

Anthony Oliveira Pinto

# **UAV Aerial Surveillance to Georeference *Aedes-Aegypti* Mosquito Breeding Grounds**

Master's Thesis presented to the Program of Postgraduate in Electrical Engineering (PPGEE) at the School of Engineering of the Federal University of Espírito Santo (UFES), as a partial requirement for obtaining the Master's Degree in Electrical Engineering, in the research field of Robotics, Control and Automation.

Universidade Federal do Espírito Santo – UFES

Departamento de Engenharia Eletrica

Programa de Pós-Graduação em Engenharia Eletrica – PPGEE

Supervisor: Prof. Dr. Mário Sarcinelli Filho

Co-supervisor: Prof. Dr. Patrick Marques Ciarelli

Vitória, ES - Brazil

January 31st, 2022

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

P659u Pinto, Anthony Oliveira, 1979-  
UAV aerial surveillance to georeference aedes-aegypti mosquito breeding grounds / Anthony Oliveira Pinto. - 2022.  
80 f. : il.

Orientador: Mário Sarcinelli Filho.  
Coorientador: Patrick Marques Ciarelli.  
Dissertação (Mestrado em Engenharia Elétrica) -  
Universidade Federal do Espírito Santo, Centro Tecnológico.

1. VANT. 2. YOLO. 3. CNN. 4. Detecção de objetos. 5. Vigilância aérea. 6. Visão computacional. I. Sarcinelli Filho, Mário. II. Ciarelli, Patrick Marques. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 621.3

---

ANTHONY OLIVEIRA PINTO

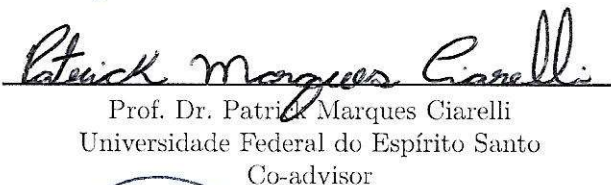
UAV AERIAL SURVEILLANCE TO GEOREFERENCE  
*Aedes-Aegypti* MOSQUITO BREEDING GROUNDS

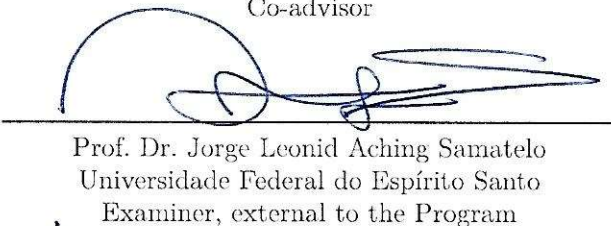
Master's Thesis presented to the Program of Postgraduate in Electrical Engineering (PPGEE) at the School of Engineering of the Federal University of Espírito Santo (UFES), as a partial requirement for obtaining the Master's Degree in Electrical Engineering, in the research field of Robotics, Control and Automation.

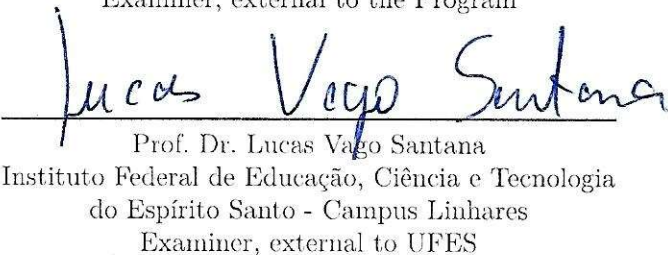
Approved in January 31st, 2022.

EXAMINATION BOARD:

  
Prof. Dr. Mário Sarcinelli Filho  
Universidade Federal do Espírito Santo  
Advisor

  
Prof. Dr. Patrick Marques Ciarelli  
Universidade Federal do Espírito Santo  
Co-advisor

  
Prof. Dr. Jorge Leonid Aching Samatelo  
Universidade Federal do Espírito Santo  
Examiner, external to the Program

  
Prof. Dr. Lucas Vago Santana  
Instituto Federal de Educação, Ciência e Tecnologia  
do Espírito Santo - Campus Linhares  
Examiner, external to UFES

*To my beloved wife Juliana. I am truly thankful for having you in my life.*

# Acknowledgements

First and foremost, I wish to thank my wife, Juliana Pereira Ribeiro, who has stood by me through all my travails, my absences, my ups, and downs. She gave me support and help, discussed ideas, and prevented several wrong turns. She also supported our family of two during much of my pursue studies.

I thank all my family, who were always present despite the distance, to my brother Geraldo and nephews/godsons Mauricio and Matheus of age 13 and 8, especially my parents, Benedito Geraldo Miglio Pinto and Eloisa Oliveira Pinto, for the love and support they always offered. I also thank my grandmother Marina Miglio and my father-in-law Jarbas Nunes Ribeiro, who has passed away and I could not say goodbye to because I lived in another country at the time.

I also thank my advisor, Prof. Dr. Mario Sarcinelli Filho for the opportunity offered, and for being an example of dedication and commitment to research, as well as my co-advisor, Prof. Dr. Patrick Marques Ciarelli, for the opportunity offered, and for being an example of dedication and commitment to research.

In addition, I thank the professors of the Postgraduate Program on Electrical Engineering of UFES, who, with great dedication, contributed to my training, as well my colleagues at the Intelligent Automation Laboratory (LAI), especially Mauro Sergio Mafra Moreira, Harrison Neves Marciano, Vinicius Pacheco Bacheti and Diego Bertolani, who shared with me the challenge of knowledge and professional growth. Besides them, I also thank all the people who directly or indirectly collaborated with the success of this work.

I thank FAPES - Fundação de Amparo à Pesquisa e Inovação do Espírito Santo for the financial support to the project in which this thesis is inserted, as well as for financing my personal participation, through the scholarship granted to me, which allowed me to focus exclusively in this work. I also thank the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), for its participation in the financial support of this project, as well as the Universidade Federal do Espírito Santo for supporting the development of this research.

To everyone that I didn't mention, after all there are many names, I will also always be grateful.

*Anthony Oliveira Pinto*

*A ship in harbor is safe, but that is not what ships are built for.*

—John A. Shedd

# Abstract

The annual number of cases of Dengue, Zika, and Chikungunya in Brazil are alarming. A way to minimize this problem is to identify and treat possible mosquito breeding grounds, such as swimming pools, untapped water tanks, etc. Focusing on creating a tool to help authorities in finding such breeding grounds, as a contribution to the fight against these mosquitoes, this thesis proposes a YOLO custom model to detect swimming pools in a certain neighborhood, from images captured by a quadrotor. Thus, it is proposed an object detection system based on a convolutional neural network, the YOLOv3 one, to detect swimming pools on images collected by the quadrotor when flying autonomously over such a neighborhood. A dataset was created, with high-resolution aerial bird's eye view images, with the desired objects annotated, to be used as a training data set for the YOLO CNN inner layers, with 150 images collected from high resolution photography websites. The evaluation of the classifier thus obtained occurred on a database containing 72 satellite images with different resolutions, at three different image scales, for two different locations, collected from Google maps. Other tests were performed over images collected by a Bebop 2 quadrotor flying over a neighborhood, besides videos, in a frame by frame basis. The result is that the classifier was able to correctly object detect, which means to identify the object searched for and to mark its localization through a bounding box. As a result, the perspective of using the proposed system to detect possible mosquitoes' breeding grounds is quite meaningful, justifying the development of the whole system, as described in this thesis.

Keywords: UAV, YOLO, CNN, object detection, Aerial surveillance, computer vision.

# Resumo

O número anual de casos de Dengue, Zika e Chikungunya no Brasil é alarmante. Uma forma de minimizar esse problema é identificar e tratar possíveis criadouros do mosquito, como piscinas, caixas d'água inexploradas, etc. Com foco em criar uma ferramenta para auxiliar as autoridades a localizar tais criadouros de mosquitos, esta Dissertação de Mestrado propõe um modelo YOLO específico para detectar piscinas em um determinado bairro, a partir de imagens capturadas por um quadrimotor. Assim, é proposto um sistema de detecção de objetos baseado em uma rede neural convolucional, a YOLOv3, para detectar piscinas em imagens coletadas pelo quadrimotor ao voar autonomamente sobre tal vizinhança. Foi criado um conjunto de dados, com imagens aéreas de alta resolução, com os objetos desejados anotados, para ser usado como um conjunto de dados de treinamento para as camadas internas da YOLO, com 150 imagens coletadas de sites de fotografias de alta resolução. A avaliação do classificador assim obtido ocorreu em um banco de dados contendo 72 imagens de satélite com diferentes resoluções, em três diferentes escalas de imagem, para duas localidades distintas, coletadas do Google maps. Outros testes foram realizados sobre imagens coletadas por um quadrimotor Bebop 2 sobrevoando um bairro, além de vídeos, quadro a quadro. O resultado é que o classificador conseguiu detectar corretamente o objeto, o que significa identificar o objeto procurado e marcar sua localização por meio de uma caixa delimitadora. Com isso, a perspectiva de utilizar o sistema proposto para detectar possíveis criadouros de mosquitos é bastante significativa, justificando o desenvolvimento de todo o sistema, conforme descrito nesta dissertação.

Palavras-chave: VANT, YOLO, CNN, detecção de objetos, vigilância aérea, visão computacional.



# Contents

	<b>Acknowledgements</b>	<b>6</b>
	<b>Abstract</b>	<b>8</b>
	<b>Resumo</b>	<b>9</b>
	<b>List of Figures</b>	<b>14</b>
	<b>List of Tables</b>	<b>15</b>
	<b>List of Abbreviations and Acronyms</b>	<b>16</b>
	<b>List of Symbols</b>	<b>17</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>18</b>
<b>1.1</b>	<b>Problem Definition</b>	<b>19</b>
<b>1.2</b>	<b>Motivation</b>	<b>19</b>
<b>1.3</b>	<b>Objectives</b>	<b>19</b>
<b>1.4</b>	<b>State-of-the-art</b>	<b>20</b>
<b>1.5</b>	<b>Thesis Structure</b>	<b>21</b>
<b>2</b>	<b>SYSTEM DESIGN</b>	<b>22</b>
<b>2.1</b>	<b>Hardware</b>	<b>22</b>
2.1.1	Bebop2	22
2.1.1.1	Bebop 2 Dynamic Modelling	23
2.1.1.2	Identification of Bebop 2 dynamic model parameters	25
2.1.2	Wi-Fi Router	26
2.1.3	Computer	27
2.1.4	PS4 Joystick	27
2.1.5	Landing pad	28
2.1.6	Tripod	28
2.1.7	Overview	28
<b>2.2</b>	<b>Bebop Camera Calibration</b>	<b>29</b>
<b>2.3</b>	<b>Software</b>	<b>30</b>
2.3.1	OS: Linux	30
2.3.2	ROS (Robot Operating System)	30
2.3.3	Bebop_lai	32
2.3.4	Bebop_autonomy	32

2.3.5	Darknet_ros . . . . .	32
2.3.6	Whycon . . . . .	33
2.3.7	BebopClass . . . . .	33
2.3.8	Exiftool . . . . .	34
2.3.9	ODM: OpenDroneMap . . . . .	35
<b>3</b>	<b>PATH PLANNING &amp; CONTROL . . . . .</b>	<b>36</b>
<b>3.1</b>	<b>UAV Input Data . . . . .</b>	<b>36</b>
<b>3.2</b>	<b>Path Planning . . . . .</b>	<b>38</b>
3.2.1	UAV Obstacles . . . . .	38
3.2.2	UAV Search Patterns . . . . .	39
<b>3.3</b>	<b>Conditional Tasking . . . . .</b>	<b>41</b>
<b>3.4</b>	<b>Position Controller . . . . .</b>	<b>43</b>
<b>3.5</b>	<b>Override Manual Controller: Joystick . . . . .</b>	<b>43</b>
<b>3.6</b>	<b>ROS Publiiser cmd_vel . . . . .</b>	<b>43</b>
<b>4</b>	<b>COMPUTER VISION . . . . .</b>	<b>44</b>
<b>4.1</b>	<b>Computer Vision . . . . .</b>	<b>44</b>
4.1.1	Computer Vision Tasks . . . . .	44
4.1.2	Object Detection . . . . .	44
<b>4.2</b>	<b>Yolo: Real Time Object Detection . . . . .</b>	<b>45</b>
<b>4.3</b>	<b>Dataset . . . . .</b>	<b>47</b>
4.3.1	Image data acquisition . . . . .	48
4.3.2	Image data augmentation . . . . .	49
4.3.3	Image annotation . . . . .	49
4.3.4	Training YOLOv3 Dataset on Google Colab . . . . .	50
<b>4.4</b>	<b>Testing of Dataset . . . . .</b>	<b>51</b>
4.4.1	High Resolution Images taken by UAV . . . . .	51
4.4.2	Low Resolution Images taken by Satellite . . . . .	51
<b>5</b>	<b>POST-PROCESSING . . . . .</b>	<b>53</b>
<b>5.1</b>	<b>Video Export . . . . .</b>	<b>53</b>
<b>5.2</b>	<b>Video Conversion . . . . .</b>	<b>53</b>
<b>5.3</b>	<b>Frame Extraction . . . . .</b>	<b>53</b>
<b>5.4</b>	<b>Metadata Inputation . . . . .</b>	<b>54</b>
<b>5.5</b>	<b>Image Homography . . . . .</b>	<b>54</b>
5.5.1	Calculating Bebob Homography Parameters . . . . .	54
<b>5.6</b>	<b>Overlap . . . . .</b>	<b>56</b>
<b>5.7</b>	<b>Orthophoto . . . . .</b>	<b>56</b>
<b>5.8</b>	<b>GSD . . . . .</b>	<b>57</b>

<b>6</b>	<b>EXPERIMENTS</b> . . . . .	<b>58</b>
<b>6.1</b>	<b>Experiment 1: Drone Flight with Yolo over a house pool</b> . . . . .	<b>58</b>
<b>6.2</b>	<b>Experiment 2: UAV Flight Yaw Orientation vs Fix Orientation</b> . . . . .	<b>59</b>
<b>6.3</b>	<b>Experiment 3: UAV Flight Different Overlap at same height</b> . . . . .	<b>60</b>
<b>6.4</b>	<b>Experiment 4: UAV Flight at 20 meters height over Parking lot</b> . . . . .	<b>61</b>
6.4.1	Experiment 4: ODM Image Results: 2D Orthophoto and 3D Point Cloud . . . . .	63
6.4.2	Experiment 4: Georeference Objects in Orthophoto . . . . .	64
<b>7</b>	<b>CONCLUSION</b> . . . . .	<b>65</b>
	<b>Conclusion</b> . . . . .	<b>65</b>
	<b>BIBLIOGRAPHY</b> . . . . .	<b>66</b>
	<b>APPENDIX</b> . . . . .	<b>71</b>
	<b>APPENDIX A – CASES FROM 2016 TO 2021 IN THE STATE OF ESPÍRITO SANTO</b> . . . . .	<b>72</b>
A.1	Accumulative Cases: Dengue, Chikungunya, Zika . . . . .	72
	<b>APPENDIX B – ROS ESSENTIALS</b> . . . . .	<b>73</b>
B.1	Install ROS Kinetic Kame . . . . .	73
B.2	ROS Cheatsheet . . . . .	74
B.3	ROS OOP (Object Oriented Programming) . . . . .	74
B.4	ROS Bebop Monocular Camera Calibration . . . . .	76
	<b>APPENDIX C – BEBOPCLASS</b> . . . . .	<b>78</b>
C.1	UML - BebopClass . . . . .	78
	<b>APPENDIX D – CONDITIONAL TASKING RECURRENT BLOCK FLOW CHART PER STATE</b> . . . . .	<b>79</b>

# List of Figures

Figure 1 – Aedes Aegypti Disease Confirmed Cases in Espírito Santo: 2016 to 2021	18
Figure 2 – Bebop 2	22
Figure 3 – Bebop 2 Mobility	23
Figure 4 – Wi-Fi Router TP-LINK TL-WR941HP	26
Figure 5 – PS4 Joystick	27
Figure 6 – Overview	29
Figure 7 – ROS Ecosystem	30
Figure 8 – ROS master node topic communication	31
Figure 9 – Whycon ROS implementation marking image center	33
Figure 10 – Design Pattern Facade: BebopClass	34
Figure 11 – Exiftool Metadata Extraction of Fig.30a	35
Figure 12 – OpenDroneMap image examples over UFES Rectory Square.	35
Figure 13 – FlowChart - Main Loop	36
Figure 14 – Browser: Google Maps UFES-CCE Parking Lot	37
Figure 15 – GUI: Bebop Input Data Steps 1 of 2	37
Figure 16 – GUI: Bebop Input Data Steps 2 of 2	37
Figure 17 – GUI: Bebop Input Data Filled	38
Figure 18 – UAV Obstacles: Physical and Interference	39
Figure 19 – Search Patterns: Creeping Line, Increasing Square, Sector	40
Figure 20 – UAV Search Pattern: Creeping Line and Increasing Square	40
Figure 21 – UAV Search Pattern: Increasing Square Evolution	41
Figure 22 – Conditional Tasking Overall Layout	41
Figure 23 – Flowchart Overview	42
Figure 24 – Flowchart State	42
Figure 25 – Computer Vision Tasks. Image from ClimateNet Website	44
Figure 26 – Yolo version 1 Network Architecture (REDMON et al., 2016)	46
Figure 27 – YOLOv4 Performance Graph. AP(Average Precision) vs FPS	46
Figure 28 – Downloaded photos from Unsplash website	48
Figure 29 – Downloaded from Unsplash.com	49
Figure 30 – Validation in High Resolution	51
Figure 31 – Validation in Low Resolution Images from Ilha Do Boi and Las Vegas.	52
Figure 32 – Bebop 2 USB Port for Video Extraction	53
Figure 33 – Applying Homography on the original image for a bird’s-eye view	54
Figure 34 – Rectangular Pyramid: Oblique and Right	55
Figure 35 – Bebop 2 Camera FoV: Oblique Rectagular Pyramide	55
Figure 36 – Image Overlap	56

