LW%2011-8-18.pdf

Alsalam, B. H. Y., Morton, K., Campbell, D., & Gonzalez, F. (2017). Autonomous UAV with vision based on-board decision making for remote sensing and precision agriculture. *2017 IEEE Aerospace Conference*, 1–12. https://doi.org/10.1109/AERO.2017.7943593

Alvarado, M., Gonzalez, F., Fletcher, A., & Doshi, A. (2015). Towards the Development of a Low Cost Airborne Sensing System to Monitor Dust Particles after Blasting at Open-Pit Mine Sites. *Sensors*, *15*(8), 19667–19687. https://doi.org/10.3390/s150819667

APM Planner 2. (2021). APM Planner 2 Home. *Ardupilot*. https://ardupilot.org/planner2/

ArduPilot Dev Team. (n.d.). ArduPilot. *Ardupilot*. Retrieved 26 June 2021, from https://ardupilot.org/ardupilot/

ArduPilot Dev Team. (2019). MAVProxy Developer GCS. *Ardupilot*. https://ardupilot.org/mavproxy/index.html#home

ArduPilot Dev Team. (2021a). Choosing a Ground Station. *Ardupilot*. https://ardupilot.org/copter/docs/common-choosing-a-ground-station.html

ArduPilot Dev Team. (2021b). *Companion Computers*. Ardupilot Versatile, Trusted, Open. https://ardupilot.org/dev/docs/companion-computers.html#companion-computers

ArduPilot Dev Team. (2021c). *Guided Mode*. Ardupilot Versatile, Trusted, Open. https://ardupilot.org/plane/docs/guided-mode.html

ArduPilot Dev Team. (2021d). *MAVLink Interface*. Ardupilot Versatile, Trusted, Open. https://ardupilot.org/dev/docs/mavlink-commands.html

ArduPilot Dev Team. (2021e). Mission Planner Home. *Ardupilot*. https://ardupilot.org/planner/

ArduPilot Dev Team. (2021f). *SiK Telemetry Radio*. Ardupilot Versatile, Trusted, Open. https://ardupilot.org/copter/docs/common-sik-telemetry-radio.html

ArduPilot Dev Team. (2021g). *Simulation*. Ardupilot Versatile, Trusted, Open. https://ardupilot.org/dev/docs/simulation-2.html

ArduPilot Dev Team. (2021h). Telemetry (landing page). *Ardupilot*. https://ardupilot.org/copter/docs/common-telemetry-landingpage.html#short-range-10km

Ariyur, K. B., & Fregene, K. O. (2008). Autonomous tracking of a ground vehicle by a UAV. *2008 American Control Conference*, 669–671. https://doi.org/10.1109/ACC.2008.4586569

Armanini, S., Caetano, J., Croon, G., Visser, C., & Mulder, M. (2016). Quasi-steady aerodynamic model of clap-and-fling flapping mav and validation using free-flight data. *Bioinspiration & Biomimetics*.

Barber, D. B., Redding, J. D., McLain, T. W., Beard, R. W., & Taylor, C. N. (2006). Vision-based Target Geo-location using a Fixed-wing Miniature Air Vehicle. *Journal of Intelligent and Robotic Systems*, *47*(4), 361–382. https://doi.org/10.1007/s10846-006-9088-7

Beard, R., Kingston, D., Quigley, M., Snyder, D., Christiansen, R., Johnson, W., McLain, T., & Goodrich, M. A. (2005). *Autonomous Vehicle Technologies for Small Fixed-wing UAVs*. 19.

Beard, R. W. (2007). A class of flight trajectories for tracking ground targets with micro air vehicles. *2007 Mediterranean Conference on Control & Automation*, 1–6. https://doi.org/10.1109/MED.2007.4433657

Beard, R. W., & McLain, T. W. (2012). *Small unmanned aircraft: Theory and practice*. Princeton University Press.

Beard, R. W., McLain, T. W., Nelson, D. B., Kingston, D., & Johanson, D. (2006). Decentralized Cooperative Aerial Surveillance Using Fixed-Wing Miniature UAVs. *Proceedings of the IEEE*, *94*(7), 1306–1324. https://doi.org/10.1109/JPROC.2006.876930

Bengio, Y., Goodfellow, I., & Courville, A. (2016). *Deep learning (adaptive computation and machine learning series)*. The MIT Press.

Benowitz, D. (2021, May 30). The Rise of Open-Source Drones. *Drone Analyst - Insights & Research for the Commercial Done Industy*. https://droneanalyst.com/2021/05/30/rise-of-open-source-drones

Bernardeschi, C., Fagiolini, A., Palmieri, M., Scrima, G., & Sofia, F. (2019). ROS/Gazebo Based Simulation of Co-operative UAVs. In J. Mazal (Ed.), *Modelling and Simulation for Autonomous Systems* (Vol. 11472, pp. 321–334). Springer International Publishing. https://doi.org/10.1007/978-3-030-14984-0_24

Bernardino, A., & Cruz, G. (2015). *Image saliency applied to infrared images for unmanned maritime monitoring*. Proceedings of the International Conference on Computer Vision Systems.

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv:2004.10934 [Cs, Eess]*. http://arxiv.org/abs/2004.10934

Borji, A., Cheng, M.-M., Hou, Q., Jiang, H., & Li, J. (2019). Salient object detection: A survey. *Computational Visual Media*, *5*(2), 117–150. https://doi.org/10.1007/s41095-019-0149-9

Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2017). FreezeOut: Accelerate Training by Progressively Freezing Layers. *ArXiv:1706.04983 [Cs, Stat]*. http://arxiv.org/abs/1706.04983

Brownlee, J. (2018, July 3). Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. *Machine Learning Mastery*. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Brownlee, J. (2019, February 27). How to use Learning Curves to Diagnose Machine Learning Model Performance. *Machine Learning Mastery*. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Burdziakowski, P. (2018). UAV Design and Construction for Real Time Photogrammetry and Visual Navigation. *2018 Baltic Geodetic Congress (BGC Geomatics)*, 368–372. https://doi.org/10.1109/BGC-Geomatics.2018.00076

Caetano, J., Percin, M., Oudheusden, B., Remes, B., Wagter, C., Croon, G., & Visser, C. (2015). Error analysis and assessment of unsteady forces acting on a flapping wing micro air vehicle: Free flight versus wind-tunnel experimental methods. *Bioinspiration & Biomimetics*, *10*(5), 1–22.

Caetano, J., Visser, C., Croon, G., Remes, B., Wagter, C., Verboom, J., & Mulder, M. (2013). Linear Aerodynamic Model Identification of a Flapping Wing MAV Based on Flight Test Data. *International Journal of Micro Air Vehicles*, *5*(4), 273–286.

Caetano, J., Weehuizen, M., Visser, C., Croon, G., & Mulder, M. (2015). Rigid-body kinematics versus flapping kinematics of a flapping wing micro air vehicle. *Journal of Guidance, Control, and Dynamics*, *38*(12), 2257–2269.

Cepni, S., Atik, M. E., & Duran, Z. (2020). Vehicle Detection Using Different Deep Learning Algorithms from Image Sequence. *Baltic Journal of Modern Computing*, *8*(2). https://doi.org/10.22364/bjmc.2020.8.2.10

Chakure, A. (2021, March 1). All you need to know about YOLO v3 (You Only Look Once). *DEV Community*. https://dev.to/afrozchakure/all-you-need-to-know-about-yolo-v3-you-only-look-once-e4m

Chen, H., Chang, K., & Agate, C. S. (2009). *Tracking with UAV using tangent-plus-Lyapunov vector field guidance*. 10.

Choi, H., Geeves, M., Alsalam, B., & Gonzalez, F. (2016). Open source computer-vision based guidance system for UAVs on-board decision making. *2016 IEEE Aerospace Conference*, 1–5. https://doi.org/10.1109/AERO.2016.7500600

Collins Aerospace. (2022). *Piccolo^{TM} Autopilots*. Collins Aerospace. https://www.cloudcaptech.com/products/piccolo-autopilots

Cruz, G., & Bernardino, A. (2016). *Aerial detection in maritime scenarios us- ing convolutional neural networks*. Proceedings of the International Conference on Advanced Concepts for Intelligent Vision Systems.