Figure 37 – Orthographic view vs Perspective view . . . . .	57
Figure 38 – Perspective Projection . . . . .	57
Figure 39 – UAV Dataset False Positives in Person Point-of-view . . . . .	58
Figure 40 – UAV Dataset True and False Positives in bird’s-eye view . . . . .	58
Figure 41 – UAV Flight Yaw Orientation Issue . . . . .	59
Figure 42 – UAV Flight Different Overlap at same height . . . . .	60
Figure 43 – UAV Flight at 20 meters height over Parking lot . . . . .	61
Figure 44 – 3D Overview of Flight at 20 meters . . . . .	61
Figure 45 – Search Pattern Photos Composition . . . . .	62
Figure 46 – ODM Image Results: 2D Orthophoto and 3D Point Cloud . . . . .	63
Figure 47 – ODM Orthophoto Georeference . . . . .	64
Figure 48 – Histograms: Confirmed Cases per Week from 2016 to 2021 . . . . .	72
Figure 49 – ROS Camera Calibration . . . . .	76
Figure 50 – UML: BebopClass . . . . .	78
Figure 51 – Recurrent(S0) Block . . . . .	79
Figure 52 – Recurrent(S1) Block . . . . .	79
Figure 53 – Recurrent(S2) Block . . . . .	79
Figure 54 – Recurrent(S3) Block . . . . .	79
Figure 55 – Recurrent(S4) Block . . . . .	80
Figure 56 – Recurrent(S5) Block . . . . .	80
Figure 57 – Recurrent(S6) Block . . . . .	80
Figure 58 – Recurrent(S7) Block . . . . .	80
Figure 59 – Recurrent(S8) Block . . . . .	81
Figure 60 – Recurrent(S9) Block . . . . .	81

# List of Tables

Table 1 – Parameters Values from the Identified Dynamic Model . . . . .	25
Table 2 – Computer Specifications . . . . .	27
Table 3 – ROS Python vs Cpp Comparison . . . . .	32
Table 4 – SAR Search Patterns Comparison . . . . .	40
Table 5 – Conditional Tasking States . . . . .	42
Table 6 – Comparison of YOLO Versions Performance for the <i>dog.jpg</i> sample photo	47
Table 7 – Comparison between some common annotation formats . . . . .	50
Table 8 – ODM Quality Report: Dataset and Processing Summary . . . . .	63
Table 9 – Georeferencing Objects on image: . . . . .	64

# List of abbreviations and acronyms

<b>UAV</b>	Unmanned Aerial Vehicle
<b>MAV</b>	Micro Aerial Vehicle
<b>VANT</b>	<i>Veículo Aéreo Não Tripulado</i> (UAV in Portuguese)
<b>GPS</b>	Global Position System
<b>GSD</b>	Ground Sample Distance
<b>SAR</b>	Search And Rescue
<b>2D</b>	Two Dimensional
<b>3D</b>	Three Dimensional
<b>FoV</b>	Field of View
<b>RGB</b>	Red, Green and Blue
<b>RGBD</b>	Red, Green, Blue and Depth
<b>FPGA</b>	Field-Programmable Gate Array
<b>CPU</b>	Central Processing Unit
<b>GPU</b>	Graphics Processing Unit
<b>CUDA</b>	Compute Unified Device Architecture
<b>IoU</b>	Intersection over Union
<b>RAM</b>	Random Access Memory
<b>USB</b>	Universal Serial Bus
<b>WiFi</b>	Wireless Fidelity
<b>LTS</b>	Long-term support
<b>OOP</b>	Object Oriented Programming
<b>SDK</b>	Software Development Kit
<b>UML</b>	Unified Modeling Language
<b>ROS</b>	Robot Operating System
<b>ROS2</b>	Robot Operating System 2
<b>EOL</b>	End-of-life
<b>IROS</b>	International Conference on Intelligent Robots and Systems
<b>ICRA</b>	International Conference on Robotics and Automation
<b>ICUAS</b>	International Conference on Unmanned Aircraft Systems
<b>ANAC</b>	National Civil Aviation Agency of Brazil
<b>UFES</b>	Federal University of Espírito Santo
<b>LAI</b>	Intelligent Automation Laboratory at UFES
<b>IFES</b>	Federal Institute of Espírito Santo
<b>ETH</b>	Swiss Federal Institute of Technology in Zurich

# List of symbols

$x$	Robot Position, translation in relation to Worlds x-axis
$y$	Robot Position, translation in relation to Worlds y-axis
$z$	Robot Position, translation in relation to Worlds z-axis
$\phi$	Roll - Robot Orientation, its x-axis rotation in relation to World x-axis
$\theta$	Pitch - Robot Orientation, its y-axis rotation in relation to World y-axis
$\psi$	Yaw - Robot Orientation, its z-axis rotation in relation to World z-axis
$\dot{x}$	Robot Linear Speed in relation to its x-axis
$\dot{y}$	Robot Linear Speed in relation to its y-axis
$\dot{z}$	Robot Linear Speed in relation to its z-axis
$\dot{\phi}$	Robot Angular Speed in relation to its x-axis
$\dot{\theta}$	Robot Angular Speed in relation to its y-axis
$\dot{\psi}$	Robot Angular Speed in relation to its z-axis
$\ddot{x}$	Robot Linear acceleration in relation to its x-axis
$\ddot{y}$	Robot Linear acceleration in relation to its y-axis
$\ddot{z}$	Robot Linear acceleration in relation to its z-axis
$\ddot{\phi}$	Robot Angular acceleration in relation to its x-axis
$\ddot{\theta}$	Robot Angular acceleration in relation to its y-axis
$\ddot{\psi}$	Robot Angular acceleration in relation to its z-axis
$\mathbf{X}$	Robot position state vector
$\dot{\mathbf{X}}$	Robot speed states vector
$\ddot{\mathbf{X}}$	Robot acceleration states vector
$\mathbf{U}$	Control Signal Vector
$f_1, f_2$	Dynamic parameter matrices of the quad engine model



# 1 Introduction

At the beginning of 2020, during the Covid-19 pandemic, there was a severe increase in the number of cases of Dengue, Zika, and especially Chikungunya (Secretaria de Saúde do Estado do Espírito Santo, 2021c) Figure 1 in the city of Vitória, State of Espírito Santo (ES), Brazil, and especially in Ilha do Boi, a neighborhood settled in an island area with huge houses, several of them having lavish gardens and swimming pools requiring constant maintenance. From January to April 2020, 59.7% of the 9293 Chikungunya cases in the state were in Vitória, with a heavy impact in Ilha do Boi neighborhood. During the same period in 2021, the number of Chikungunya cases in the state came down to 1411 (Secretaria de Saúde do Estado do Espírito Santo, 2021a), which still is unacceptable. For more confirmed cases information see Appendix A.

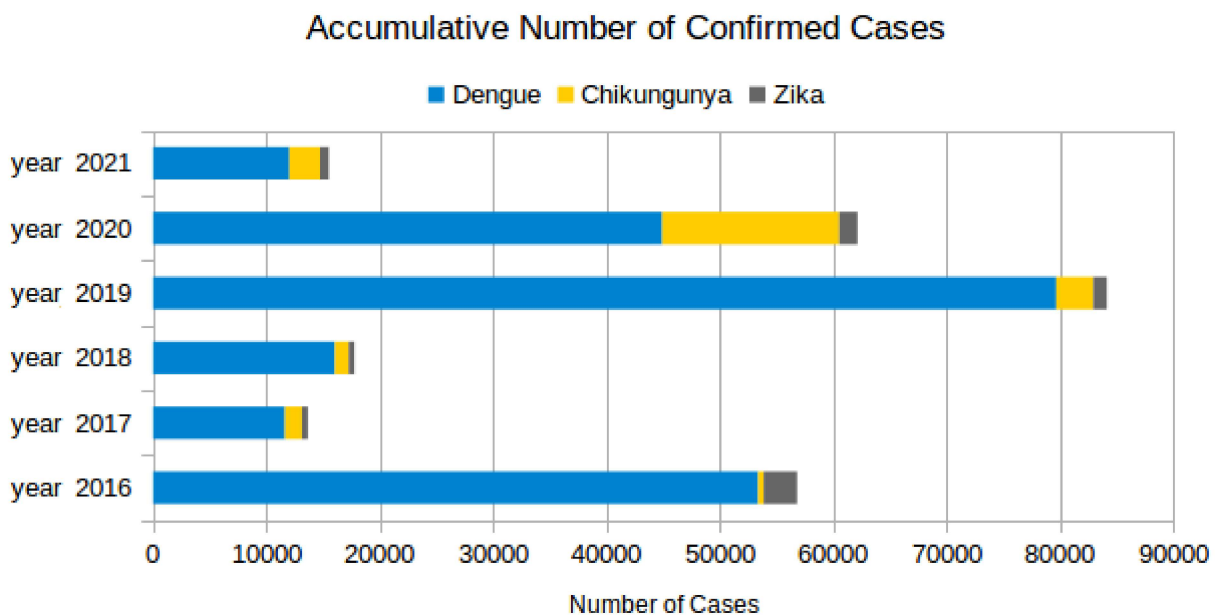


Figure 1 – Aedes Aegypti Disease Confirmed Cases in Espírito Santo: 2016 to 2021

Taking into account the demand for actions directed to reduce the incidence of the Aedes Aegypti mosquito, this thesis proposes a tool to help the authorities to search for possible breeding grounds, through the use of aerial images collected using an unmanned aerial vehicle (UAV), in particular a quadrotor, popularly known as drone. The proposed system focuses on collecting images and identifying in such images, through computer vision techniques, objects with potential to store mosquitoes larvae. In particular, this work focuses on identifying swimming pools, which are quite frequent in Ilha do Boi neighborhood, Vitória, and are much likely, when not correctly treated, to be breeding ground for the mosquitoes.

## 1.1 Problem Definition

The state of Espírito Santo is a natural region of the mosquito *Aedes Aegypti*<sup>1</sup> (MALTA et al., 2005), and annually many features contribute to create the necessary and suitable conditions for its procreation (Secretaria de Saúde do Estado do Espírito Santo, 2021b). Typical high temperatures at the end of summer and good local afforestation, for instance, create a natural habitat for such a mosquito. In 2020, at the beginning of the pandemic, almost no residents were wandering around the city. Besides, with the high precipitation then verified (VALENÇA et al., 2013) puddles and stagnant water came on, which caused, as a consequence, proliferation of mosquito larvae.

## 1.2 Motivation

With a significant increase in the number of Chikungunya confirmed cases during the shortage in health resources caused by the Covid-19 pandemic, this work proposes a tool to minimize the mosquito problem and, as a consequence, its diseases.

As our laboratory<sup>2</sup> deals with mobile robotics and computer vision, we decided to develop a system to search for nest environments of such a mosquito using a micro aerial vehicle Parrot Bebop 2 (PINTO et al., 2020), controlled via ROS (Robot Operating System) as UAV (Unmanned Aerial Vehicle), working with the video streaming of its camera for Object Detection, looking for things that could hold water, thus being, therefore, favorable to the proliferation of such mosquito. Places searched on previous studies (MALTA et al., 2005) (VALENÇA et al., 2013), such as swimming pools, untapped water tanks, lost tires, vases without plant, some varieties of plant bromeliad (COGLIATTI-CARVALHO et al., 2010), down to a tiny bottle lid on the ground, are among the items described by the State Health Department (Secretaria de Saúde do Estado do Espírito Santo, 2021c). The success of the application is directly related to the capability of the drone camera and the use of internet to search and download high-definition aerial images to create a custom Dataset. To simplify, we started by searching big objects, such as a swimming pool, leaving smaller object classes for future work.

## 1.3 Objectives

This work has the objective of implementing a UAV Aerial surveillance system to capture images for georeferencing a water reservoir and visually detect potential *Aedes*

<sup>1</sup> <[https://en.wikipedia.org/wiki/Aedes\\_aegypti](https://en.wikipedia.org/wiki/Aedes_aegypti)>

<sup>2</sup> LAI - Laboratory of Automation and Intelligence from the Portuguese "Laboratório Automação Inteligente", Universidade Federal do Espírito Santo - UFES

Aegypti breeding ground, as a tool intended to help decrease the number of cases of diseases caused by this mosquito.

As part of this general objective, the following specific objectives are stressed:

- To write a code to control the Drone autonomously, using Open Source tools, such as Python and ROS.
- To create a custom dataset of water reservoirs (to detect potential mosquito breeding grounds) in urban areas from high-resolution bird's-eye view images downloaded from the internet.
- To run experiments to test the custom Dataset.
- To run experiments correspondent to outdoor UAV flights to test the swimming pool detection system.

## 1.4 State-of-the-art

Quadrotors have become a quite typical rotary-wing vehicle, and, therefore, they are present in several daily-life tasks. In many applications, quadrotors, the popular drones or UAVs, have been used as remotely operated vehicles, although being working autonomously in a crescent number of cases, such as in this thesis.

They have been used in a bunch of civil and even military applications ([CHEN; LAEFER; MANGINA, 2016](#)), such as in fire prevention and extinguishing ([MERINO; DIOS; OLLERO, 2015](#); [AMORIM; VASSALLO; SARCINELLI-FILHO, 2019](#); [HARIKUMAR; SENTHILNATH; SUNDARAM, 2019](#); [LENZ, April 28, 2021](#)), plantation inspection and other activities in agriculture ([ZHANG; KOVACS, 2012](#); [RADOGLU-GRAMMATIKIS et al., 2020](#)), load transportation ([VILLA; BRANDÃO; SARCINELLI-FILHO, 2020](#)), and building inspection ([TAN et al., 2021](#)), just to mention a few of them. Much of the versatility of using such vehicles as the main hardware in inspection is their capability to collect bird-eye-view images of the areas being inspected, allowing an overview of large areas through a single image. Therefore, many of these applications explore the combined use of image processing facilities and autonomous navigation based on high-level control of the UAV motion.

As mentioned above, this is the case in this thesis, which combines autonomous navigation of a quadrotor Bebop 2 from Parrot Drones SAS with machine learning techniques to analyze images collected by the UAV. The idea is to collect high-quality images of a residential area, looking for swimming pools in such images, to record the georeference correspondent to the detected swimming pools, using a YOLO CNN ([REDMON et al., 2016](#)) as the image processing structure, in particular, its 3rd version ([REDMON; FARHADI, 2018](#)).

Several other proposals are using the YOLO structure in its third version to process aerial images, such as in (SUN; PIAO; CUI, 2020), where the focus is in optimizing the YOLOv3 structure, in (JIAO et al., 2020), whose focus is the feature learning for fire detection, and in (QU et al., 2018), whose focus is to detect pedestrians in urban images. Similarly, this thesis focuses on detecting objects in aerial images collected by an autonomous drone flying over a previously established region, particularly swimming pools, considering different image scales (the drone can fly in different altitudes) and resolutions (satellite images got from the internet are used in the training step).

Throughout this thesis research development, the partial results obtained reported in articles published in international conferences: (PINTO et al., 2020) and (PINTO; CIARELLI; SARCINELLI-FILHO, 2021).

- A. O. PINTO, H. N. MARCIANO, V. P. BACHETI, M. S. M. MOREIRA, A. S. BRANDÃO AND M. SARCINELLI-FILHO, "High-Level Modeling and Control of the Bebop 2 Micro Aerial Vehicle," *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 939-947, doi: 10.1109/ICUAS48674.2020.9213941.
- A. O. PINTO, P. M. CIARELLI AND M. SARCINELLI-FILHO, "Creating a Yolov3 Custom Model for Aerial Surveillance," *2021 XIX Workshop on Information Processing and Control (RPIC)*, 2021, pp. 1-6, doi: 10.1109/RPIC53795.2021.9648411.

## 1.5 Thesis Structure

The organization of this Master Thesis follows the structure below:

**Chapter 1: Introduction** This chapter shows the damage of the *Aedes-Aegypti* mosquito; mentions UAV autonomous navigation applications with image processing, and proposes an autonomous UAV aerial surveillance YOLO.

**Chapter 2: System Design** This chapter details the parts used that make up the system in hardware and software and the characteristics to help understand and replicate the case study of this master's thesis.

**Chapter 3: Path Planning & Control** This chapter presents the application workflow by dividing it into three sections Input Data, Path Planning, and Control.

**Chapter 4: Computer Vision** This chapter shows how to custom train YOLO, for object detection swimming pool, one probable mosquito breeding ground.

**Chapter 5: Post-processing** This chapter displays the post-processing techniques to overcome the Bebop 2 Drone imagery limitations.