Day, M. A., Clement, M. R., Russo, J. D., Davis, D., & Chung, T. H. (2015). Multi-UAV software systems and simulation architecture. *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, 426–435. https://doi.org/10.1109/ICUAS.2015.7152319

Dobrokhodov, V. N., Kaminer, I. I., Jones, K. D., & Ghabcheloo, R. (2008). Vision-Based Tracking and Motion Estimation for Moving Targets Using Unmanned Air Vehicles. *Journal of Guidance, Control, and Dynamics*, *31*(4), 907–917. https://doi.org/10.2514/1.33206

Dorrer, G., Koriukin, M., Yushkova, S., & Sviridova, L. (2019). Vehicle detection in aerial images. *IOP Conference Series: Earth and Environmental Science*, *315*, 022014. https://doi.org/10.1088/1755-1315/315/2/022014

Erdelj, M., Saif, O., Natalizio, E., & Fantoni, I. (2019). UAVs that fly forever: Uninterrupted structural inspection through automatic UAV replacement. *Ad Hoc Networks*, *94*, 101612. https://doi.org/10.1016/j.adhoc.2017.11.012

Erdos, D., & Watkins, S. E. (2008). UAV Autopilot Integration and Testing. *2008 IEEE Region 5 Conference*, 1–6. https://doi.org/10.1109/TPSD.2008.4562731

Ernst Dickmanns & Birger Mysliwetz. (1992). Recursive 3-D Road and Relative Ego-State Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *14*(2), 199–213.

Ezat, W. A., Dessouky, M. M., & Ismail, N. A. (2021). Evaluation of Deep Learning YOLOv3 Algorithm for Object Detection and Classification. *Menoufia Journal of Electronic Engineering Research*, *30*(1), 52–57. https://doi.org/10.21608/mjeer.2021.146237

Felix, L., Gomes, A., & Suleman, A. (2013). *Topology optimization of a wing including self-weight load*. Proceedings of the 54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference.

Felix, L., Suleman, A., & Gomes, M. (2012). *Integration of structural topology optimization in an MDO framework*. Proceedings of the 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.

Ferreira, F., Félix, L., Oliveira, T., & Franco, V. (2019). *Projeto conceptual de uma aeronave de asa fixa com descolagem vertical*. Academia da Força Aérea Portuguesa.

Figueiredo, D. L. (2014). *Autopilot and Ground Control Station for UAV*. 86.

Frew, E. W., Lawrence, D. A., Dixon, C., Elston, J., & Pisano, W. J. (2007). Lyapunov Guidance Vector Fields for Unmanned Aircraft Applications. *2007 American Control Conference*, 371–376. https://doi.org/10.1109/ACC.2007.4282974

Fu, C.-Y., Liu, W., Ranga, A., Tyagi, A., & Berg, A. C. (2017). DSSD: Deconvolutional Single Shot Detector. *ArXiv:1701.06659 [Cs]*. http://arxiv.org/abs/1701.06659

Fuller, B., Kok, J., Kelson, N., & Gonzalez, F. (2014). Hardware Design and Implementation of a MAVLink Interface for an FPGA-Based Autonomous UAV Flight Control System. *Proceedings of the 16th Australasian Conference on Robotics and Automation 2014*, 6.

Gad, A. (2020). Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall. *PaperspaceBlog*. https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/

Gandhi, R. (2018, July 9). R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. *Towards Data Science*. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

Girshick, R. (2015). *Fast R-CNN*. 1440–1448.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *ArXiv:1311.2524 [Cs]*. http://arxiv.org/abs/1311.2524

Gonçalves, P., Sobral, J., & Ferreira, L. (2016). Reliability database for unmanned aerial vehicles based on morphological analysis. *The Aeronautical Journal*, *120*, 1262–1274.

Gonçalves, P., Sobral, J., & Ferreira, L. (2017). Establishment of an ini- tial maintenance program for UAVs based on reliability principles. *Aircraft Engineering and Aerospace Technology*, *89*(1), 65–75.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *ArXiv:1512.03385 [Cs]*. http://arxiv.org/abs/1512.03385

Huang, T. (1996). Computer Vision: Evolution and Promise. *CERN European Organization for Nuclear Research-Reports-CERN*, 21–26.

Hui, J. (2018, March 18). Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3. *Towards Data Science*. https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

Husby, R. (2005). *Path generation tactics for a uav following a moving target.* [Master Thesis, University of Washington]. https://apps.dtic.mil/sti/pdfs/ADA434165.pdf

Ju, M., Luo, H., & Wang, Z. (2020). An improved YOLO V3 for small vehicles detection in aerial images. *2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, 1–5. https://doi.org/10.1145/3446132.3446188

Kang Liu, & Mattyus, G. (2015). Fast Multiclass Vehicle Detection on Aerial Images. *IEEE Geoscience and Remote Sensing Letters*, *12*(9), 1938–1942. https://doi.org/10.1109/LGRS.2015.2439517

Kanistras, K., Martins, G., Rutherford, M., & Valavanis, K. (2015). Survey of Unmanned Aerial Vehicles (UAVs) for TrafficMonitoring. *Handbook of Unmanned Aerial Vehicles*, 2643–2666.

Kathuria, A. (2018, April 23). What's new in YOLO v3? *Towards Data Science*. https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

Kim, C. E., Oghaz, M. M. D., Fajtl, J., Argyriou, V., & Remagnino, P. (2019). A Comparison of Embedded Deep Learning Methods for Person Detection. *ArXiv:1812.03451 [Cs]*. http://arxiv.org/abs/1812.03451

Kim, H.-M. & Lee, D. W. (2019). Integrated simulation environment for heterogeneous unmanned vehicles using ROS and Pixhawk. *Journal of the Korean Society for Aviation and Aeronautics*, *27*(3), 20.

Kim, Y., & Bang, H. (2019). Introduction to Kalman filter and its applications. *Introduction and Implementation of the Kalman Filter*. https://www.intechopen.com/chapters/63164

Kingma, D. P., & Ba, J. (2017). Adam: A Method for Stochastic Optimization. *ArXiv:1412.6980 [Cs]*. http://arxiv.org/abs/1412.6980

Kingston, D., Beard, R. W., & Holt, R. S. (2008). Decentralized Perimeter Surveillance Using a Team of UAVs. *IEEE Transactions on Robotics*, *24*(6), 1394–1404. https://doi.org/10.1109/TRO.2008.2007935

Klausen, K., Fossen, T. I., & Johansen, T. A. (2017). Nonlinear Control with Swing Damping of a Multirotor UAV with Suspended Load. *Journal of Intelligent & Robotic Systems*, *88*(2–4), 379–394. https://doi.org/10.1007/s10846-017-0509-6

Koenig, N., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, *3*, 2149–2154. https://doi.org/10.1109/IROS.2004.1389727

Kortunov, V. I., Mazurenko, O. V., Gorbenko, A. V., Mohammed, W., & Hussein, A. (2015). Review and comparative analysis of mini- and micro-UAV autopilots. *2015 IEEE International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*, 284–289. https://doi.org/10.1109/APUAVD.2015.7346622

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Kuwata, Y., & How, J. (2004, August 16). Three Dimensional Receding Horizon Control for UAVs. *AIAA Guidance, Navigation, and Control Conference and Exhibit*. AIAA Guidance, Navigation, and Control Conference and Exhibit, Providence, Rhode Island. https://doi.org/10.2514/6.2004-5144

Label Studio. (2022, May 26). *Open Source Data Labeling Tool*. Label Studio. https://labelstud.io/

LambDrive. (2016a). *Firmware—On-board Flight Control Software for Pixhawk Family of Controllers*. LambDrive. https://www.lambdrive.com/depot/Robotics/Controller/Firmware/index.html

LambDrive. (2016b). *Ground Control Station*. LambDrive. https://www.lambdrive.com/depot/Robotics/Controller/Ground-Control-Station/

Lee, J., Huang, R., Vaughn, A., Xiao, X., Hedrick, J. K., Zennaro, M., & Sengupta, R. (2003). *Strategies of Path-Planning for a UAV to Track a Ground Vehicle*. *2003*, 6.