**Chapter 6: Experiments** This chapter presents the experiments and their results.

**Chapter 7: Conclusion and Future Works** This chapter lists the main conclusions of the work carried out and proposals for continuing the research.

## 2 System Design

This chapter describes the development of this research from a physical and logical point of view. After reading it, the reader should understand the test environment and replicate it if necessary. It will be explained technical data of the hardware used, software architecture, the developed interfaces, and the physical environment used for testing.

### 2.1 Hardware

In this section, the hardware aspects used will be discussed, such as specifications, equipment models, and essential characteristics considered for the correct execution of the experiment.

#### 2.1.1 Bebop2

The Bebop 2 is a rotary wing "quadrimotor" drone produced by the company Parrot Drones SAS, Figure 2. As a filming drone, it is equipped with a 14 mega-pixel fisheye front camera, with FullHD 1080p resolution and optical stabilization. It also has dualcore CPU and quadcore GPU embedded processors. It comes with built-in instruments such as sensors: barometer and ultrasound (help to measure altitude), GPS, accelerometers, battery charge level and a face-down camera for optical flow. Most of these sensors can be accessed through software provided by the manufacturer or through a ROS package [2.3.4](#) (used in this work), based on the official Parrot SDK (Software Development Kit).



Figure 2 – Bebop 2

### 2.1.1.1 Bebop 2 Dynamic Modelling

From the operational point of view, the UAV has 6 Degrees of Freedom (6 DoF). It can activate linear velocity commands in all three axes  $x^b$ ,  $y^b$  and  $z^b$  (where the superscript  $b$  indicates Bebop's frame), as well as angular velocity commands around the same axes. These angular velocity commands correspond to the Roll, Pitch and Yaw movements, represented by the angles  $\phi$ ,  $\theta$  and  $\psi$ , respectively. Combinations of such movements are performed by controlling the forces generated by each one of the rotors, as indicated in Figure 3.

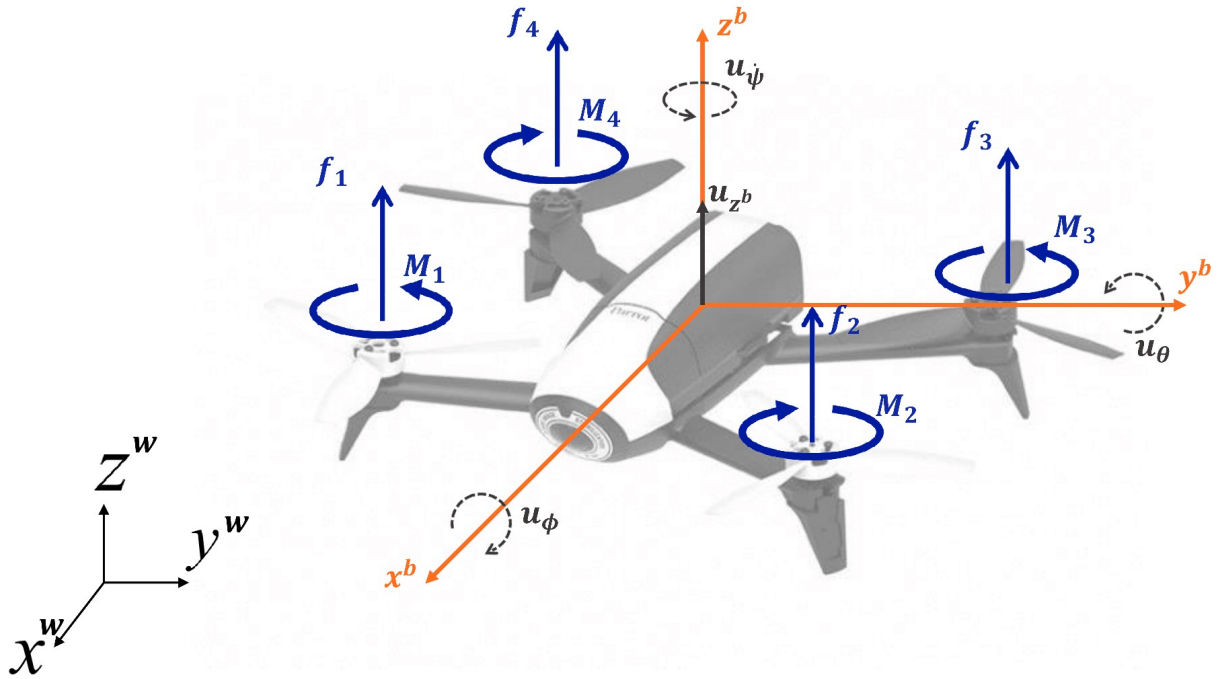


Figure 3 – Bebop 2 Mobility

Roll and Pitch movements are not the focus in this work since it uses a bebop model where these movements were limited to  $\pm 5^\circ$ , to be possible to model the UAV as a fully actuated system (see (PINTO et al., 2020)), whose velocity commands are

$$\mathbf{U} = [u_{v_x} \quad u_{v_y} \quad u_z \quad u_{\dot{\psi}}]^\top, \quad (2.1)$$

without considerable loss in the precision of the model. Since this application happens in an outdoor environment and to overcome the wind, one of the uncontrollable forces of nature, more aggressive Roll and Pitch movements were applied and limited to  $\pm 10^\circ$ . As a consequence, they were introducing a small error to the model  $x^b$  axis and  $y^b$  axis. As observed in Experiment on Chapter 6.2 stability for the linear axes; however, Yaw movement was influenced by such model change, therefore to bypass this issue, a fix North Orientation to Yaw movement was established.

The kinematic model of the Bebop 2 also considers it as a massless virtual point which can respond to the velocity commands instantaneously.

Taking into account only the four DoF specified in (2.1), and considering that the Pitch and Roll movements are small enough so that they may be ignored when the UAV moves alongside the  $x^b$  and  $y^b$  axes (see (PINTO et al., 2020)), its matrix of direct kinematics is

$$\mathbf{A}_b = \begin{bmatrix} \cos \psi & -\sin \psi & 0 & 0 \\ \sin \psi & \cos \psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (2.2)$$

and its matrix of inverse kinematics is

$$\mathbf{A}_b^{-1} = \begin{bmatrix} \cos \psi & \sin \psi & 0 & 0 \\ -\sin \psi & \cos \psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (2.3)$$

so that the transformation of the velocities from the Bebop Frame (BF) to the WF is written as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\psi} \end{bmatrix} = \mathbf{A}_b \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_\psi \end{bmatrix}, \quad (2.4)$$

and the transformation from the WF to the PF is written as

$$\begin{bmatrix} u_x \\ u_y \\ u_z \\ u_\psi \end{bmatrix} = \mathbf{A}_b^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\psi} \end{bmatrix}, \quad (2.5)$$

where  $[\dot{x} \ \dot{y} \ \dot{z} \ \dot{\psi}]^\top$  are the linear velocities and the yaw velocity, in the WF.

Now, taking into account the dynamics of the UAV, a new model is considered, described as (SANTANA; BRANDÃO; SARCINELLI-FILHO, 2016)

$$\ddot{\mathbf{X}} = \mathbf{f}_1 \mathbf{u}_d - \mathbf{f}_2 \dot{\mathbf{X}}, \quad (2.6)$$

where,

$$\mathbf{f}_1 = \begin{bmatrix} \cos \psi & -\sin \psi & 0 & 0 \\ \sin \psi & \cos \psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} K_1 & 0 & 0 & 0 \\ 0 & K_3 & 0 & 0 \\ 0 & 0 & K_5 & 0 \\ 0 & 0 & 0 & K_7 \end{bmatrix} = \mathbf{A}_b \mathbf{K}_1, \quad (2.7)$$

$$\mathbf{f}_2 = \begin{bmatrix} \cos \psi & -\sin \psi & 0 & 0 \\ \sin \psi & \cos \psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} K_2 & 0 & 0 & 0 \\ 0 & K_4 & 0 & 0 \\ 0 & 0 & K_6 & 0 \\ 0 & 0 & 0 & K_8 \end{bmatrix} = \mathbf{A}_t \mathbf{K}_2, \quad (2.8)$$

For the design of the controller to be used for the quadrotor, the dynamic model proposed in (SANTANA et al., 2014; SANTANA; BRANDÃO; SARCINELLI-FILHO, 2016; SANTANA, 2016) and already used in (SANTOS, 2017), (SANTOS et al., 2017), (SANTOS et al., 2017) and (SANTOS et al., 2019) given by

$$\ddot{\mathbf{X}} = \mathbf{f}_1 \mathbf{U} - \mathbf{f}_2 \dot{\mathbf{X}}, \quad (2.9)$$

from which the controller is designed based on the vehicle inverse dynamics, given by

$$\mathbf{U} = \mathbf{f}_1^{-1}(\nu + \mathbf{f}_2 \dot{\mathbf{X}}) \quad (2.10)$$

where

- $\mathbf{U}$  is a vector that contains the control signal sent to the drone;
- $\mathbf{f}_1^{-1}$  represents a matrix of positive constants characteristic of each drone;
- $\nu$  is given by  $\nu = \ddot{\mathbf{X}}_d + \mathbf{K}_p \tilde{\mathbf{X}} + \mathbf{K}_d \dot{\tilde{\mathbf{X}}}$ ;
- $\mathbf{K}_p$  and  $\mathbf{K}_d$  are defined positive diagonal matrices (gains);
- $\tilde{\mathbf{X}} = \mathbf{X}_d - \mathbf{X} = [x_d - x \quad y_d - y \quad z_d - z \quad \psi_d - \psi]^T$ ; and
- $\mathbf{f}_2$  also represents a matrix of positive constants characteristic of each drone.

### 2.1.1.2 Identification of Bebop 2 dynamic model parameters

Table 1 – Parameters Values from the Identified Dynamic Model

$K_1$	$K_2$	$K_3$	$K_4$	$K_5$	$K_6$	$K_7$	$K_8$
0.8417	0.1823	0.8354	0.1710	3.966	4.001	9.8524	4.7295

$\dot{\mathbf{X}} = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{\psi}]^T$ ,  $\ddot{\mathbf{X}} = [\ddot{x} \quad \ddot{y} \quad \ddot{z} \quad \ddot{\psi}]^T$  and  $\mathbf{u}_d$  is a vector with the control signals sent to the UAV. As for the parameters  $K_1, \dots, K_8$ , they are estimated using experimental values gathered according to the procedure described in (PINTO et al., 2020). The values thus obtained for such parameters, which are the values used in this thesis, are presented in Table 1.



## 2.1.2 Wi-Fi Router

The communication between UAV and computer is Wi-Fi connection. The Parrot Bebop 2 Drone is 2.4 and 5.0 GHz compatible. The range is higher in the 2.4 GHz band because lower frequencies can penetrate solid objects, such as walls and floors. However, higher frequencies allow data to be transmitted faster than lower frequencies, so the 5 GHz band allows faster file uploading and downloading. Since the speed is within the band range speed and the experimentation is outdoor, a 2.4 GHz Wi-Fi router was the better option. In other words, reach longer distance flights and penetrating solid objects or possible obstacles, such as treetops, parts of the external housing wall, chimneys, or rooftop edges. For these reasons, the Wifi router TL-WR941HP Figure 4 was selected.

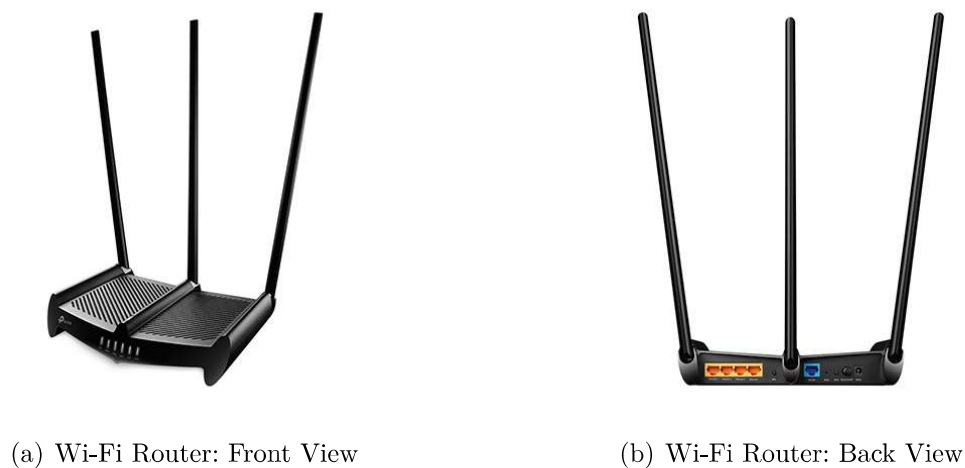


Figure 4 – Wi-Fi Router TP-LINK TL-WR941HP

Brand: **TP-LINK**

Model: **TL-WR941HP**<sup>1</sup>

Description: **2.4 GHz 450Mbps (802.11n) High Power Wireless N Router**

Specs overview:

- Wireless Range up to 900m<sup>2</sup> – High power amplifiers and high-gain antennas.
- Wall-Penetrating Wi-Fi – cuts through walls and obstacles, eliminating dead zones.
- Three wireless Functionality - Router, Range Extender, and Access Point.
- 450Mbps Wireless Speed – Ideal for video streaming, online gaming and VoIP.

<sup>1</sup> <<https://www.tp-link.com/en/home-networking/wifi-router/tl-wr941hp/v2/#specifications>>

### 2.1.3 Computer

The experiment run on a computer described on Table 2 from UFES LAI Laboratory. Computer model Dell XPS 8700, a Tower desktops form factor computer with a GPU upgrade to GeForce GTX 1650<sup>2</sup>.

Model	Dell XPS 8700
OS	Ubuntu 16.04 LTS 64-bit
CPU	Intel® Core™ i7-4790 CPU @ 3.60GHz × 8 cores
GPU	GeForce GTX 1650
RAM	16 GB DIMM DDR3-1600
SSD	500 GB
PSU	460 Watt
Others	Monitor, Keyboard and Mouse

Table 2 – Computer Specifications

### 2.1.4 PS4 Joystick

For safety reasons the PS4 Joystick DualShock 4 controller is set and overrides the position controller ROS speed message `cmd_vel` before publishing to the Bebop 2. It features two sticks, a touchpad, a 3.5mm headphone jack, four triggers, a directional pad, plus triangle, cross, circle, and square buttons, the PS buttons, and the share button and options button. This last two work as land and takeoff command when the code is running. For flight commands controller configuration see Figure 5.



Figure 5 – PS4 Joystick

<sup>2</sup> <<https://www.nvidia.com/pt-br/geforce/graphics-cards/gtx-1650/>>

### 2.1.5 Landing pad

During the development of this thesis, some types of visual markers as Landing pads got used. While some Figure 6c painted with white on 50x50cm black rubber tiles, geometric forms to create a high contrast image for the bebop2 low-resolution camera to better detect optical flow and better position itself in space during takeoff and landing. Other markers had more visual information to pass as the marker's precise position to the center of the camera lenses, like the Whycon used to calculate the homography Figure 35 and visual calibration.

### 2.1.6 Tripod

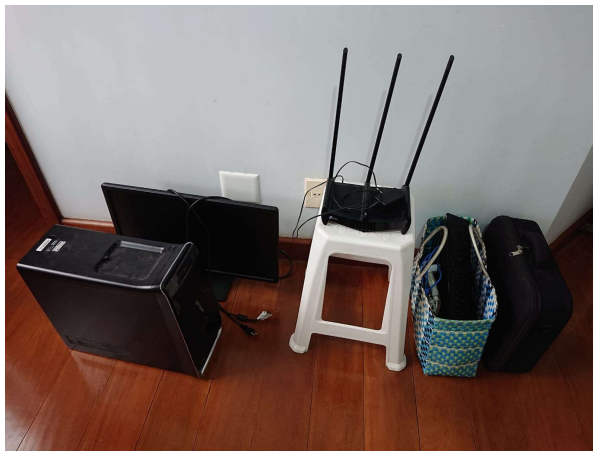
To measure the FoV (Field of View) of the Bebop 2 Camera and its video aspect ratio after homography from 83 degrees, which is the maximum pitch angle of this drone to 90 degrees angle bird's-eye view in Chapter 5.5 Figure 35. A firm and stable base created with a flat plastic cover 35x25cm screwed onto a Professional Tripod Model FT-6905 *FanCies*. With its handles and built-in bubble level, it is possible to level the drone on top of the base at one-meter height from the camera line of sight to the ground level. This allowed to implement camera functionalities while drone is on the ground facing down at different visual pattern:

Technical Data: Anodized aluminum, height of 73cm to 176cm, load 6kg and weight 6.5kg.

### 2.1.7 Overview

As shown in Figure 6, a system composed of a computer running Linux operating systems and many peripherals connected outside on an open environment. Furthermore, the task force for a single flight experiment needed some effort by packing, transporting, setting up at the flight site. A detailed list of equipment is below:

- **Drone case:** Drone, 3 batteries, extra propeller, and necessary tools.
- **Computer Linux ROS:** Desktop DELL 7th generation INTEL Core I7 processor, 16GB DDR4 RAM, 1TB hard drive and Graphics Card GeForce GTX 1650;
- **Peripherals:** Monitor, keyboard and mouse;
- **Joystick:** PS4 model for drone safety.
- **Wifi Router:** Model Tplink Gigabit 2.4GHz wifi transmission;
- **Furniture:** Plastic Table and plastic chairs.
- **Landing pad:** Rubber matres with visual markers painted.
- **Cabling:** Energy cable 25 meters and Ethernet Cable 25 meters.
- **Notebook:** Pen and paper for annotation.