Li, Z., Hovakimyan, N., Dobrokhodov, V., & Kaminer, I. (2010). Vision-based target tracking and motion estimation using a small UAV. *49th IEEE Conference on Decision and Control (CDC)*, 2505–2510. https://doi.org/10.1109/CDC.2010.5718149

Lin, H.-Y., Tu, K.-C., & Li, C.-Y. (2020). VAID: An Aerial Image Dataset for Vehicle Detection and Classification. *IEEE Access*, *8*, 212209–212219. https://doi.org/10.1109/ACCESS.2020.3040290

Lin, Y. (2018, January 22). *ROS 2 vs. ROS 1: Which One Is Better For Me?* https://www.theconstructsim.com/infographic-ros-1-vs-ros-2-one-better-2/

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. *ArXiv:1512.02325 [Cs]*, *9905*, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

Long, X., Deng, K., Wang, G., Zhang, Y., Dang, Q., Gao, Y., Shen, H., Ren, J., Han, S., Ding, E., & Wen, S. (2020). PP-YOLO: An Effective and Efficient Implementation of Object Detector. *ArXiv:2007.12099 [Cs]*. http://arxiv.org/abs/2007.12099

López, J., Royo, P., Pastor, E., Barrado, C., & Santamaria, E. (2007). A middleware architecture for unmanned aircraft avionics. *Proceedings of the 8th ACM/IFIP/USENIX International Conference on Middleware - Middleware '07*, 1. https://doi.org/10.1145/1377943.1377962

Lu, J., Ma, C., Xing, X., Zhang, Y., wang, Z., & Xu, J. (2018). A Vehicle Detection Method for Aerial Image Based on YOLO. *Journal of Computer and Communications*, *6*(11). https://www.scirp.org/journal/paperinformation.aspx?paperid=88545

Manathara, J. G., Sujit, P. B., & Beard, R. W. (2011). Multiple UAV Coalitions for a Search and Prosecute Mission. *Journal of Intelligent & Robotic Systems*, *62*(1), 125–158. https://doi.org/10.1007/s10846-010-9439-2

Mandal, M., Shah, M., Meena, P., Devi, S., & Vipparthi, S. K. (2020). AVDNet: A Small-Sized Vehicle Detection Network for Aerial Visual Data. *IEEE Geoscience and Remote Sensing Letters*, *17*(3), 494–498. https://doi.org/10.1109/LGRS.2019.2923564

Mandal, M., Shah, M., Meena, P., & Vipparthi, S. K. (2019). SSSDET: Simple Short and Shallow Network for Resource Efficient Vehicle Detection in Aerial Scenes. *2019 IEEE International Conference on Image Processing (ICIP)*, 3098–3102. https://doi.org/10.1109/ICIP.2019.8803262

Mao, Q.-C., Sun, H.-M., Liu, Y.-B., & Jia, R.-S. (2019). Mini-YOLOv3: Real-Time Object Detector for Embedded Applications. *IEEE Access*, *7*, 133529–133538. https://doi.org/10.1109/ACCESS.2019.2941547

Marques, J., Bernardino, A., Cruz, G., & Bento, M. (2014). *An algorithm for the detection of vessels in aerial images*. Proceedings of the 11th IEEE International Conference on Advanced Video and Signal Based Surveillance.

McLean, D. (1990). Automatic Flight Control Systems. *Prentice Hall*.

Melita, C., Longo, D., Muscato, G., & Giudice, G. (2015). Measurement and Exploration in Volcanic Environments. *Handbook of Unmanned Aerial Vehicles*, *2667–2690*.

Merino, L., Dios, J., & Ollero, A. (2015). Cooperative Unmanned Aerial Systems for Fire Detection,Monitoring, and Extinguishing. *Handbook of Unmanned Aerial Vehicles*, 2693–2721.

Mohamed, N., Al-Jaroodi, J., Jawhar, I., & Lazarova-Molnar, S. (2013). Middleware requirements for collaborative unmanned aerial vehicles. *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, 1051–1060. https://doi.org/10.1109/ICUAS.2013.6564794

Mohamed, N., Al-Jaroodi, J., Jawhar, I., & Lazarova-Molnar, S. (2014). A Service-Oriented Middleware for Building Collaborative UAVs. *Journal of Intelligent & Robotic Systems*, *74*(1–2), 309–321. https://doi.org/10.1007/s10846-013-9942-3

Muehlemann, A. (2022, April 8). *TrainYourOwnYolo*. Github. https://github.com/AntonMu/TrainYourOwnYOLO

Natarajan, G. (2001). Ground Control Stations for Unmanned Air Vehicles. *DEF SCI J*, *51*, 9.

Nguyen, N.-D., Do, T., Ngo, T. D., & Le, D.-D. (2020). An Evaluation of Deep Learning Methods for Small Object Detection. *Journal of Electrical and Computer Engineering*, *2020*, 1–18. https://doi.org/10.1155/2020/3189691

NVidia Developer. (2022). Jetson TX2 Module. https://developer.nvidia.com/embedded/jetson-tx2

Oliveira, T., & Encarnação, P. (2013). Ground Target Tracking Control System for Unmanned Aerial Vehicles. *Journal of Intelligent & Robotic Systems*, *69*(1–4), 373–387. https://doi.org/10.1007/s10846-012-9719-0

Oliveira, T., Encarnação, P., & Aguiar, A. (2013). *Moving path following for autonomous robotic vehicles*. Proceedings of the European Control Conference (ECC).

Oliveira, T., Encarnação, P., & Aguiar, A. (2016a). *A convoy protection strategy using the moving path following method*. Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS).

Oliveira, T., Encarnação, P., & Aguiar, A. (2016b). Moving path following for unmanned aerial vehicles with applications to single and multiple target tracking problems. *IEEE Transactions on Robotics*, *32*(5), 1062–2078.

Oliveira, T., Encarnação, P., & Aguiar, A. (2017). *Three dimensional moving path following for fixed-wing unmanned aerial vehicles*. Accepted for publication in the Proceedings of the IEEE International Conference on Robotics and Automation.

OpenCV team. (2022). *OpenCV*. OpenCV. https://opencv.org/

Pastor, E., Lopez, J., & Royo, P. (2006). A Hardware/Software Architecture for UAV Payload and Mission Control. *2006 Ieee/Aiaa 25TH Digital Avionics Systems Conference*, 1–8. https://doi.org/10.1109/DASC.2006.313738

Pedoeem, J., & Huang, R. (2018). YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers. *ArXiv:1811.05588 [Cs]*. http://arxiv.org/abs/1811.05588

Pedro Mendes, Luís Félix, Tiago Oliveira, & Vasco Franco. (2021). *Design of a Fixed-Wing Tilt-Rotor Quadcopter Class I Mini Unmanned Aircraft*. Academia da Força Aérea Portuguesa.

Pokhrel, N. (2018). *Drone Obstacle Avoidance and Navigation using Artificial Intelligence*. 114.

Polo, J., Hornero, G., Duijneveld, C., García, A., & Casas, O. (2015). Design of a low-cost Wireless Sensor Network with UAV mobile node for agricultural applications. *Computers and Electronics in Agriculture*, *119*, 19–32. https://doi.org/10.1016/j.compag.2015.09.024

PX4 User Guide. (n.d.-a). PX4 Autopilot. *PX4 Autopilot*. Retrieved 26 June 2021, from https://px4.io/

PX4 User Guide. (n.d.-b). *Using DroneKit to communicate with PX4* [PX4 Autopilot]. Retrieved 26 June 2021, from https://docs.px4.io/master/en/robotics/dronekit.html

PX4 User Guide. (2021a, January 31). *Offboard Mode*. PX4 Autopilot. https://docs.px4.io/v1.12/en/flight_modes/offboard.html

PX4 User Guide. (2021b, March 4). *ROS (Robot Operating System)*. PX4 Autopilot. https://docs.px4.io/v1.12/en/ros/

PX4 User Guide. (2021c, March 17). FC and Companion Computer. *PX4 Autopilot*. https://docs.px4.io/master/en/concept/px4_systems_architecture.html

PX4 User Guide. (2021d, June 3). *Plane/Fixed Wing*. PX4 Autopilot. https://docs.px4.io/v1.12/en/simulation/gazebo_vehicles.html

PX4 User Guide. (2021e, June 22). *ROS 2 User Guide (PX4-ROS 2 Bridge)*. PX4 Autopilot. https://docs.px4.io/v1.12/en/ros/ros2_comm.html

PX4 User Guide. (2021f, October 17). *PX4 Architectural Overview* [PX4 Autopilot]. PX4 User Guide. https://docs.px4.io/master/en/concept/architecture.html

PX4 User Guide. (2022a, February 16). *MAVLink Messaging*. PX4 Autopilot. https://docs.px4.io/master/en/middleware/mavlink.html

PX4 User Guide. (2022b, May 5). *ROS with Gazebo Simulation*. PX4 Autopilot. https://docs.px4.io/master/en/simulation/ros_interface.html

PX4 User Guide. (2022c, June 17). *Simulation*. PX4 Autopilot. https://docs.px4.io/v1.12/en/simulation/

QGroundControl. (2019). QGroundControl. *QGroundControl*. http://qgroundcontrol.com/

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). *ROS: an open-source Robot Operating System*. 6.