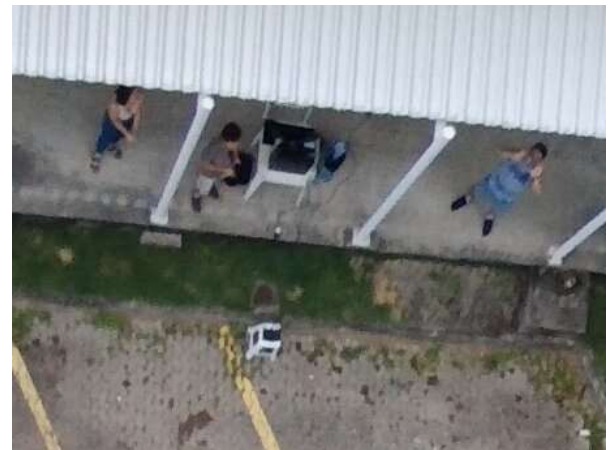
(a) Equipment packing



(b) Equipment transportation



(c) Setting up Equipment



(d) Flying Outdoor Experiments

Figure 6 – Overview

## 2.2 Bebop Camera Calibration

When a camera act as a visual sensor as part of one of these fields like robotics, surveillance, space exploration, or industrial automation, there is a need to rely upon imagery for measurement purposes and determine an accurate relationship between the actual world 3D points and its corresponding 2D pixel projected in the image. Knowing the camera's parameters in this application is essential to use it effectively as a visual sensor. Therefore, camera calibration is the process of estimating these camera parameters (intrinsic and extrinsic) by captured images of known ground truth, such as a chessboard<sup>3</sup>, in different poses, angles, distances, and locations on the image. Appendix B.4

**Intrinsic parameters** refer to the camera/lens system include focal length, optical center, and lens radial distortion coefficients; And, **Extrinsic parameters** refer to the camera's orientation (rotation and translation) concerning a world coordinate system.

<sup>3</sup> <[http://wiki.ros.org/camera\\_calibration/Tutorials/MonocularCalibration?action=AttachFile&do=view&target=check-108.pdf](http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration?action=AttachFile&do=view&target=check-108.pdf)>

## 2.3 Software

This section shows an approach to the software architecture used and the software developed to carry out the experiments, and the interfaces developed to exchange information between the equipment.

### 2.3.1 OS: Linux

Due to pre-requirements of the ROS bebop\_autonomy package, the OS Linux<sup>4</sup> Distribution is 16.04, and the ROS version suitable for this is ROS Kinetic Kame<sup>5</sup>.

Ubuntu is a Linux OS (Operating System) distribution based on Debian and composed mostly of free and open-source software. Ubuntu is officially released in three editions: Desktop, Server, and Core for IoT (Internet of things) devices and robots. All the editions can run on the computer alone or in a virtual machine. Ubuntu is released every six months, with LTS (long-term support) released every two years.

### 2.3.2 ROS (Robot Operating System)

This topic, however, will stick to the concepts necessary for the reader to understand their use in the context of this dissertation. ROS Essentials: Installation Appendix-B.1 and Cheatsheet Appendix-B.2.

ROS is not a specific operating system for robots, as the name says. ROS is an open-source and flexible framework that connects to a low-level system through a wide variety of high-level inputs for the most diverse systems and hardware used in robots. The ROS ecosystem Figure 7 comprises plumbing, tools, capabilities, and community that facilitate the development of complex behaviors of robots acting individually or together. At its core, ROS provides a message-passing system, often called "middleware" or "plumbing". The tools include launch, introspection, debugging, visualization, plotting, logging, and playback. Capabilities from drivers to algorithms to user interfaces, ROS provides the building blocks that focus on the application. The ROS community is large, diverse, and global.



Figure 7 – ROS Ecosystem

<sup>4</sup> <<https://ubuntu.com/16-04>>

<sup>5</sup> <<http://wiki.ros.org/kinetic>>

In simple terms, ROS operates with a central service called Master node, which coordinates communication between all nodes in the network. This node has the information of the agents that make up the network, together with their respective messages, which are published and received.

When connecting to the network, the new node informs its address, topics and types of messages to the Master node that has a fixed and known address<sup>6</sup>. It checks if any other node has the same topics with the same types of messages, if there is, the master node responds to the requesting node the address of the other nodes and which topics are the same. Thus, requesting node one sends a message to the second node, which connects to the first node when sending a response, thus building an information network where messages are published and subscribed all the time. Nodes can be Publisher and/or Subscriber types. Figure 8 illustrates this communication.

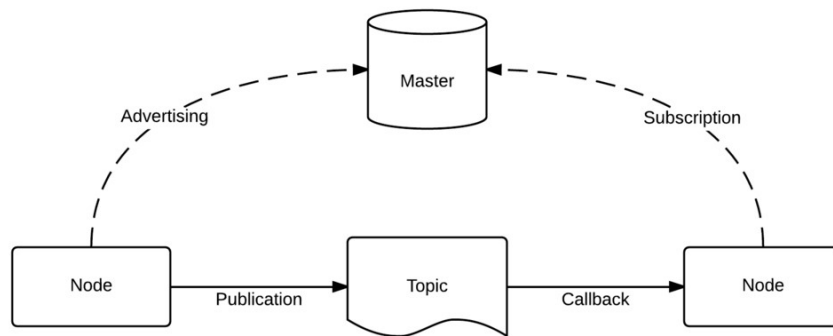


Figure 8 – ROS master node topic communication

**BebopClass** was written using ROS Python with OOP and using a patterns design called Facade. See section 2.3.7 of this chapter.

Python vs Cpp with ROS Depending on the application, performance, and programmer skills, one language might be better than the other see Table 3. The code is in Python in this project because of its versatile nature for prototyping and facility for future implementation. On the other hand, some nodes are in Cpp due to performance, especially image handling nodes that carry a vast amount of information at a high-frequency rate.

ROS officially supports Python (rospy) and Cpp (roscpp), and has modules implemented in that language for interacting with ROS, respectively rospy and roscpp.

As good programming practices, codes written in Python and Cpp using OOP (Object Oriented Programming) are presented Appendix-B.3, showing one of the biggest advantages of this implementation, which is the future change in the code.

<sup>6</sup> `ROS_MASTER_URI= http://hostname:11311/`

Table 3 – ROS Python vs Cpp Comparison

Python	Cpp
Academic or learning ROS	Use ROS Professionally
No need for performance	Need for Performance
Short Development Time	Long Development Time
Not familiar with Cpp	Familiar with Cpp
Easy Code Maintenance	Hard Code Maintenance
There is only a Python library	There is only a Cpp library
Interpreted Code	Compiled Code

### 2.3.3 Bebop\_lai

The `Bebop_lai` is a collection of ROS packages `Bebop_autonomy`, `Darknet_ros`, `Whycon` bounded together on a single package and accessed through the `BebopClass`. It was developed at LAI Intelligent Automation Laboratory at UFES by this thesis's author.

### 2.3.4 Bebop\_autonomy

The `bebop_autonomy`<sup>7</sup> package acts as a connection driver between ROS and official SDK made available by Parrot. It was developed at the Autonomy Lab at Simon Fraser University by Mani Monajjemi and a few collaborators, and maintained by Sepehr MohaimenianPour (also from the Autonomy Lab, Simon Fraser University), Thomas Bamford (from the Dynamic Systems Lab, University of Toronto) and Tobias Naegeli (Advanced Interactive Technologies Lab, ETH Zurich).

Developed in the C++ programming language, the package offers a series of topics and resources that enable the `Bebop 2` user to access internal data and send commands to the quad engine. To explain in a superficial way, the communication driver publishes threads that send messages with information on odometry and status of the four engine, and receive messages like speed commands.

The communication driver downloaded in February 2020 and used in this research was run on a Linux operating system distribution Ubuntu 16.04, along with the ROS Kinetic version cited in Subsection 2.3.2.

### 2.3.5 Darknet\_ros

The `Darknet_ros`<sup>8</sup> is a ROS package (BJELONIC, 2016–2018) developed for object detection in-camera images. It is an implementation ROS Package of YOLO ("You only look once") a state-of-the-art, real-time object detection system. In the following ROS package, use YoloV3 (YOLO version 3) (REDMON; FARHADI, 2018) on GPU and CPU.

<sup>7</sup> <[https://github.com/AutonomyLab/bebop\\_autonomy](https://github.com/AutonomyLab/bebop_autonomy)>

<sup>8</sup> <[https://github.com/leggedrobotics/darknet\\_ros](https://github.com/leggedrobotics/darknet_ros)>

The pre-trained model of the convolutional neural network can detect pre-trained classes, including the data set from VOC and COCO, or create a network with custom detection objects. For more information about YOLO<sup>9</sup>, Darknet, available training data, and training YOLO.

### 2.3.6 Whycon

WhyCon<sup>10</sup> is a vision-based localization system that can be used with low-cost web cameras and achieves millimeter precision with very high performance (NITSCHKE et al., 2015). These characteristics allow its use as an alternative to more expensive localization systems. The system is capable of efficient real-time detection and precise position estimation of several circular markers in a video stream. This is the Landing pad used for Video Calibration see Chapter 3.3, State S2 on Table 4. It can be used both off-line, as a source of ground truth for robotics experiments, or online as a component of robotic systems that require real-time, precise position estimation Figure 9. WhyCon is an alternative to widely used and expensive localization systems. It is fully open-source.

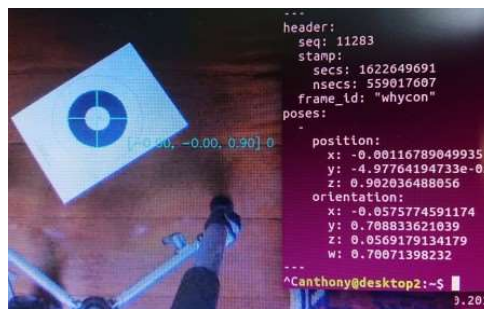


Figure 9 – Whycon ROS implementation marking image center

### 2.3.7 BebopClass

The **BebopClass** class is a superset class of the bebop\_autonomy package. It uses a Structural Design Pattern called Facade (pronounced [*fa-sahd*]), implemented in Python with ROS and Object-Oriented Programming principles. According to (HUNT, 2013) book Gang-of-Four Design Patterns, the concept behind the Facade design pattern is to create a complex system like the Bebop 2 Drone by aggregating instances of smaller subsystems, such as the GPS or the camera. Therefore making it more straightforward to create and, if necessary, modify the complex system. Figure 10 is a visual description of the BebopClass structure using UML (Unified modeling language) (GREGOIRE, 2018), and complete description is shown on appendix C.

<sup>9</sup> <<https://pjreddie.com/darknet/yolo/>>

<sup>10</sup> <<https://github.com/anthonymiglio/whycon>>



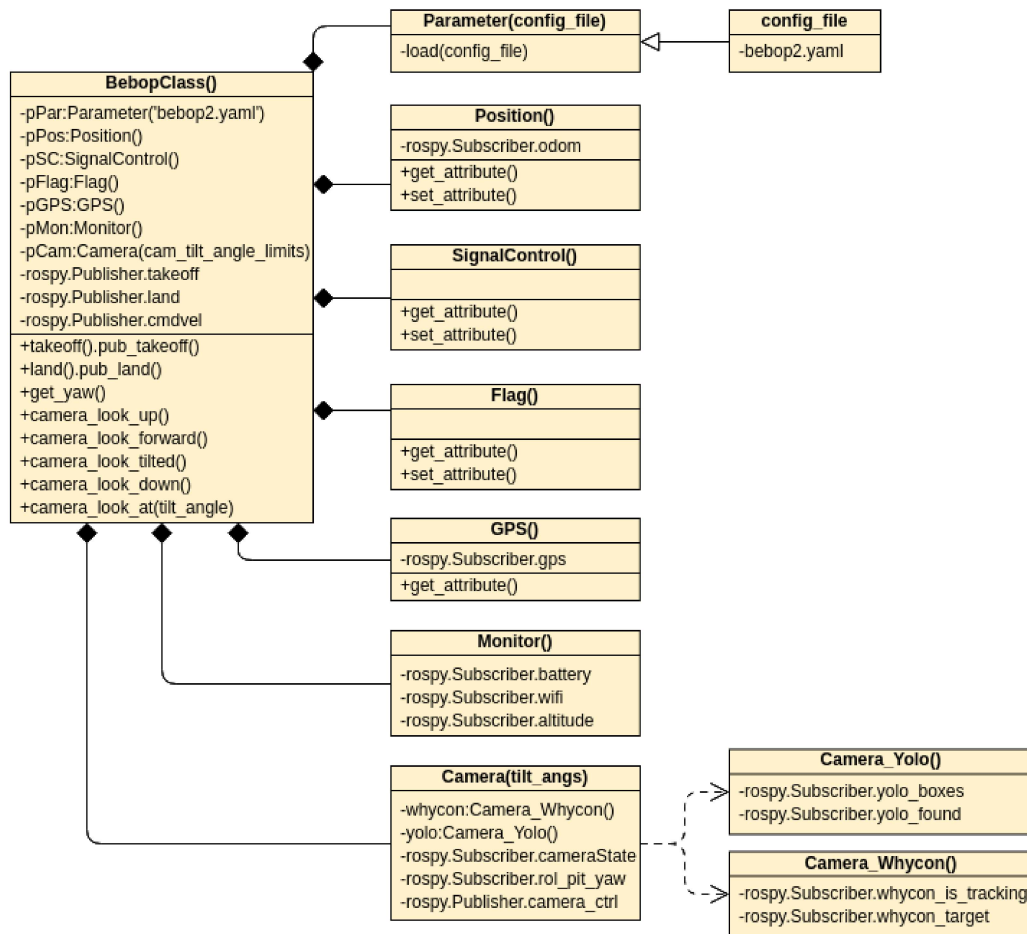


Figure 10 – Design Pattern Facade: BebopClass

### 2.3.8 Exiftool

Exiftool<sup>11</sup> is a command-line utility, technically a Perl open-source library written by Phil Harvey, first released in 2003, that has an approach in Python (TOEVS, 2015). Since then, ExifTool has become the go-to tool for working with metadata at the command line due to the vast array of file formats and types of metadata it supports, enabling read and write standard and custom metadata tags quickly without being a software developer.

To install Exiftool:

```
sudo apt install libimage-exiftool-perl
```

Metadata is information embedded in documents and media typically hidden from view stored as metadata tags or just tags. Tags are key-value pairs embedded into files that store information about that file, such as timestamp, camera info, Global Positioning System (GPS) data, and other necessary information of the picture taken, for instance Figure 11. Tags are classified, like Digital Imaging and Communications in Medicine (DICOM) and Exchangeable image file format (EXIF). Exif is a specification followed by

<sup>11</sup> <<https://exiftool.org/>>

digital camera manufacturers that record information about the technical conditions of image capture and the image file itself in the form of tagged metadata.

```

anthony@desktop2:~$ exiftool -s Bebop2_20200603121944-0300.jpg
ExifToolVersion      : 10.10
FileName             : Bebop2_20200603121944-0300.jpg
Directory            : .
FileSize             : 2.9 MB
FileModifyDate       : 2020:06:03 12:19:50-03:00
FileAccessDate       : 2022:01:04 19:20:17-03:00
FileInodeChangeDate  : 2021:12:29 16:51:21-03:00
FilePermissions      : rw-r--r--
FileType             : JPEG
FileTypeExtension    : jpg
MIMEType             : image/jpeg
ExifByteOrder        : Big-endian (Motorola, MM)
Compression          : JPEG
PhotometricInterpretation : YCbCr
ImageDescription     : {"product_id": "090C", "uid": "25E1C5
01D05F317EC40E48115F993158", "run_date": "2020-06-03T121626-0300", "filen
ame": "Bebop_2_2020-06-03T121944-0300_25E1C5.jpg", "media_date": "2020-06
-03T121944-0300" }
Make                 : PARROT
Model                : Bebop 2
Orientation           : Rotate 270 CW
XResolution           : 72
YResolution           : 72
ResolutionUnit        : inches
Software              : Dragon 4.7.1 ; Impala:0.0 P3A:0.1
ModifyDate            : 2020:06:03 12:19:44
YCbCrPositioning     : Centered
ExposureTime          : 1/2012
FNumber               : 2.3
ExposureProgram       : Action (High speed)
ISO                   : 150
ExifVersion           : 0210
DateTimeOriginal     : 2020:06:03 12:19:44
CreateDate            : 2020:06:03 12:19:44
ComponentsConfiguration : Y, Cb, Cr, -
ShutterSpeedValue     : 1/2012
ApertureValue         : 2.3
MaxApertureValue     : 2.3
MeteringMode          : Other
LightSource            : Daylight
FocalLength           : 1.8 mm
FlashpixVersion       : 0100
ColorSpace            : Uncalibrated
ExifImageWidth       : 4096
ExifImageHeight      : 3320
ExposureMode         : Auto
WhiteBalance          : Auto
FocalLengthIn35mmFormat : 6 mm
GPSVersionID          : 2.2.0.0
GPSLatitudeRef        : South
GPSLongitudeRef       : West
GPSAltitudeRef        : Above Sea Level
ImageUniqueID         : 4DAC98DA443F300D66CAA4207054BBE8
CameraSerialNumber    : P1020574A851677010
Yaw                   : 72.184036255
Pitch                 : -3.650763750
Roll                  : 6.878931046
IntegratedC gyroX     : 315.111267090
IntegratedC gyroY     : -571.003784180
IntegratedC gyroZ     : 10.252861977
AboveGroundAltitude   : 23.729
DataTimestamp         : 338.134460
FrameTimestamp        : 338.132477
ImageWidth            : 4096
ImageHeight           : 3320
EncodingProcess       : Baseline DCT, Huffman coding
BitsPerSample         : 8
ColorComponents        : 3
YCbCrSubSampling     : YCbCr4:2:0 (2 2)
Aperture              : 2.3
GPSAltitude           : 40 m Above Sea Level
GPSLatitude           : 20 deg 18' 45.52" S
GPSLongitude          : 40 deg 17' 1.86" W
GPSPosition           : 20 deg 18' 45.52" S, 40 deg 17' 1.86" W
ImageSize             : 4096x3320
Megapixels            : 13.6
ScaleFactor35efl      : 3.3
ShutterSpeed          : 1/2012
CircleOfConfusion     : 0.009 mm
FOV                   : 143.1 deg
FocalLength35efl      : 1.8 mm (35 mm equivalent: 6.0 mm)
HyperfocalDistance    : 0.16 m
LightValue             : 12.8

```

Figure 11 – Exiftool Metadata Extraction of Fig.30a on Chapter 4.4.1

### 2.3.9 ODM: OpenDroneMap

ODM<sup>12</sup> is an open-source command-line toolkit for processing aerial drone imagery. Typical drones use simple point-and-shoot cameras, so the images from drones, while from a different perspective, are similar to any pictures taken from point-and-shoot cameras non-metric imagery. OpenDroneMap turns those simple images into three-dimensional geographic data and combines them with other geographic datasets into Orthophoto or Orthorectified Imagery (Fig.12a), Point Clouds (Fig.12b), Digital Surface Models (Fig.12c), Textured Digital Surface Models (Fig.12d), Digital Elevation Models, among other models. There is a Graphical version of ODM called WebODM.