Quintero, S. A. P., & Hespanha, J. P. (2014). Vision-based target tracking with a small UAV: Optimization-based control strategies. *Control Engineering Practice*, *32*, 28–42. https://doi.org/10.1016/j.conengprac.2014.07.007

Ramirez-Atencia, C., & Camacho, D. (2018). Extending QGroundControl for Automated Mission Planning of UAVs. *Sensors*, *18*(7), 2339. https://doi.org/10.3390/s18072339

Raspberry Pi Foundation. (n.d.). Raspberry Pi Foundation. Retrieved 23 May 2022, from https://www.raspberrypi.org/

Razakarivony, S., & Jurie, F. (2016). Vehicle detection in aerial imagery: A small target detection benchmark. *Journal of Visual Communication and Image Representation*, *34*, 187–203. https://doi.org/10.1016/j.jvcir.2015.11.002

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *ArXiv:1506.02640 [Cs]*. http://arxiv.org/abs/1506.02640

Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. *ArXiv:1612.08242 [Cs]*. http://arxiv.org/abs/1612.08242

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *ArXiv:1804.02767 [Cs]*. http://arxiv.org/abs/1804.02767

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

Rizzoli, A. (2022, March 15). Mean Average Precision (mAP) Explained: Everything You Need to Know. *V7labs*. https://www.v7labs.com/blog/mean-average-precision

Rosebrock, A. (2021, January 19). *Image Making with OpenCV*. Pyimagesearch. https://pyimagesearch.com/2021/01/19/image-masking-with-opencv/

ROS.org. (2018, March 3). *MAVROS*. ROS.Org. http://wiki.ros.org/mavros

Sabikan, S., & Nawawi, S. W. (2016). *Open-Source Project (OSPs) Platform for Outdoor Quadcopter*. *24*(1), 16.

Schwartz, C. E. (1990). A Radar for Unmanned Air Vehicles. *The Linclon Laboratory Journal*, *3*(1), 26.

Shin, D. (2019). *Nonlinear Disturbance Observer Based Path Following for a Small Fixed Wing UAV* [Master's Thesis, Ulsan National Institute of Science and Technology]. https://scholar.google.pt/scholar?q=Nonlinear+Disturbance+Observer+Based+Path+Following+for+a+Small+Fixed+Wing+UAV&hl=pt-PT&as_sdt=0&as_vis=1&oi=scholart

Silva, H. & Oliveira, T. (2017). *Avaliação e comparação de alternativas de pilotos automáticos para UAS*. Academia da Força Aérea Portuguesa.

Silva, R., Félix, L. & Oliveira, T. (2020). *Projeto Conceptual de uma Aeronave Pequena de Baixo Custo para Aplicações em Controlo Cooperativo*. Academia da Força Aérea Portuguesa.

Skulstad, R., Syversen, C., Merz, M., Sokolova, N., Fossen, T., & Johansen, T. (2015). Autonomous net recovery of fixed- wing UAV with single-frequency carrier-phase differential GNSS. *IEEE Aerospace and Electronic Systems Magazine*, *30*(5), 18–27. https://doi.org/10.1109/MAES.2015.7119821

Sørensen, L., Jacobsen, L., & Hansen, J. (2017). Low Cost and Flexible UAV Deployment of Sensors. *Sensors*, *17*(12), 154. https://doi.org/10.3390/s17010154

Sujit, P. B., & Beard, R. (2008). Multiple UAV exploration of an unknown region. *Annals of Mathematics and Artificial Intelligence*, *52*(2–4), 335–366. https://doi.org/10.1007/s10472-009-9128-7

Supeshala, C. (2021, August 23). YOLO v4 or YOLO v5 or PP-YOLO? *Towards Data Science*. https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109

Süzen, Ali, A., Duman, B., & Şen, B. (2020). *Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn.* 1–5. https://ieeexplore.ieee.org/document/9152915

Taylor, C. J., Kosecka, J., Blasi, R., & Malik, J. (1999). A Comparative Study of Vision-Based Lateral Control Strategies for Autonomous Highway Driving. *Int. J. Robotics Research*, *18*(5), 18.

The Robotics Back-End. (n.d.). *ROS1 vs ROS2, Practical Overview For ROS Developers*. Retrieved 26 June 2021, from https://roboticsbackend.com/ros1-vs-ros2-practical-overview/

Thuan, D. (2021). *EVOLUTION OF YOLO ALGORITHM AND YOLOV5: THE STATE-OF-THE-ART OBJECT DETECTION ALGORITHM*. 61.

Torrico, R. A. (2020). *Circuit Cellar 357 April 2020 (PDF)*. CC-Webshop. https://cc-webshop.com/products/circuit-cellar-357-april-2020-pdf

Umair Iqbal, Syed Irtiza Ali Shah, Zia Ur Rehman, & Moshsin Jmail. (2015). *Long Range Radio Modules for Model Unmanned Aerial Systems: A Short Comparison for Disaster Relief Applications*. 274–283.

Van Breda. (1995). *Operator performance in multi Maritime Unmanned Air Vehicle control*. 22.

Wang, S., Zhen, Z., Jiang, J., & Wang, X. (2016). Flight Tests of Autopilot Integrated with Fault-Tolerant Control of a Small Fixed-Wing UAV. *Mathematical Problems in Engineering*, *2016*, 1–7. https://doi.org/10.1155/2016/2141482

Wang, X., Zhu, H., Zhang, D., Zhou, D., & Wang, X. (2014). Vision-Based Detection and Tracking of a Mobile Ground Target Using a Fixed-Wing UAV. *International Journal of Advanced Robotic Systems*, *11*(9), 156. https://doi.org/10.5772/58989

Welch, G., & Bishop, G. (1995). *An Introduction to the Kalman Filter*. 81.

Willis, M. (1999). *Proportional-Integral-Derivative Control* [Dept. of Chemical and Process Engineering University of Newcastle]. educypedia.karadimov.info/library/PID.pdf

Wise, R., & Rysdyk, R. (2006, August 21). UAV Coordination for Autonomous Target Tracking. *AIAA Guidance, Navigation, and Control Conference and Exhibit*. AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, Colorado. https://doi.org/10.2514/6.2006-6453

Xu, X., Zhang, X., Yu, B., Hu, X. S., Rowen, C., Hu, J., & Shi, Y. (2018). DAC-SDC Low Power Object Detection Challenge for UAV Applications. *ArXiv:1809.00110 [Cs]*. http://arxiv.org/abs/1809.00110

Yao, P., Wang, H., & Ji, H. (2017). Gaussian mixture model and receding horizon control for multiple UAV search in complex environment. *Nonlinear Dynamics*, *88(2)*, 903–919.

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). mixup: Beyond Empirical Risk Minimization. *ArXiv:1710.09412 [Cs, Stat]*. http://arxiv.org/abs/1710.09412

Zhu, S., & Wang, D. (2012). Adversarial Ground Target Tracking Using UAVs with Input Constraints. *Journal of Intelligent & Robotic Systems*, *65*(1–4), 521–532. https://doi.org/10.1007/s10846-011-9574-4

Zhu, S., Wang, D., & Low, C. B. (2013). Ground Target Tracking Using UAV with Input Constraints. *Journal of Intelligent & Robotic Systems*, *69*(1–4), 417–429. https://doi.org/10.1007/s10846-012-9737-y

ANNEXES

ANNEX A – Comparison of Different Detectors with Specific Datasets

Table A-1. Comparison of the speed and accuracy of different object detectors on the MS-COCO (Retrieved from Long et al., 2020).

Method	Backbone	FPS	mAP
YOLO v3 + ASFF	Darknet-53	29.4	43.9%
YOLO v4	CSPDarknet-53	62	43.5%
PP-YOLO	ResNet50-vd-dcn	72.9	45.2%

Table A-2. Results on Pascal VOC2007 test (Retrieved from Liu et al., 2016).

Method	FPS (Titan X)	mPA
SSD 300	59	74.3%
Faster R-CNN (VGG16)	7	73.2%
YOLOv1 (VGG16)	45	63.4%

Table A-3. Comparison detection performance in terms of mean average precision (mAP) of several state-of-art detectors (Retrieved from Mandal et al., 2019).

Method	VEDAI	DLR	DOTA	Complete
YOLOv2_416	9.08	9.61	33.36	28.86
YOLOv2_608	25.12	26.81	47.45	48.04
Faster R-CNN	34.82	20.04	42.29	38.02
YOLOv3_416	32.07	52.11	74.46	70.35
YOLOv3_608	38.98	54.49	76.60	75.21
RetinaNet	43.47	54.77	73.77	71.28
YOLOv3-tiny_416	11.10	26.42	47.88	46.73
YOLOv3_tiny_608	31.73	39.74	65.89	59.17
SSSDet	**45.97**	**58.25**	**79.52**	**77.22**

Table A-4. Computation and space complexity comparison of the SSDNet, YOLOv3 and YOLOv3-tiny with input layer size=608x608x3. The FPS is computed over a CPU system (Retrieved from Mandal et al., 2019).

Method	FPS
YOLOv3	0.04
YOLOv3-tiny	**0.40**
SSSDet	0.15

Table A-5. Comparison of SSD512, YOLOv3 and RetinaNet methods in the Dorrer et al. (2019) dataset (Retrieved from Dorrer et al., 2019).

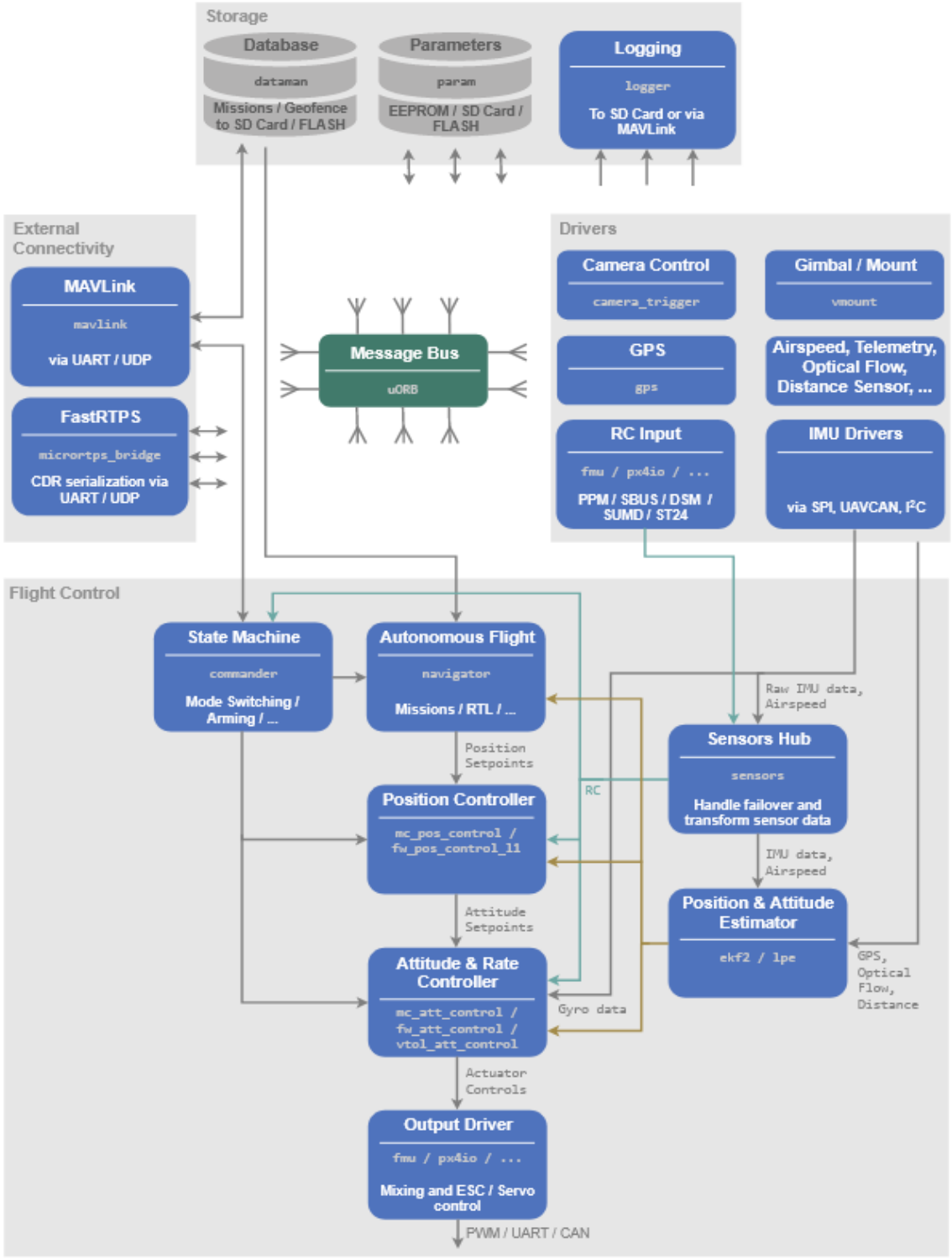
Method	Recall rate	Precision rate
SSD512	21.19%	6.52%
YOLOv3	88.38%	58.36%
RetinaNet	**89.44%**	**64.61%**

Table A-6. Comparison Detection Performance of ADVNet and several state-of-art detectors (Retrieved from Mandal et al., 2020).

Method	VEDAI	DLR-3K	DOTA	Complete
Coupled R-CNN	12.04	11.74	25.60	19.66
YOLOv2_416x416	9.08	9.61	33.36	28.86
YOLOv2_608x608	25.12	26.81	47.45	48.04
Faster R-CNN	34.82	20.04	42.29	38.02
YOLOv2_416x416	32.07	52.11	74.46	70.35
YOLOv2_608x608	38.98	54.49	76.60	75.21
RetinaNet43.47	43.47	54.77	73.77	71.28
AVDNet	**51.95**	**56.24**	**79.65**	**80.02**

ANNEX B – High-Level Software Architecture

Figure B-1. PX4 Flight Stack High-Level Software Architecture (Retrieved from PX4 User Guide, 2021d).



ANNEX C – Detailed Information regarding PI Controller Simulation

Table C-1. Detailed information regarding the PI controller validation in climb and descent scenarios.

			Climb	Descent
Parameteres	Initial altitude (meters)		0	100
	Final altitude (meters)		100	50
	Turn radius (meters)		150	150
	Airspeed Setpoint (meters/second)		15	15
Results	Time to reach 100 meters (seconds)		97.26	104.89
	Time to converge to 100 meters and 15 m/s (seconds)		137.0	137.79
	Settling time (seconds)		39.80	32.89
	Altitude (meters) maintained after reaching 100 m	Average (meters)	100.15	50.09
		Relative error (meters)	0.149	0.002
		Maximum (meters)	101.38	50.83
		Minimum (meters)	99.51	49.45
	Pitch (radians) maintained after reaching 100 m	Average (meters)	-0.05	-0.04
		Maximum (meters)	0.13	0.14
		Minimum (meters)	-0.20	-0.20
	Thrust (percentage/100) maintained after reaching 100 m	Average (meters)	0.23	0.23
		Maximum (meters)	0.53	0.52
		Minimum (meters)	0.05	0.05
	Airspeed (meters/second) maintained after reaching 100 m	Average (meters)	15.33	15.35
		Relative error (meters)	0.022	0.023
		Maximum (meters)	16.44	17.11
		Minimum (meters)	14.38	14.60
	Thrust (percentage/100) during climb	Average (meters)	0.69	0.07
		Maximum (meters)	1	0.28
		Minimum (meters)	0.25	0.05
	Airspeed (meters/second) during climb	Average (meters)	15.23	16.14
		Maximum (meters)	19.02	16.33
		Minimum (meters)	0.82	15.10

ANNEX D – Detailed Information regarding the Geolocation Validation with the Target Moving

Table D- 1. Detailed information regarding the path from the target.