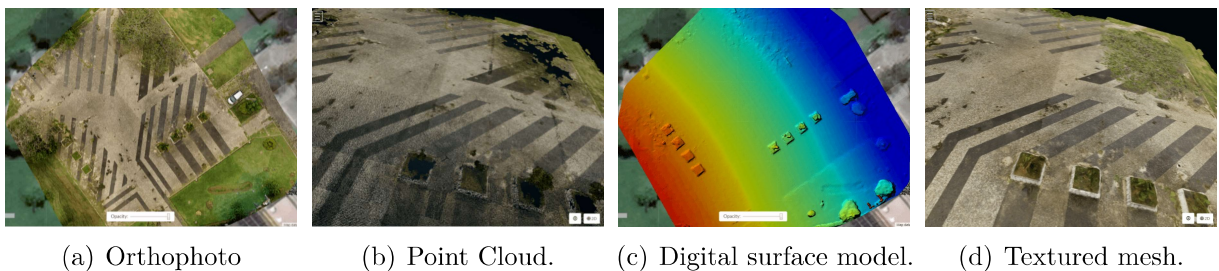


Figure 12 – OpenDroneMap image examples over UFES Rectory Square.

<sup>12</sup> <<https://www.opendronemap.org/>>

## 3 Path Planning & Control

This chapter describes the development of this research of the necessary data to input for path planning to be executed on the loop control and its controllers. After reading it, the reader should understand how the `Bebop_lai` app operate from input data with its GUI interface to execution in a outdoor environment. This chapter is broken down into these three sections: UAV Input Data, Path Panning and Control Loop, and the app work flow is represented in Figure 13.

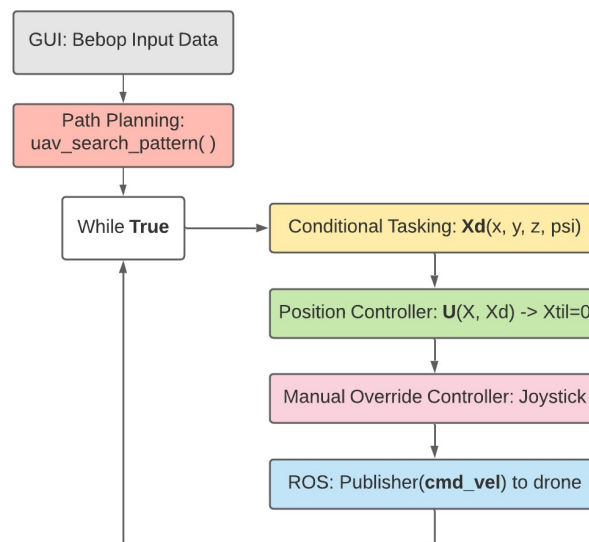


Figure 13 – FlowChart - Main Loop

### 3.1 UAV Input Data

For this project as part of `bebop_lai` package, a GUI (Graphical User Interface) Figure 17 was created to make Path Planning Input Data in an easier and parametric way diminishing human input error. When the `bebop_lai` ROS app run, a browser window will open with Google Maps predefined GPS coordinates Figure 14, that can be adjusted to fit the geographical location. Since Google Maps is an approximated orthophoto map, it is possible to use it to measure distance and angles for Path Planning, by filling the required parametric variables `distance1`, `angle1`, `distance2`, `angle2`, `height`, `forward overlap` and `horizontal overlap`, shown on Figure 15 and 16

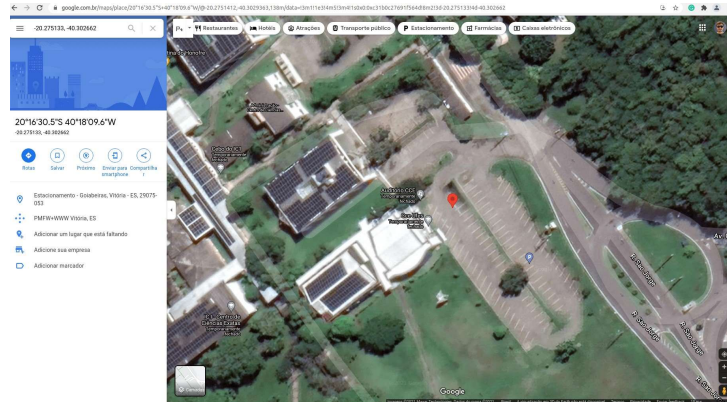
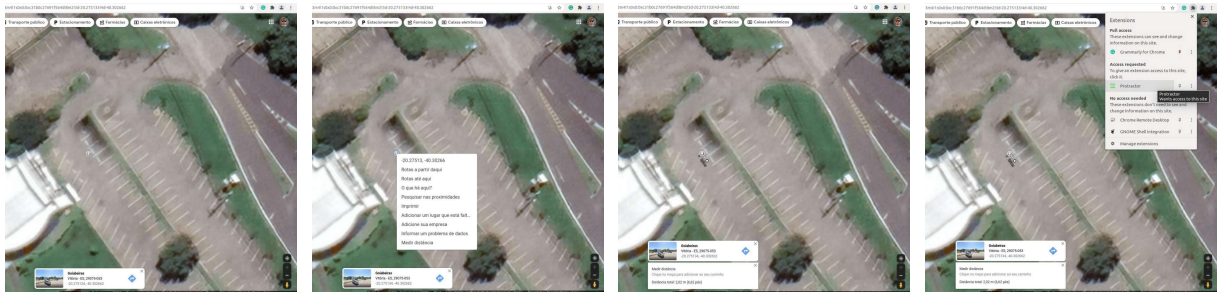


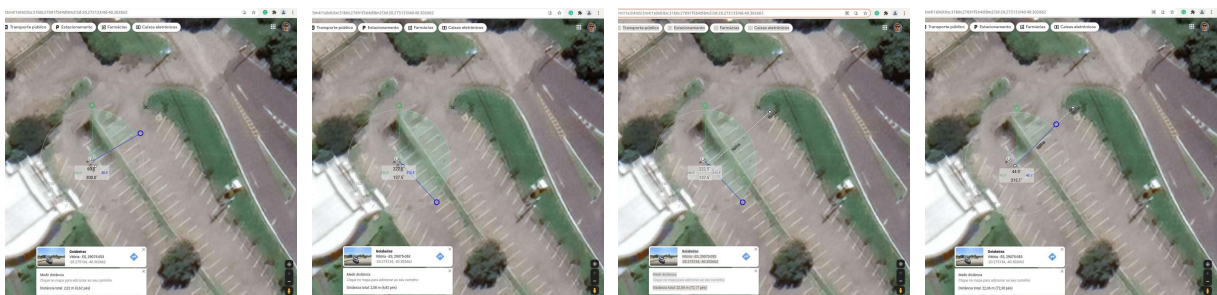
Figure 14 – Browser: Google Maps UFES-CCE Parking Lot



(a) Spot: TakeOff (b) Dist. Measurements (c) dist1 (First Corner) (d) Tool: Protractor

Figure 15 – GUI: Bebop Input Data Steps 1 of 2

- a) Left Click for selecting TakeOff spot.
- b) Right Click for Measuring Distances.
- c) Left Click for dist1, distance from Takeoff to first corner of search area.
- d) Left Click Extensions, then select Protractor.



(a) Place Protractor (b) angl1 (CCW angle) (c) dist2 (total distance) (d) angl2 (CCW angle)

Figure 16 – GUI: Bebop Input Data Steps 2 of 2

- a) Place Protractor Center over TakeOff spot > Green line point up (North).
- b) angl1 = Angle CCW from Green to Blue Line align with first line segment.
- c) Left Click for dist2 = dist1 + distance from first to second corner of search area.
- d) angl2 = Angle CCW from Green to Blue Line align with first line segment.

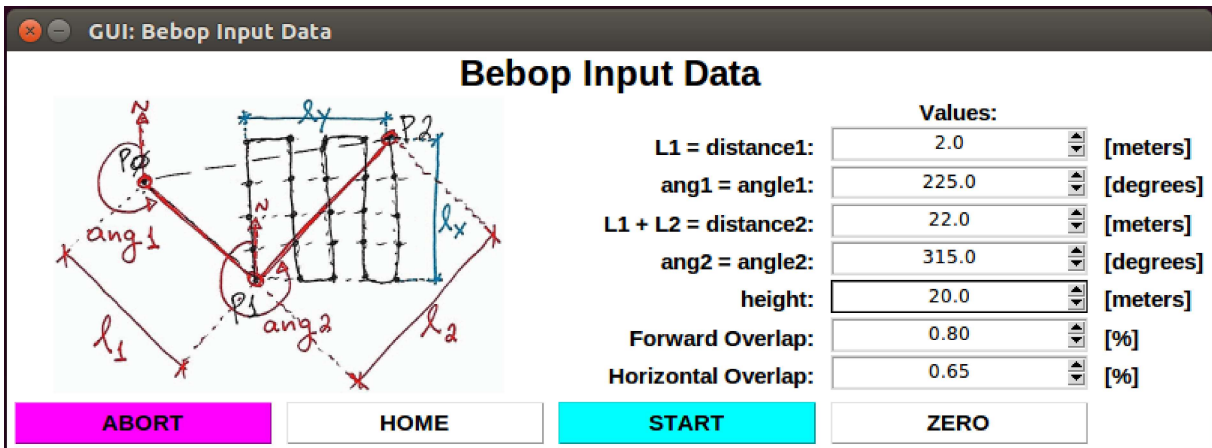


Figure 17 – GUI: Bebop Input Data Filled

## 3.2 Path Planning

Path planning or Motion planning is a computational problem to find a sequence of valid configurations that moves the object from the source to the destination. Consider navigating a mobile robot inside a building to a distant waypoint for illustration. It should execute this task while avoiding walls and not falling downstairs. A motion planning algorithm would describe these tasks as input and produce the speed and turning commands sent to the robot's wheels. In this thesis, since the Bebop 2 drone has only one high-resolution camera and is facing down in a Bird-eye-view during operation, Obstacle Avoidance is not on the scope since the camera must be facing the same direction as the UAV motion.

### 3.2.1 UAV Obstacles

The obstacles can be physical Fig.18a, Fig.18b, Fig.18c or interference in nature Fig.18d, Fig.18e, Fig.18f. To overcome the physical obstacles that are not unsurmountable, the operator has a ground point-of-view from the desired search area and a satellite view from Google Maps Fig.14; thus, the robot will rise to a considerable safe flight height (stipulated by the operator) Fig.17 in an unobstructed environment (SCHULDT; KURUCAR, 2018). Nevertheless, the interference obstacles must be dealt with by signal treatments, such as adjusting WiFi signal frequency or avoidance, for instance, flying from 10:00h to 14:00h during sunny days to avoid rain and long cast shadows.

*In this thesis, no wildlife was injured or hurt during the outdoor experiments. Avoid wildlife at any cost, and the ones involved in the UAV project must understand that the drone is the intruder in the wildlife environment. For instance, Fig.18a shows Aggressive Birds Are Attacking Google's Delivery Drones in Australia.*

Source: Fig.18a<sup>1</sup>, Fig.18b<sup>2</sup>, Fig.18c<sup>3</sup>, Fig.18d<sup>4</sup>, Fig.18e<sup>5</sup>, Fig.18f<sup>6</sup>.

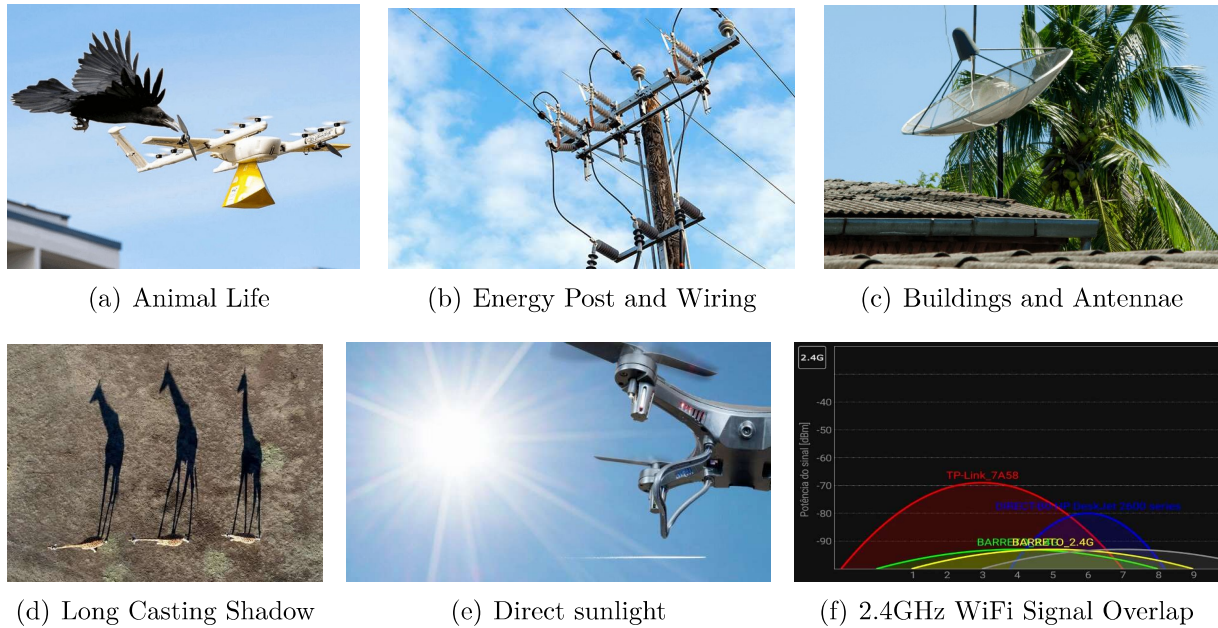


Figure 18 – UAV Obstacles: Physical and Interference

### 3.2.2 UAV Search Patterns

The 1979 Convention, adopted at a Conference in Hamburg, was aimed at developing an international maritime SAR<sup>7</sup> (Search And Rescue) plan so that, no matter where an accident occurs, the rescue of persons in distress at sea will be coordinated by a SAR organization and, when necessary, by co-operation between neighboring SAR organizations. Based on the developments of this convention, some recent research adapted SAR techniques for UAVs Fig.19 (SCHULDT; KURUCAR, 2018) to support with the same operational force and a few drones to cover a broader area in a shorter time to do the greater good to rescue lives at sea (KARAMANOU et al., 2018), and in land (POLKA; PTAK; KUZIORA, 2017), whilst others uses swarm drone behavior (ARNOLD; YAMAGUCHI; TANAKA, 2018).