Path settings

1st Section	Waypoints (X_I, Y_I) (meters)	(0,0) → (50,50)
	Distance (meters)	70.71
	Time between waypoints (seconds)	30
	Velocity (meters/second)	2.36
2nd Section	Waypoints (X_I, Y_I) (meters)	(50,50) → (-50,50)
	Distance (meters)	100
	Time between waypoints (seconds)	30
	Velocity (meters/second)	3.33
3rd Section	Waypoints (X_I, Y_I) (meters)	(-50,50) → (50,-50)
	Distance (meters)	141.43
	Time between waypoints (seconds)	20
	Velocity (meters/second)	7.07
4th Section	Waypoints (X_I, Y_I) (meters)	(50,-50) → (-50,-50)
	Distance (meters)	100
	Time between waypoints (seconds)	20
	Velocity (meters/second)	5
5th Section	Waypoints (X_I, Y_I) (meters)	(-50,-50) → (0,0)
	Distance (meters)	70.71
	Time between waypoints (seconds)	30
	Velocity (meters/second)	2.36

Table D- 2. Camera. UAV and target settings used in this simulation.

Parameters maintained during simulation

Camera settings	Pan	68.75°
	Tilt	-36.31°
	Sensor width/height (pixels)	640
	Sensor focal distance (pixels)	320
UAV settings	Height (meters)	100
	Airspeed (meters/second)	15
	Turn radio (meters)	150
	Loiter	Clockwise
Target settings	Position	See Table D-1
	Dimension (meters)	12 x 9
	Color	Red

ANNEX E – Detailed Information regarding the Kalman Filter Validation with the Target at a Constant Position

Table E- 1. Camera. UAV and target settings used in this simulation.

Parameters maintained during simulation

Camera settings	Pan	68.75º
	Tilt	-36.31º
	Sensor width/height (pixels)	640
	Sensor focal distance (pixels)	320
UAV settings	Height (meters)	100
	Airspeed (meters/second)	15
	Turn radio (meters)	150
	Loiter	Clockwise
Target settings	Position (meters)	X_I=-10, Y_I=-30
	Dimension (meters)	12 x 9
	Color	Red

ANNEX F – Detailed Information regarding the Closed-loop Validation with the Red Target

Table F- 1. Camera. UAV and target settings used in the simulation.

Parameters maintained during simulation

Camera settings	Pan	68.75º
	Tilt	-36.31º
	Sensor width/height (pixels)	640
	Sensor focal distance (pixels)	320
UAV settings	Height (meters)	100
	Airspeed (meters/second)	15
	Turn radio (meters)	150
	Loiter	Clockwise
	g_1	0.1
	g_2	0.00012
Target settings	Position	See Table F-2
	Dimension (meters)	12 x 9
	Color	Red

Table F- 2. Detailed information about the path of the target.

Path settings

1st Section	Waypoints (X_I, Y_I) (meters)	(0,0) → (0,200)
	Distance (meters)	200
	Time between waypoints (seconds)	90
	Velocity (meters/second)	2.22
2nd Section	Waypoints (X_I, Y_I) (meters)	(0,200) → (0,200)
	Distance (meters)	0
	Time between waypoints (seconds)	30
	Velocity (meters/second)	0
3rd Section	Waypoints (X_I, Y_I) (meters)	(0,200) → (-200,150)
	Distance (meters)	206.15
	Time between waypoints (seconds)	100
	Velocity (meters/second)	2.06
4th Section	Waypoints (X_I, Y_I) (meters)	(-200,150) → (-200,0)
	Distance (meters)	150
	Time between waypoints (seconds)	75
	Velocity (meters/second)	2
5 th Section	Waypoints (X_I, Y_I) (meters)	(-200,0) → (80,80)
	Distance (meters)	291.20
	Time between waypoints (seconds)	120
	Velocity (meters/second)	2.42
6 th Section	Waypoints (X_I, Y_I) (meters)	(80,80) → (0,-30)
	Distance (meters)	136.01
	Time between waypoints (seconds)	70
	Velocity (meters/second)	1.94
7 th Section	Waypoints (X_I, Y_I) (meters)	(0,-30) → (0,0)
	Distance (meters)	30
	Time between waypoints (seconds)	60
	Velocity (meters/second)	0.5

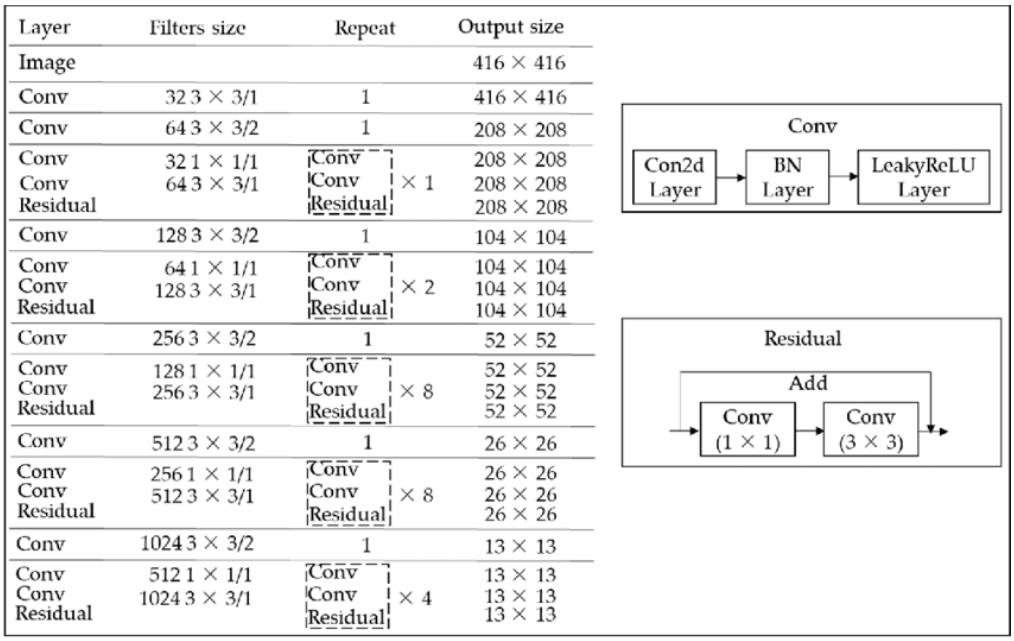
ANNEX G – Camera and UAV Settings used to Capture the Dataset Images

Table G- 1. Camera and UAV settings used to capture the dataset images. Tests from 1 to 6 were conducted in the Ground plane scenario displayed in Figure 45(c). Tests 7 and 8 were performed in the Runway scenario depicted in Figure 50(a).

Settings	Parameters		1º test	2º test	3º test	4º test	5º test	6º test	7º test	8º test
Camera	Tilt		-36.31°	-40.12°	-36.31°	-34.38°	-90°	-60°	-36.31°	-90°
	Pan		68.76°	90°	-68.76°	-90°	0°	0°	-90° to 90°	0°
	Resolution (pixel)		920 x 920	920 x 920	920 x 920	920 x 920	920 x 920	920 x 920	640 x 640 and 920 x 920	640 x 640 and 920 x 920
	HFOV		90°	90°	90°	90°	90°	90°	90°	90°
	Sensor focal distance (pixel)		460	460	460	460	460	460	320 and 460	320 and 460
UAV	Turn radius (meters)		100	100	100	90	75	60	60 to 100	60 to 100
	Altitude (meters)		75	60	50	30	60	45	30 to 75	30 to 75
	Type of flight	Loiter	Clockwise	Clockwise	Counterclockwise	Counterclockwise	-	-	Clockwise/ Counterclockwise	-
		Vertical passes	-	-	-	-	Vertical passes	Vertical passes	-	Vertical passes

ANNEX H – Architecture Diagram of YOLOv3

Figure H- 1. Architecture diagram of YOLOv3 (Retrieved from Cheng et al., 2020).