<sup>1</sup> <<https://screenrant.com/wing-drone-delivery-bird-attacks/>>

<sup>2</sup> <<https://www.cniguard.com/latest-news/overhead-power-lines/>>

<sup>3</sup> <<https://epicroofs.com/top-5-reasons-not-to-install-a-satellite-dish-on-your-roof/>>

<sup>4</sup> <<https://roboticsbiz.com/breathtaking-drone-photos-in-the-world-drone-photography/>>

<sup>5</sup> <<https://www.thedronegirl.com/2021/04/06/reduce-glare-flying-drones-controller/>>

<sup>6</sup> WiFi Anilizer App for Android

<sup>7</sup> <[https://ec.europa.eu/home-affairs/pages/glossary/search-and-rescue-sar-operation\\_en](https://ec.europa.eu/home-affairs/pages/glossary/search-and-rescue-sar-operation_en)>

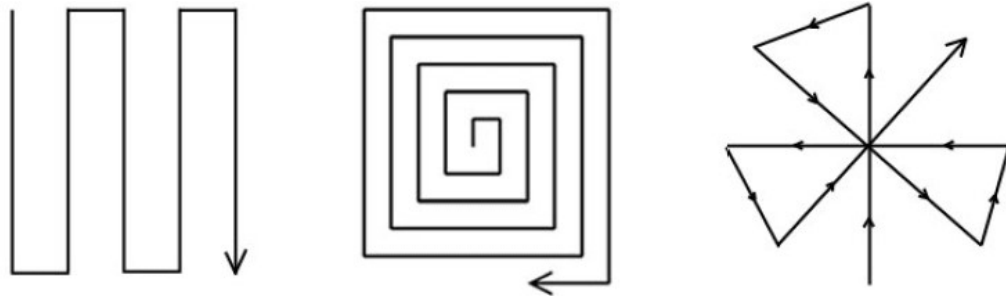
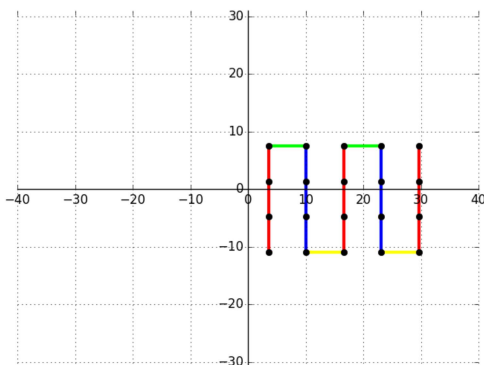


Figure 19 – Search Patterns: Creeping Line, Increasing Square, Sector

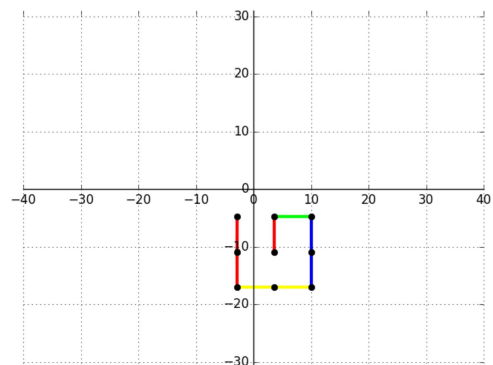
Table 4 – SAR Search Patterns Comparison

SAR Patterns	Creeping Line	Increasing Square	Sector
Area Covered	Large / Medium	Medium / Short	Short / tiny
Start at	Close to the operator	Specific coordinate	Current coordinate
Victim lost for	some time	short time	recent sighted
Victim location	not sure	some sure	Know the location
Sea/wind Currents	Against or parallel	without current	with and without

The aerial surveillance search techniques of this proposed work are the same as on the research cited in the previous paragraph and shown in Figure 19, and a comparison between the pattern is presented in Table 4 to clarify the moment to use each one. Based on these SAR (Search and Rescue) techniques, uses only two patterns implementation: Creeping line Figure 20a for a broader area and Increasing Square Figure 20b for a more refine Figure 21 or the need to create the orthophoto from the victims found coordinate. The victim here is the Yolo Custom trained Classification for Object Detection on Chapter 4, and in this work, the first classification is Swimming Pool for the Custom UAV Dataset Chapter 4.3.

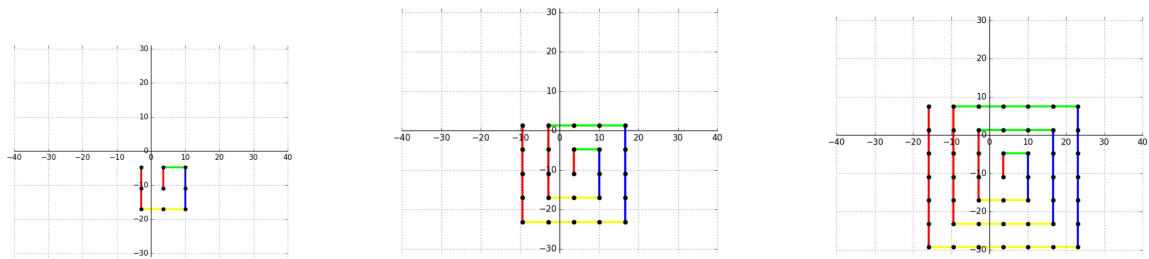


(a) Creeping Line



(b) Increasing Square

Figure 20 – UAV Search Pattern: Creeping Line and Increasing Square



(a) Increasing Square 1 turn      (b) Increasing Square 2 turns      (c) Increasing Square 3 turns

Figure 21 – UAV Search Pattern: Increasing Square Evolution

### 3.3 Conditional Tasking

The Conditional Tasking is a Behavior Controller and the first controller on the Loop Control of the Bebop\_lai app. This module is parametric and divided into ten states as described in Table 5 and Conditional Tasking Overall Layout shown in Figure 22. Depending on the Input Data Section 3.1, it will execute Path Planning by checking if the conditions of each state have been fulfilled to achieve the state task and changing the UAV desired position  $X_d$ , allowing to go for the next state at another desired position, consequently reaching the final state and completing the surveillance task.

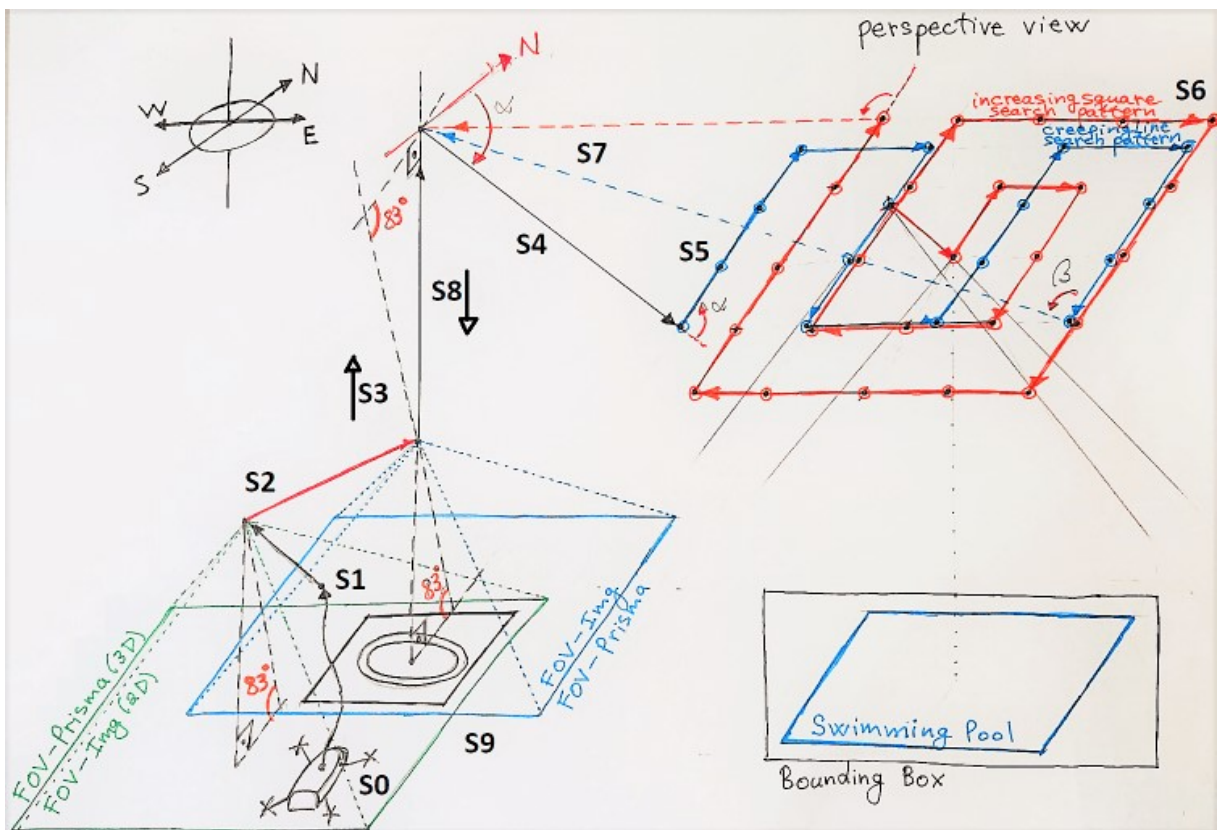


Figure 22 – Conditional Tasking Overall Layout



Table 5 – Conditional Tasking States

State	Name	Description
S0	Wifi Connection	Wifi connection is established
S1	Takeoff Procedure	All necessities steps for Takeoff
S2	Visual Calibration	Calibrate landing pad as the origin
S3	Ascend to Height	Ascend to UAV unobstructed environment Height
S4	Goto Pose	Go from Home to first search pattern point
S5	Surveillance	UAV displaces according to path planning
S6	Mission Control	Repeat previous State or go Home
S7	Home	Go from last search pattern point to Home
S8	Landing Procedure	All necessities steps for Landing
S9	Save and Power Off	Save flight data and finish app

The Conditional Task has some similarities with a state machine, but one significant difference is that it does not stay in the state until it finishes the task; it works in the control loop; therefore, it enters a state and will leave on the same state no task done Figure 23. In the case of a task accomplished, it will switch to the next state unless abort occurs and follow through a safe return path. Each state has this flowchart Figure 24, and the only difference is the recurrent block depicted in Appendix C that holds the main state action.

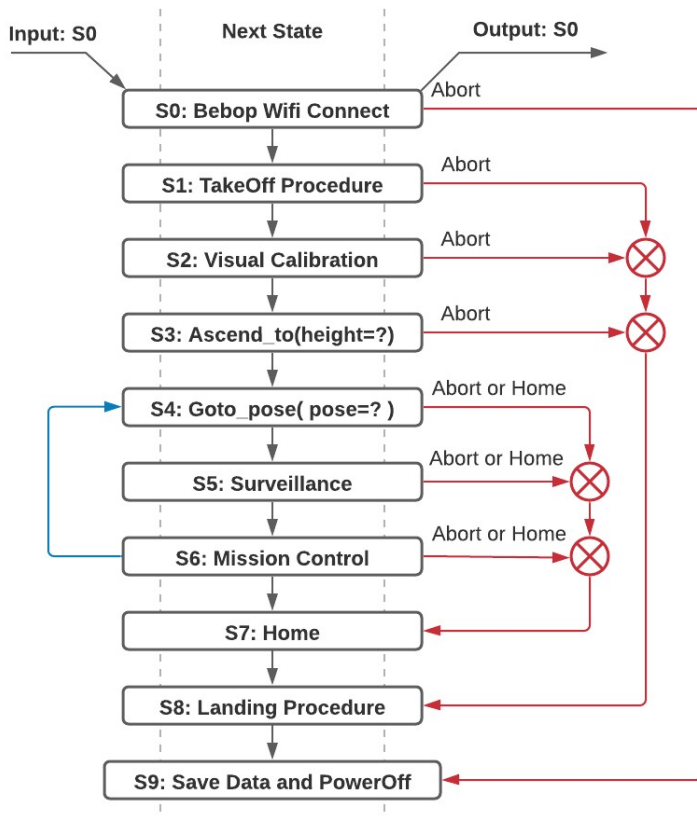


Figure 23 – Flowchart Overview

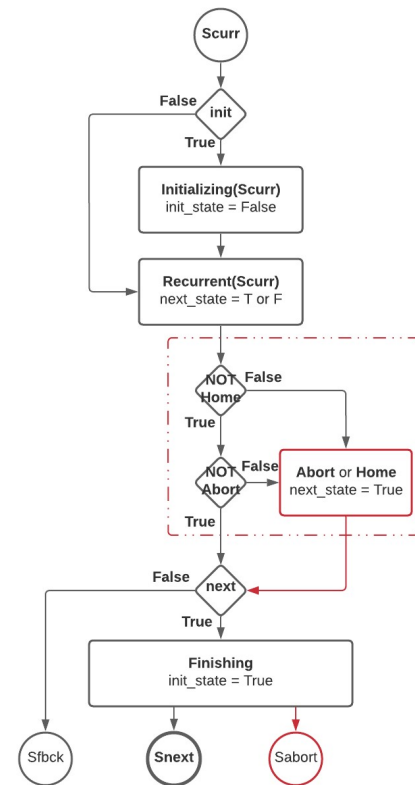


Figure 24 – Flowchart State

## 3.4 Position Controller

This thesis adopted a position control strategy for the UAV. The Bebop 2 dynamic controller characterized in Equation (2.10), initially proposed in (SANTANA et al., 2014), was used to obtain the  $\mathbf{U}_d$  signal. To characterize the desired robot position, the actual robot position in the global coordinate system, given by  $\mathbf{X}$ , measured through the drone's internal odometry that uses the data from its GPS as sensor fusion in its internal controller, and its derivative  $\mathbf{X}'$ , obtained numerically from  $\mathbf{X}$ . The Position Controller creates a control variable  $\tilde{\mathbf{X}}$ , the difference between desired position and current position, and work to decrease this control variable to zero by calculate the instant  $\mathbf{U}$  speed to take the UAV from its current position to the desired position, as described on (PINTO et al., 2020).

## 3.5 Override Manual Controller: Joystick

Together with the `Bebop_lai` Class, the **Override Manual Controller** is a class instantiated as a global variable; therefore, it is hierarchy superior to the other controller that are local instances of the Main algorithm. Every control loop cycle; Conditional Task Controller sets the desired position  $\mathbf{X}_d$ ; the Position Controller calculates the speed  $\mathbf{U}$ ; the Override Manual Controller keeps and watchdog over the analog Joystick checking for a minimal movement threshold. If the Joystick moves (configuration Chapter 2.1.4), the Position Controller speed ROS message `cmd_vel` is overwritten with the Joystick `cmd_vel` while actuated for the time being. Otherwise, the Joystick movement goes down the threshold, and the Position Controller establishes the UAV `cmd_vel`.

## 3.6 ROS Publiser `cmd_vel`

At the end of every control loop cycle, a ROS Publisher keeps sending the ROS message `cmd_vel` to the Bebop 2 at a rate of 5Hz. Moreover, there is only one ROS Publisher for each published topic to avoid duplicated commands. Before the Publisher, ROS message is only information, but it becomes command actions sent to the UAVs and converts into movement.

## 4 Computer Vision

### 4.1 Computer Vision

This chapter will explain the different tasks in Computer Vision, focusing on Object Detection, the main task for this project, some applications, concepts, and a few methods.

#### 4.1.1 Computer Vision Tasks

Nowadays, some of the most research topics in Computer Vision are Image Classification, Image Localization, Object Orientation, and Image Segmentation.

Human vision exploits natural intelligence (SINHA, 2020), while artificial intelligence controls computer vision, and because of this, we have much superior comprehension of Vision. Looking at Figure 25, we can see different tasks<sup>1</sup> performed by Computer Vision. We have image classification, image localization, object detection, and image segmentation from left to right.

Human beings can perform these tasks quite easily, while artificial intelligence needs a lot of 2D data (ZHANG, 2010) to train Machine Learning (ML) Models and use Deep Learning Networks to perform these tasks in real-time.

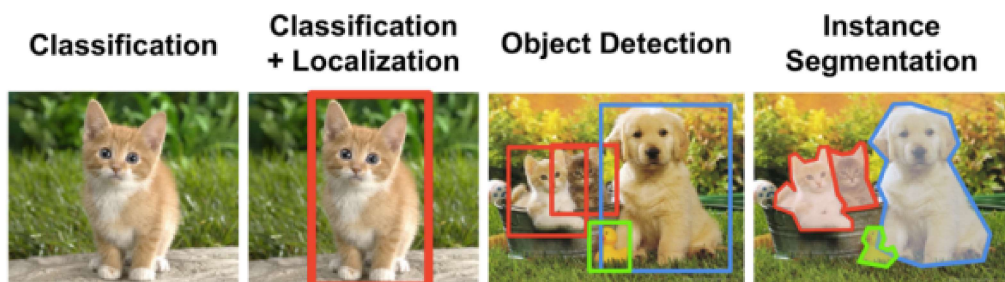


Figure 25 – Computer Vision Tasks. Image from ClimateNet Website  
Source: ClimateNet Homepage

#### 4.1.2 Object Detection

Object Detection is a computer technology related to computer vision and image processing tasks where ML models detect various objects within a digital image or video, predicting their corresponding classes and position with bounding boxes. Well-researched domains of object detection include face detection, face recognition, and pedestrian

<sup>1</sup> <<https://www.nersc.gov/research-and-development/data-analytics/big-data-center/climatenet/>>

detection, including image retrieval and video surveillance, during real-time or post-processing.

A comparison between different Object Detection methods with various detectors is performed in (BOCHKOVSKIY; WANG; LIAO, 2020), regarding speed and accuracy. These methods fall into either neural network-based or non-neural approaches. The later group (Viola–Jones object detection based on Haar features, Scale-invariant feature transform (SIFT)(LINDEBERG, 2012), Histogram of oriented gradients (HOG) features (DALAL; TRIGGS, 2005)) demands the definition of features using some method, in association with a technique to classify them, such as a support vector machine (SVM). In opposition, deep neural network techniques (Region Proposals R-CNN (GIRSHICK et al., 2014), Fast R-CNN (GIRSHICK, 2015), Faster R-CNN (REN et al., 2015), cascade R-CNN (CAI; VASCONCELOS, 2021)), Single Shot MultiBox Detector (SSD) (LIU et al., 2016), You Only Look Once (YOLO) (REDMON et al., 2016) (REDMON; FARHADI, 2017) (REDMON; FARHADI, 2018) (BOCHKOVSKIY; WANG; LIAO, 2020) (WANG; BOCHKOVSKIY; LIAO, 2021), Single-Shot Refinement Neural Network for Object Detection (RefineDet) (ZHANG et al., 2018), Retina-Net (LIN et al., 2020) (PANG et al., 2019), Deformable convolutional networks (ZHU et al., 2019)) can do end-to-end object detection without defining handcrafted features, being typically based on convolutional neural networks (CNN).

## 4.2 Yolo: Real Time Object Detection

YOLO is a state-of-the-art real-time object detection system for images that uses a CNN-based deep neural network, written in Darknet<sup>2</sup>, a fast open-source neural network framework written in C and CUDA with computations in CPU or GPU. While previous detection systems used the sliding window technique, Yolo only needs one evaluation to predict the object class probability and the bounding box (BBox). And it uses the metric IoU (Intersection over Union), which is a metric for comparing the similarity of a set of samples between predicted BBox and annotated BBox. It deals with object detection as a regression problem to compose and associate the separated parts into bounding boxes and class probabilities.

The base YOLO model architecture, illustrated in Figure 26, is fast enough to process in real-time images captured at 45 frames per second (FPS) on a computer (CPU: i7-5820K RAM 32GB with GPU: Titan X), but this depends on the framework. The Darknet framework runs better with GPU, while for CPU computations the better choice, according to (STADNIK; SAZHIN; HNATIC, 2020), is the OpenCV framework. There is also a smaller version (called tiny-yolo), lighter in terms of memory usage, and faster than

<sup>2</sup> <<https://pjreddie.com/darknet/>>

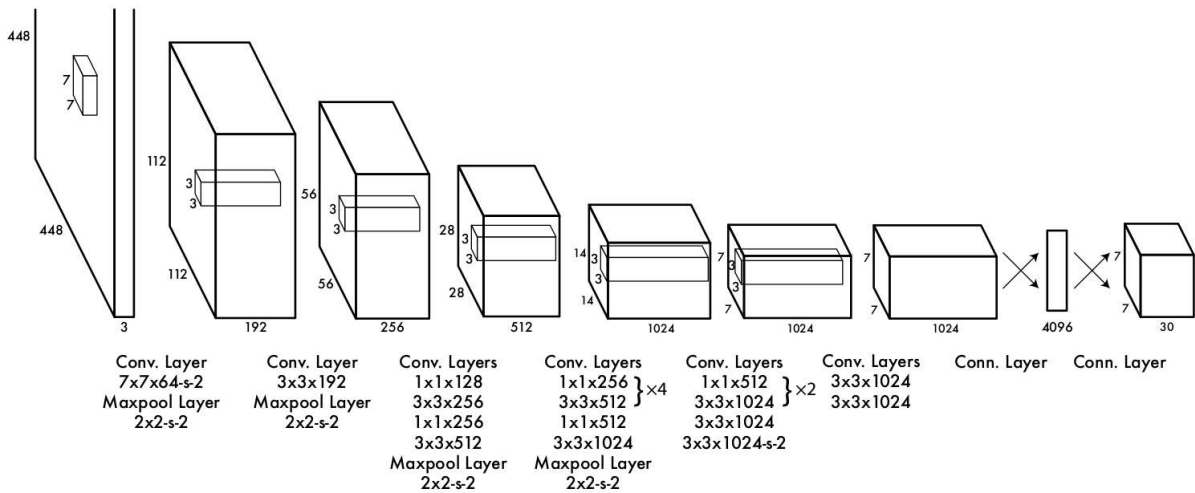


Figure 26 – Yolo version 1 Network Architecture (REDMON et al., 2016)

the full version.

Even though YOLO is prone to more localization errors and far less likely to predict false detections than the other previous state-of-the-art detection methods, like RCNN, Fast RCNN, and Faster RCNN, it outperformed all of these methods in speed and keeping a high average precision. Because its network only looks at parts of the image with a high probability of containing the objects.

When launched, YOLOv2 performed from twice up to 10 times faster than the others and higher AP (Average Precision). YOLOv3 was a significant improvement, in terms of FPS and AP, the same happening when launching YOLOv4, as one can see in Figure 27, which compares the performance of YOLOv3 and YOLOv4 with other object detection algorithms. As one can see, YOLOv4 has a good real-time performance, including in terms of FPS. However, not everything worked along the way: YOLOv3 did not performed well with Anchor box (x, y) offset predictions, Linear (x, y) predictions,

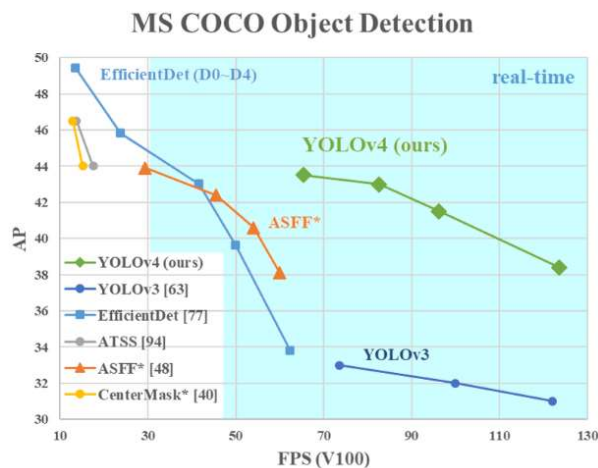


Figure 27 – YOLOv4 Performance Graph. AP(Average Precision) vs FPS

Table 6 – Comparison of YOLO Versions Performance for the *dog.jpg* sample photo

YOLO Version	YOLOv1	YOLOv2	YOLOv2-tiny	YOLOv3	YOLOv3-tiny	YOLOv4	YOLOv4-tiny
BFLOPS	40155	29475	5412	65879	5571	128459	6910
Workspace Size [MB]	104.87	131.08	52.43	39.83	19.57	52.44	19.57
Loaded layers	32	32	16	107	24	162	38
Prediction Time [s]	0.727524	0.363412	0.343773	0.394389	0.348965	0.421282	0.349144
Probability:							
bicycle	0.39	0.84	0.84	0.99	0.39	0.92	0.59
car	0.74	-	0.67	-	0.71	0.22	0.40
dog	0.26	0.79	0.52	1.00	0.81	0.98	0.84
motorbike	-	-	0.50	-	-	-	-
*person	-	-	0.43	-	-	-	-
*pottedplant	-	-	-	-	-	0.33	-
truck	-	0.78	-	0.93	0.61	0.92	0.80

\*The object: Person and pottedplant, are False Positive found by YOLO.

and Focal loss.

For comparison reason between Yolo versions, Table 6 synthesizes the performance of the YOLO versions on the sample photo *dog.jpg* on this hardware (CPU: i7-4790 8x3.60GHz 16GB with GPU: GeForce GTX 1650). A difference between Yolo versions is the number of deep layers inside the CNN and its structure; this is directly related to performance (the execution time and probability of detection of objects).

This research uses YOLO full version. It is more precise and reliable than its tiny version developed for embedded systems, and we chose YOLOv3 because it must be compatible with the ROS version UAV application that uses an implementation of darknet compatible up to YOLOv3 and OS Linux Ubuntu 16.04. And not YOLOv4 because it is another ROS module implementation and not compatible with the OS of this application.

### 4.3 Dataset

YOLO is a supervised technique, so it needs to be trained by an image dataset for several epochs to perform well. One approach is to grab a camera and move around while taking pictures in different angles of an object, such as in (Li et al., 2019), work which created the Richly Annotated Pedestrian (RAP) dataset. Alternatively, the dataset DOTA (XIA et al., 2018) is a large-scale aerial image dataset that uses a huge amount of satellite images labeled by aerial image experts, from six international science departments, into 15 common object categories. Even though one of the classes is a swimming pool, the image information is not ideal for a quadrotor flying at up to 30 meters of altitude. Moreover, nowadays high-resolution drone photography cameras are full-HD and 4K.

### 4.3.1 Image data acquisition

As it would be an invasion of privacy to fly with a drone over people’s homes to collect swimming pool photos, this work proposes to use websites specialized in photography<sup>3</sup> with licenses for non-commercial use by searching and downloading images from the Internet. The same image selection technique (GAUEN et al., 2017) occurs in many renowned datasets, such as PASCAL VOC (EVERINGHAM et al., 2014), COCO (LIN et al., 2014), SUN (XIAO et al., 2010), and ILSVRC (RUSSAKOVSKY et al., 2014), for instance.

According to the images required to create this Dataset (the list with 150 images for download available on GitHub.<sup>4</sup>), many high-resolution photos Fig.28 in 4K and FULL-HD at 72dpi of the desired objects on sunny day in birds-eye view are necessary for training a YOLO model to classify an object of interest and locate it in the frame. The purpose of this is to aid the UAV localization and decision-making based on computer vision, considering classified objects. An excellent example of these objects is a swimming pool detected and used as a visual reference to take pictures in the most suitable poses respecting image overlap criteria for georeferencing mapping by post-processing.

Before starting flying for data acquisition, ANAC (the National Civil Aviation Agency of Brazil) recommends to consult local weather forecast<sup>5</sup>, to plan the flight, and to

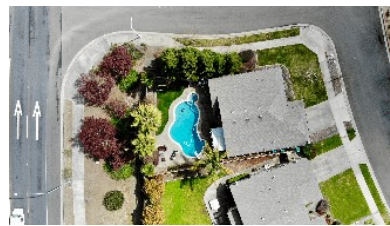
<sup>3</sup> Such as: <<https://unsplash.com/>>, <<https://www.pinterest.com/>>, <<https://pixabay.com/>>, <<https://www.dreamstime.com/>>, <<https://www.pexels.com/>>, <<https://www.istockphoto.com/>>

<sup>4</sup> <<https://github.com/anthonymiglio/YoloV3TrainingCustomModel>>

<sup>5</sup> <https://www.climatempo.com.br/vento/>



(a) Photo: Guilherme Schneider



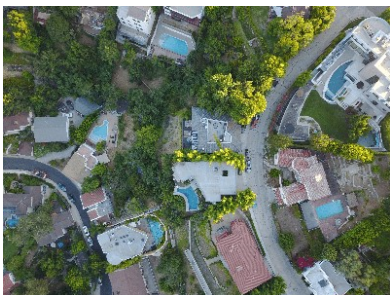
(b) Photo: Carles Rabada



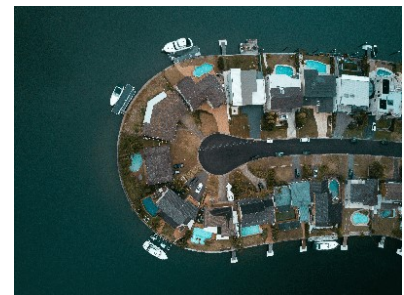
(c) Photo: Avi Waxman



(d) Photo: Chuttersnap



(e) Photo: Bart Jaillet



(f) Photo: Mudassir Ali

Figure 28 – Downloaded photos from Unsplash website

keep the Visual Line of Sight recommended in its website section *Drones and Meteorology*<sup>6</sup>. Furthermore, one should stop the flight if it starts raining, in case of a fog or when the speed of wind gusts exceeds 30 km/h.

### 4.3.2 Image data augmentation

An image processing technique is image rotation. It makes the model more robust and versatile when detecting an object that can appear in any orientation. According to Rfid-CNN technique (CHENG et al., 2019), object detection has several significant challenges, including object rotation, within-class diversity, and between-class similarity. This is why a sports court has been detected and classified as swimming pool class in the image shown in Figure 29a, but not in every scene of Figure 29b. Even though they are not in the classes of this Dataset, it has intrinsic similarities with the swimming pool class from a birds-eye point of view. Therefore, classes of objects similar to the desired one should be added to the Dataset to avoid these mistakes, improving the search. Another reason is that professional photographers have a knack for framing and aligning their pictures. Therefore, this is a must for downloading high-resolution images. Furthermore, it increases the number of images in the Dataset, since they are not as easy to get. Moreover, remember that to get well-shot photos light is essential. Sunlight between 10 a.m. and 2 p.m. is a good choice to minimize shadows effect on the image.

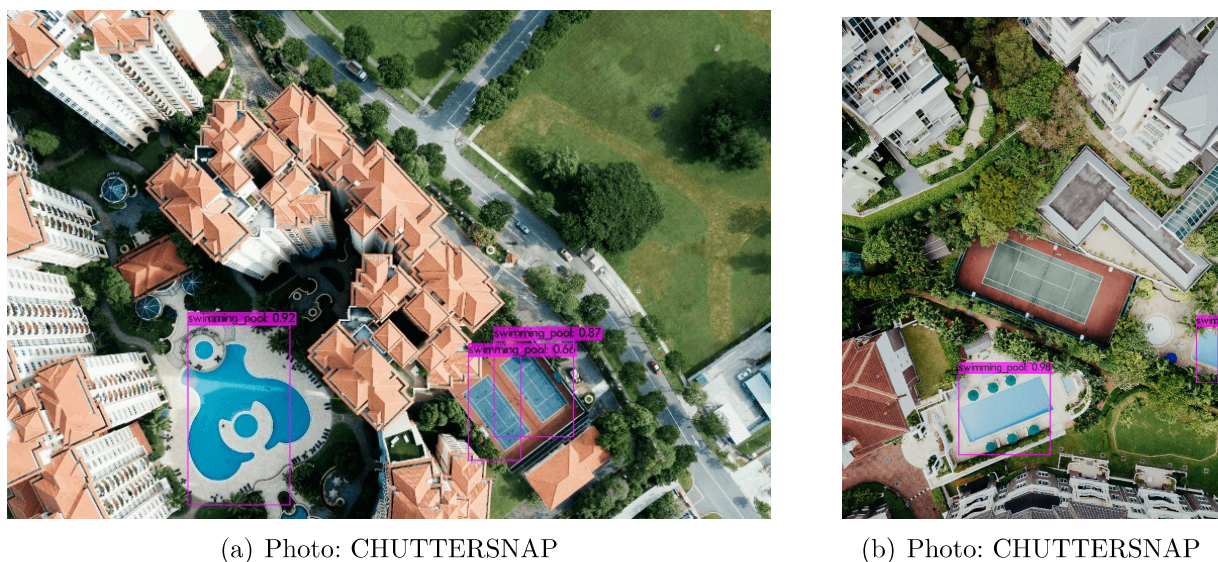


Figure 29 – Downloaded from Unsplash.com

### 4.3.3 Image annotation

After downloading the images, one should select objects by marking them with a bounding box, and choose a class for each one, in each image, since YOLO is supervised

<sup>6</sup> <https://www.anac.gov.br/en/safety/aeronautical-meteorology/drones-and-meteorology>



Table 7 – Comparison between some common annotation formats

Annotation Comparison	COCO	YOLO	DOTA
File	JSON	TXT with same name	TXT with same name
Object Instance	One per line	One per line	One per line
Bounding Box	BB	BB	OBB (Oriented Bounding Box)
<b>BB Variables</b>	ID, x, y, width, height ID = Category ID x = BB_top-left x y = BB_top-left y w = BB_width h = BB_height	ID x y width height ID = object-class x = BB_center_x y = BB_center_y w = BB_width y = BB_height	x1, y1, x2, y2, x3, y3, x4, y4, ID, difficult OBB 1st point is Yellow = (x1, y1) OBB 2nd point clockwise = (x2, y2) OBB 3rd point clockwise = (x3, y3) OBB 4th point clockwise = (x4, y4) ID = category difficult (from 0.0 to 1.0)
Observations	have more object variables outside the Bounding Box. Category IDs start from 1	x, y, w, h are relative to Image Width and Height.	Annotation File contain Metadata: image acquisition date, satellite source, GSD. (Ground sample distance) in meters

and needs this for training. After, it is necessary to create a TXT file for each image with the data of each classified object found in the image as per standard YOLO annotation, using the description `<ID> <left_x> <top_y> <width> <height>` for each object in all the training images.

It is a tedious job to manually identify each object and tag it on each image. Fortunately, there is a software called **LabelImg.exe**<sup>7</sup> that simplifies and minimizes human error by assisting labeling and doing the necessary math from the coordinates of a two clicks bounding box into the YOLO annotation standard.

Different Object Detection and Segmentation models use different object annotations (labels) formats, as shown in Table 7. It is simple algebra to convert COCO annotation to YOLO since image width and height are the same first-order equation system. On the other hand, to Convert from DOTA annotation to YOLO is not trivial, but there is a code to do this<sup>8</sup>, in which x, y, width, and height are relative to the corners, width and height of the bounding box delimiting a classified object in the image. The code to convert the format is *data\_transform/YOLO\_Transform.py*.

#### 4.3.4 Training YOLOv3 Dataset on Google Colab

In this repository<sup>9</sup> there is the code with instructions to train YOLOV3 Custom Dataset.