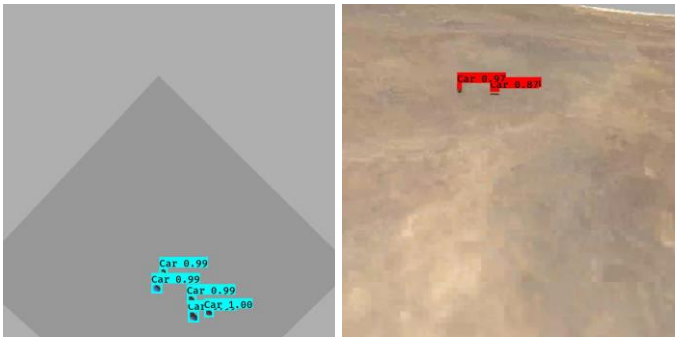
Layer	Filters size	Repeat	Output size
Image			416 × 416
Conv	32 3 × 3/1	1	416 × 416
Conv	64 3 × 3/2	1	208 × 208
Conv Conv Residual	32 1 × 1/1 64 3 × 3/1	Conv Conv Residual × 1	208 × 208 208 × 208 208 × 208
Conv	128 3 × 3/2	1	104 × 104
Conv Conv Residual	64 1 × 1/1 128 3 × 3/1	Conv Conv Residual × 2	104 × 104 104 × 104 104 × 104
Conv	256 3 × 3/2	1	52 × 52
Conv Conv Residual	128 1 × 1/1 256 3 × 3/1	Conv Conv Residual × 8	52 × 52 52 × 52 52 × 52
Conv	512 3 × 3/2	1	26 × 26
Conv Conv Residual	256 1 × 1/1 512 3 × 3/1	Conv Conv Residual × 8	26 × 26 26 × 26 26 × 26
Conv	1024 3 × 3/2	1	13 × 13
Conv Conv Residual	512 1 × 1/1 1024 3 × 3/1	Conv Conv Residual × 4	13 × 13 13 × 13 13 × 13

ANNEX I – Example of Four Images used in the Test Set

Figure I- 1. Example of four images used in the test set (marked with 1) and the corresponding detection images (marked with 2) from the four different worlds used in our dataset. Images (a) and (b) have a resolution of 920x920 pixels, and images (c) and (d) have a resolution of 640x640 pixels.

(a^1) (b^1)

(c^1) (d^1)



(a^2) (b^2)

(c^2) (d^2)

ANNEX J – Detailed Information regarding the Closed-loop Validation with the Vehicle at a Constant Position

Table J- 1. Parameters maintained during simulation from Camera, UAV and Target.

Parameters maintained during simulation

Camera settings	Pan	68.75º
	Tilt	-31.81º
	Sensor width/height (pixels)	640
	Sensor focal distance (pixels)	320
UAV settings	Height (meters)	40
	Airspeed (meters/second)	15
	Turn radius (meters)	100
	Loiter	Clockwise
	g_1	0.1
	g_2	0.00012
Target settings	Position (meters)	X_I=-30, Y_I=-10
	Dimension (meters)	1 x 1
	Car model	Lexus

ANNEX K – Detailed Information regarding the Closed-loop Validation with the Vehicle Moving

Table K- 1. Detailed information about the path with the target moving.

Path settings

1st Section	Waypoints (X_I, Y_I) (meters)	(30,10) → (600,1000)
	Distance (meters)	1142,4
	Time between waypoints (seconds)	400
	Velocity (meters/second)	2.8
2nd Section	Waypoints (X_I, Y_I) (meters)	(600,1000) → (1000,400)
	Distance (meters)	721.1
	Time between waypoints (seconds)	500
	Velocity (meters/second)	1.4
3rd Section	Waypoints (X_I, Y_I) (meters)	(1000,400) → (1200,800)
	Distance (meters)	447.2
	Time between waypoints (seconds)	200
	Velocity (meters/second)	2.2
4th Section	Waypoints (X_I, Y_I) (meters)	(1200,800) → (250,100)
	Distance (meters)	1180.1
	Time between waypoints (seconds)	350
	Velocity (meters/second)	3.4
5 th Section	Waypoints (X_I, Y_I) (meters)	(250,100) → (30,10)
	Distance (meters)	291.20
	Time between waypoints (seconds)	120
	Velocity (meters/second)	4.8

Table K- 2. Camera, UAV and target settings used in the simulation.

Parameters maintained during simulation

Camera settings	Pan	68.75º
	Tilt	-31.81º
	Sensor width/height (pixels)	640
	Sensor focal distance (pixels)	320
UAV settings	Height (meters)	40
	Airspeed (meters/second)	15
	Turn radio (meters)	100
	Loiter	Clockwise
	g_1	0.23
	g_2	0.00024
Target settings	Position	See Table K-1
	Dimension (meters)	1x1
	Color	Red

ANNEX L – YOLOv3 detailed information regarding the detection process, architecture and loss function

1. YOLO v3

While there is a handful of distinguishing object detection algorithms, as stated in the computer vision literature review, we use YOLO. More specifically in this research, YOLO v 3.

"You only look Once" (Redmon et al., 2016. p. 1) or YOLO is a family of deep learning models designed by Redmon et al. (2016) to perform fast object detection as a regression task. Redmon et al. (2016) frame object detection as a single neural network that predicts bounding boxes and class probabilities from an entire image with just one evaluation. As a result, the detection is extremely fast since the whole detection pipeline is a single network while maintaining high accuracy (Redmon et al., 2016).

1.1. YOLO Detection Process

This system starts by dividing the loaded figure into a S x S grid, and if the centre of an object matches a grid cell, that grid cell is assigned for detecting the object. Each grid cell predicts the bounding boxes and the related confidence (Kathuria, 2018).

The confidence reflects how sure the model is that the object is within that box and the accuracy that the model attributes itself by knowing that inside the box is the object that it predicts. If a grid cell is empty, the confidence score should be zero. Otherwise, the confidence value should be equivalent to the IoU between the predicted and ground truth boxes from the original dataset to find the most common shapes and sizes (Redmon et al., 2016).

Each bounding box includes 5 predictions, x, y, width, height and confidence. The (x, y) represents the coordinates of the box's centre relative to the grid cell's bound. The width and height are coherent to the whole image, and the confidence corresponds to the IoU between the predicted bounding box and any ground truth box (Redmon et al., 2016).

Furthermore, conditional class probabilities, $P_r = (Class|Object)$, is predicted by each grid cell that contains an object. Only one set of class probabilities per grid cell is predicted even if there are several bounding boxes. In the test, the conditional class probabilities are multiplied by the individual box confidence prediction as follows

$$P_r(Class|Object) * P_r(Object) * IOU_{pred}^{truth} = P_r(Class) * IOU_{pred}^{truth}. \quad (4.1)$$

The result is a class-specific confidence score for each box that comprises the probability of the class appearance in the box and how well the prediction box fits the object (Redmon et al., 2016).

1.2.YOLOv3 Architecture

YOLOv3 architecture, shown in Figure L-1, was inspired by other famous architectures, particularly ResNet and FPN (Feature Pyramid Network) (Chakure, 2021). YOLOv3 uses a variant of Darknet to perform feature extraction, which originally had 53 layer network trained on Imagenet with skip connection like ResNet and three predict heads like FPN. This enables YOLOv3 to process images at a different spatial scales. As a result, YOLOv3 has 53 convolutional layers from Darknet-53 and 53 more layers for detection (Redmon & Farhadi, 2018).

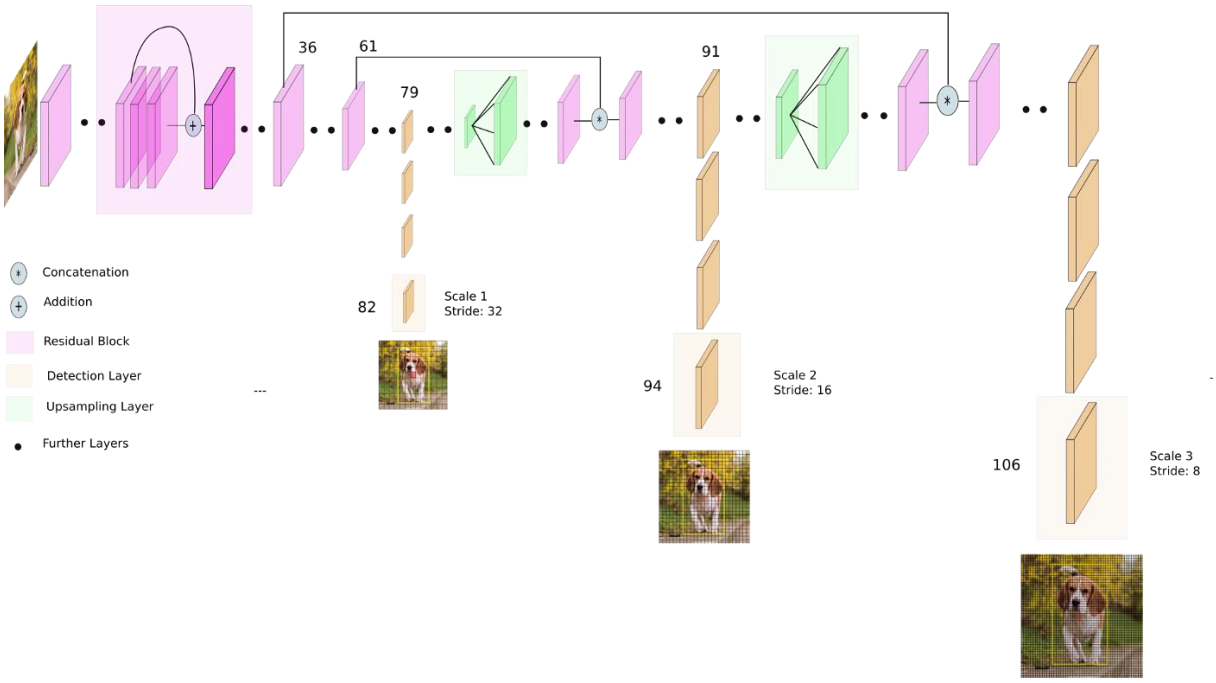


Figure L- 1. YOLOv3 network architecture diagram (Retrieved from Kathuria, 2018).

The network is framed with a bottleneck structure 1 x 1 followed by 3 x 3 convolution layers inside each residual block plus a skip connection (Thuan, 2021). All the convolutional layers in YOLOv3 are followed by a batch normalization layer and Leaky ReLU activation. There are no pooling layers; instead, there are convolutional layers with stride 2 used to downsample feature maps. Consequently, it prevents the loss of low-level features that the pooling layer would exclude. As a result, capturing low-level features improved the capacity to detect small objects (Kathuria, 2018) (detailed information regarding the YOLOv3 architecture is displayed in Figure H-1).

FPN is a method where a feature map gradually decreases in spatial dimension and increases again and then is concatenated with previous feature maps with corresponding sizes.

Concatenation is a method that gets more meaningful semantic data from the upsampled features and detailed information from the earlier feature map (Redmon & Farhadi, 2018).

YOLO detection is done by applying 1 x 1 kernel on feature maps of three different sizes at three distinct places of the network. The size of the detection kernel filter

$$X = (B \times (N + conf + C)), \tag{4.2}$$

where *X* corresponds to the size of the detection kernel, *B* represents the number of bounding boxes the filter can predict, variable *N* the number of bounding boxes attributes, variable *conf* the object confidence, and term *C* is the number of class predictions (Redmon & Farhadi, 2018).

The detection occurs at three different layers, 82nd, 94th and 106th coinciding with the three last residual blocks. So, in the 81st layer, the stride is 32, which means that if the image has dimensions 416 x 416 pixels, a feature map of 13 x 13 pixels will be obtained. As a result, one detection by the 1 x 1 detection Kernel gets a 13 x 13 x 255 pixels detection feature map. Following this logic, in the 94th layer, the detection feature map is 26 x 26 x 255 pixels, and in the 106th the detection feature map is 52 x 52 x 255 pixels (Mao et al., 2019).

Each scale has 3 anchor boxes, making 9 anchor boxes for the overall model generated using K-means clustering. The anchors are organized in descending order of dimension, so the three biggest anchors are for the first scale, the next three for the second scale and the last three anchors for the third scale (Kathuria, 2018).

Between the detection scales in Figure L-1, the feature map from the previous two layers is upsampled by 2 times. Then the feature map from earlier in the network is merged with the upsampled features using concatenation. Afterwards, the feature map is subjected to a convolutional set and outputs the prediction result. Finally, it connects to the next feature map (Redmon & Farhadi, 2018).

The reason for YOLOv3 to use three scales is to detect objects of the same class with different dimensions in the same image. Therefore, it uses a 16 x 16 pixels map to detect larger objects and 52 x 52 pixels to detect small objects (Kathuria, 2018).

After the image has passed through the three scales, a fusion of the three scale predictive outputs is performed, and a probability score filter runs out the anchors with low scores. Lastly, the Non-Maximum Suppression (NMS) takes the bounding box with the best objectness score in the given model, suppresses the other boxes. or performs an IOU of the bounding boxes (Mao et al., 2019).

1.3.Loss function

YOLO uses a sum-squared error between the predictions and the ground truth, to calculate loss. The loss function can be divided into classification loss, localization loss, and confidence loss (Hui, 2018).

First, the classification loss computes the loss of class probability. except for the $\mathbb{1}_i^{obj}$ term, which is '0' if no objects are present in the cell (Thuan, 2021). The equation is expressed through

$$classification_{loss} = \sum_{i=1}^{s^2} \mathbb{1}_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2, \tag{4.3}$$

where $\hat{p}_i(c)$ is the conditional class probability for the class c in cell i (Redmon et al., 2016).

Second, the localization mean-squared error measures the errors in the predicted bounding box locations and size through

$$localization_{loss} = \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \tag{4.4}$$

$$+\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right], \tag{4.5}$$

where $\mathbb{1}_{ij}^{obj}$ denotes 1 if jth bounding box in cell i is responsible for detecting the object; otherwise, 0 (Redmon et al., 2016). YOLO attributes the highest penalty (λ_{coord}) for the predictions from bounding boxes containing objects and the lowest (λ_{noobj}) for predictions without an object to avoid overwhelming the gradient of the cells containing the object, since it can lead to training divergence and model instability (Redmon et al., 2016).

The first term (4.4) computes the loss related to the deviation from the predicted bounding box position and the ground truth bounding box based on coordinates (x_{center}, y_{center}). Subsequently, the second term (4.5) calculates the square root of the difference between the width and height of the bounding box (Thuan, 2021).

Since the magnitude of the error in large boxes affects the equation less than in the small boxes, the second term takes the square root of width and height to the loss function and considers the deviation in terms of the size of the bounding box. Therefore, the small bounding boxes with little deviation should be more meaningful than the large ones. Moreover, to increase the bounding box accuracy, the term λ_{coord} is multiplied with the loss (Thuan, 2021).

At last, the confidence loss is composed of two terms and is represented by the following equation

$$confidence_{loss} = \sum_{i=1}^{s^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 \quad (4.6)$$

$$\lambda_{noobj} \sum_{i=1}^{s^2} \sum_{j=1}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2, \quad (4.7)$$

where $\hat{C}_i$ is the box confidence score of the box in cell i and $\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$ (Redmon et al., 2016).

The first term (4.6) represents the addition of squared error between the predicted confidence value and the ground truth for each bounding box in each cell. The second term (4.7) computes the mean-squared sum of cells that do not incorporate any bounding box. Furthermore, a regularization parameter, λ_{noobj}, is used to make this loss small because most boxes do not contain any object, so it causes a class imbalance problem (Thuan, 2021).

In YOLOv3. the (4.3), (4.6) and (4.7) terms of the equation, which are squared errors, are replaced by cross-entropy error terms. As a result, object confidence and class predictions in YOLOv3 are now predicted through logistic regression. The objectness value should be 1 if the preceding bounding box overlaps a ground truth object by more than any other bounding box (Kathuria, 2018).