After 3350 epochs with an average loss of 0.079327, YOLOv3 training stopped because it exceeded the Google Colab Free GPU (NVIDIA Tesla T4 16GB) processing time limit with remaining 2h40min to go from the initial preview. The algorithm Yolo uses data augmentation internally on the 150 original images used for training.

<sup>7</sup> [<https://tzutalin.github.io/labelImg/>](https://tzutalin.github.io/labelImg/)

<sup>8</sup> [<https://github.com/ringringyi/DOTA\\_YOLOv2/tree/master/data\\_transform>](https://github.com/ringringyi/DOTA_YOLOv2/tree/master/data_transform)

<sup>9</sup> [<https://github.com/anthonymiglio/YoloV3TrainingCustomModel>](https://github.com/anthonymiglio/YoloV3TrainingCustomModel)

## 4.4 Testing of Dataset

This section presents the visual result showing how the model behaves in different scenarios and image quality (high and low resolution). Hardware used (CPU: i7-4790 8x3.60GHz 16GB, GPU: GeForce GTX 1650) run at 15 FPS.

### 4.4.1 High Resolution Images taken by UAV

The YOLOv3 Custom Model detected the object (swimming pool) successfully in the three images of Figure 30a, a fish-eye aerial photography of resolution 4096 x 3072, Figure 30b, a frame removed from aerial video of 1080p resolution, and last but not least, Figure 30c, a PrintScreen from ROS Steaming UAV camera at 856 x 480 pixel. The model is robust enough to detect objects captured by different types of lenses and in various resolutions. On the other hand, it is not perfect, as we can see that it lacks some classes, like sports courts and cars in bird's-eye view images. Due to similar features, misjudgement can occur. Following for a video demonstrating the Yolov3 Custom Model in action: [https://www.youtube.com/watch?v=fKb\\_Imok1gw](https://www.youtube.com/watch?v=fKb_Imok1gw)



(a) Bebop 2 Fish-eye Photography

(b) Bebop 2 Video Frame

(c) ROS Streaming Bebop 2 Camera

Figure 30 – Validation in High Resolution

### 4.4.2 Low Resolution Images taken by Satellite

A total of 72 satellite images from Google Maps, tested and visually analyzed for the city of Vitória, ES, Brazil, Figure 31(a, c and e), at Ilha do Boi neighborhood and for a sunny Las Vegas, NV, USA, Figure 31(b, d and f). Divided evenly among the locations into three scales (5, 10 and 20 m/inch), and respectively outcome for True Positives: Ilha do Boi 49.52%, 29.16%, 0.0%; Las Vegas 94.36%, 69.23%, 20.83%. Even though one is a semi-tropical island in the southern hemisphere and the other is a desert area in the northern part of the Globe, these cities were selected by exhibiting a similar neighborhood visual environment, ample housing, extensive gardens, many pools, sunny days. However, one attribute of the satellite images is that the Las Vegas pictures are cleaner and with

slightly higher definition, consequently aiding the model in finding more pools than in Ilha do Boi.



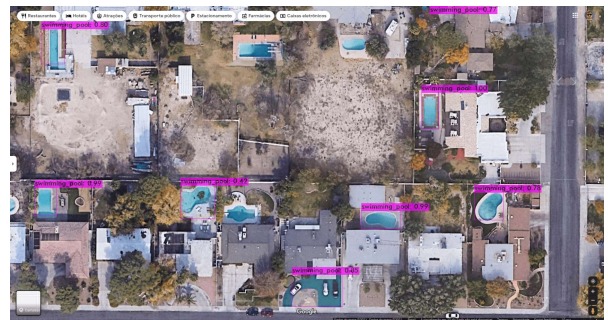
(a) 5 m/inch scale. 5 of 9 objects detected



(b) 5 m/inch scale. 7 of 7 objects detected



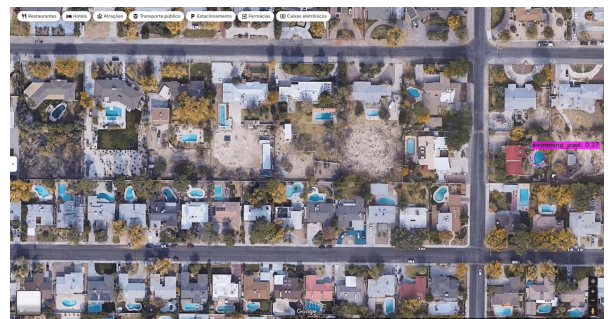
(c) 10 m/inch scale. 4 of 13 objects detected



(d) 10 m/inch scale. 8 of 11 objects detected



(e) 20 m/inch scale. 0 of 41 objects detected



(f) 20 m/inch scale. 1 of 40 objects detected

Figure 31 – Validation in Low Resolution Images from Ilha Do Boi and Las Vegas.  
Source: Google Maps

## 5 Post-processing

### 5.1 Video Export

To extract the video file into the computer, connect the Bebop 2 rear USB-micro port Figure 32 to the computer via USB-micro/USB-A cable. Switch the Parrot Bebop 2 on. The computer detects the Parrot Bebop 2 automatically and the Photos application opens. Copy the last video to the computer.

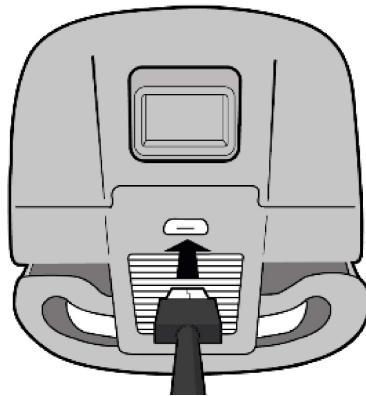


Figure 32 – Bebop 2 USB Port for Video Extraction  
source: Parrot Bebop 2 Drone - User Guide

### 5.2 Video Conversion

Unfortunately, for some reason of incompatibility between ROS OpenCV module `ros-kinetic-opencv3-3.3.1` and the Bebop 2 video mp4 parameters and the following error will come up:

```
1 OpenCV Error: Assertion failed (size.width>0 && size.height>0) in imshow, file /tmp/
  binarydeb/ros-kinetic-opencv3-3.3.1/modules/highgui/src/window.cpp, line 331
2 Segmentation fault (core dumped)
```

To overcome this issue, use VLC media player 3.0 to work around and convert the original mp4 Bebop 2 Video File to another mp4 file; otherwise, OpenCV will signal an Error message and not be able to extract frames from the original mp4 Video File.

### 5.3 Frame Extraction

Access rosbag file to grab the timestamp numbers of the GPS coordinates collected at the points of the desired position according to path planning. Subtract by the Takeoff timestamp to get the mp4 Video time reference that starts at zero seconds.

Access mp4 Converted Video Properties to get the new FPS of the video to calculate the frame number. Divide the time of the video events by the FPS and round to get the frame number. Use OpenCV to extract these frames from the mp4 Converted Video.

## 5.4 Metadata Imputation

As introduced in Chapter 2.3.8, Metadata is information within the photography that can tell a lot about the image, such as timestamp, lightning parameters, camera technical data, georeference information, and much more. The frames extracted from the video have no metadata, and ExifTool is a library to use for metadata imputation, starting by copying all the Metadata from a high-resolution image from Bebop 2 Figure 30a into all the extracted frames. Then delete all GPS Tags from the frames and write each GeoTag collected during the flight to its correspondent frame.

## 5.5 Image Homography

Assuming a pinhole camera model, any two images of the same planar surface in space have a homography relation. Moreover, there is an estimated homography matrix, and this information may be used for navigation or to adjust the photo at 83 degrees (maximum Bebop 2 pitch angle camera) to a photo at 90 degrees with the correct perspective and appear as if initially taken at bird's-eye view Figure 33.

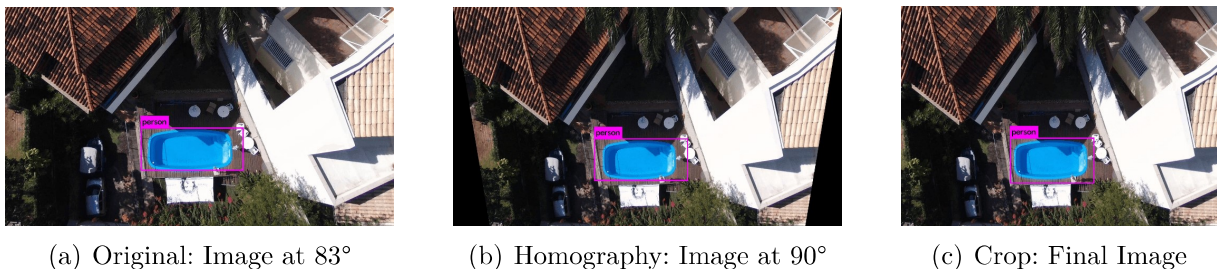


Figure 33 – Applying Homography on the original image for a bird's-eye view

Google Colab Notebook *Bebop2\_Homography.ipynb*, source code on Github<sup>1</sup>

### 5.5.1 Calculating Bebop Homography Parameters

A camera bird's-eye view, seen from a geometric perspective, forms different shapes from the camera lens to the observed surface. It depends mainly on the alignment angle and the camera's internal projection shape to form the image. For fisheye lens image is circular, the volume observed is a cone, while the standard camera image is rectangular and the

<sup>1</sup> <<https://gist.github.com/anthonymiglio/2900af4b1e3678f4eb5cc824f701859f>>

volume observed is a pyramid, and wide-angle lenses form images and volumes between the two types as mentioned earlier lenses. The Bebop 2 MAV has a fisheye lens, register still images as fisheye, but for video recording, the drone has an FPGA (Field-Programmable Gate Array) that undistort and crops the digital image in real-time into a rectangular shape according to the CMOS or CCD, registering a Full-HD resolution video at 30 FPS.

There are different types of pyramids. However, the Bebop 2 camera at bird's-eye view at the maximum pitch angle of  $-83$  degrees creates an FoV of an Oblique Rectangular Pyramid (Figure 34)<sup>2</sup> to avoid distortions during orthophoto creation; must apply a homography procedure to each captured image for a pitch angle of  $-90$  degrees.

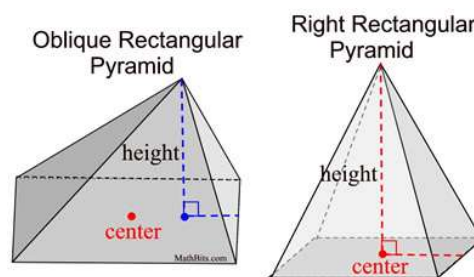
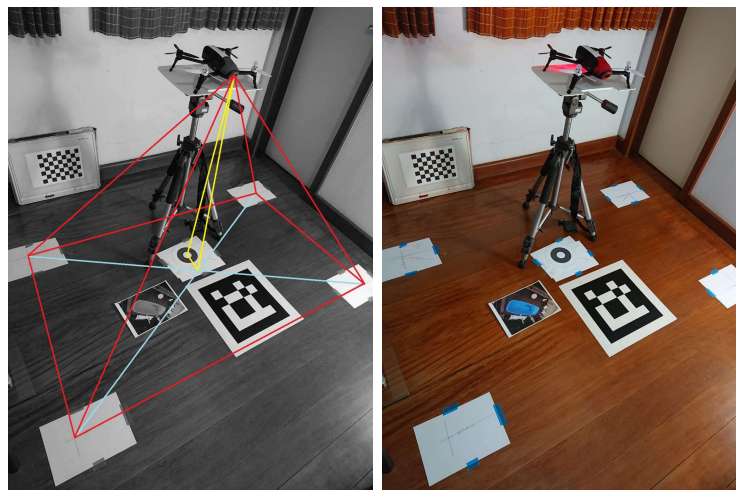


Figure 34 – Rectangular Pyramid: Oblique and Right

According to Perspective Projection on a pyramid, if the measurement of the image's aspect on the ground at 1,00-meter height Figure 35, there is a ratio directly proportional to the other desired height, in this case, the UAV surveillance flying height from 20,00 to 30,00 meters.



(a) Bebop 2 FoV Visualization (b) Bebop 2 Camera Pitch Angle at  $83^\circ$

Figure 35 – Bebop 2 Camera FoV: Oblique Rectangular Pyramid  
Photographer: Anthony Oliveira Pinto

<sup>2</sup> <<https://mathbitsnotebook.com/Geometry/3DShapes/pyramidoblique1.jpg>>

To measure the FoV (Field of View) of the Bebop 2 Camera in the video aspect: drone placed on a flat base set at the top of a photography tripod Figure 35 at the height of 1,00-meter from the center of the camera lens to the Whycon marker Chapter 2.3.6 on the ground as reference for the drone visual 90 degrees angle projection. Four A4 pieces of paper taped on the ground passing the captured image limits on the ground. Viewing the Bebop 2 camera streaming video and by hand with a pencil, the boundaries marked on the A4 papers the linear distances measured with a tape measure and angle with a protractor. The aspect ratio (with the camera at 1,00-meter height) from the bottom border to the top border is 1.532 / 1.692. In other words, the quadrilateral projection on the ground has a bottom border 90.53% the size of the top border of the original image.

## 5.6 Overlap

Overlap is an essential concept that must be used in order to make an Orthophoto. It consists on how much of a photo is part of the adjacent photo or adjacent Lidar beam (ALSADIK; REMONDINO, 2020), usually presented in percentage and it can be of two natures forward overlap and lateral overlap Figure 36<sup>3</sup>. The higher the overlap, more image information to process by the orthophoto software, but with a greater accuracy as the final result, on the other hand a smaller overlap, a smaller accuracy on the final orthophoto. There is a balance between forward and lateral overlap, and according to ODM (Open Drone Map) chapter 2.3.9 forward overlap 0.80 and lateral overlap 0.65. See, Experiment 3 on Chapter 6.3.

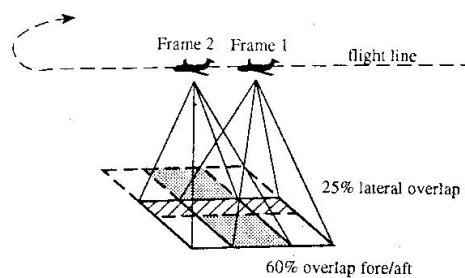


Figure 36 – Image Overlap

## 5.7 Orthophoto

An orthophoto is photos in orthogonal projection Figure 37<sup>4</sup>. It is a photographic representation of a region of the earth's surface in aerial photography. All elements present the same scale, free from errors and deformations with the same validity as a cartographic plan.

<sup>3</sup> <<https://www.fao.org/3/W0615E/W0615E02.HTM>>

<sup>4</sup> <<https://en.wikipedia.org/wiki/Orthophoto>>

To create an Orthophoto, a set of aerial images Figure 45<sup>5</sup> (taken from an airplane, UAV, or satellite) digitally corrected to represent an orthogonal projection without perspective effects. It is possible to perform exact measurements, unlike an aerial photograph, which always has deformations caused by the perspective of the camera, the altitude, or the speed at which the camera is moving. This digital correction process is called orthorectification. Google Maps approximates an orthophoto taken from its satellites, but it is good to estimate approximate distances between points.

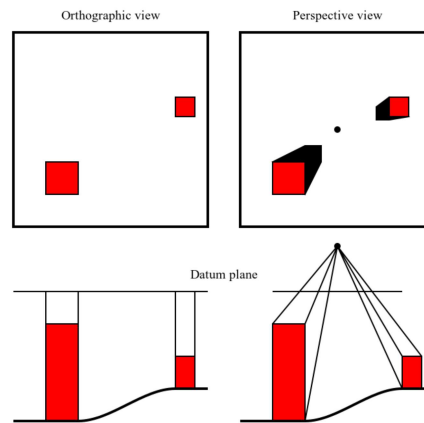


Figure 37 – Orthographic view vs Perspective view

## 5.8 GSD

The Ground Sampling Distance (GSD) is a metric that measures the distance between two consecutive pixel centers measured on the ground Figure 38. The bigger the value of the image GSD, the lower the spatial resolution of the image and the less visible details.

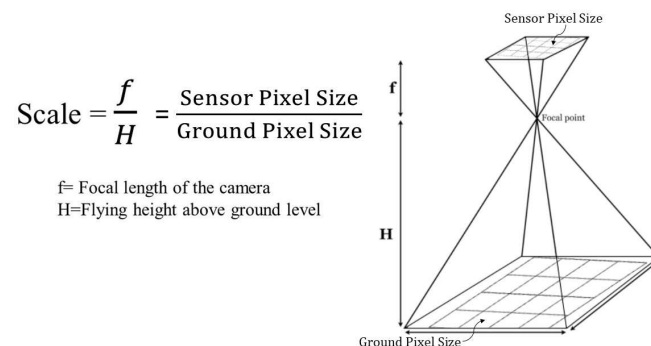


Figure 38 – Perspective Projection

<sup>5</sup> <<https://en.wikipedia.org/wiki/Orthophoto>>



## 6 Experiments

### 6.1 Experiment 1: Drone Flight with Yolo over a house pool

In this first experiment, the Bebop 2 fly manually by remote control on a natural urban outdoor environment to test the drone dynamics and the Yolo custom UAV Dataset in action searching for classified objects, a swimming pool<sup>1</sup>. The mission consisted of taking off from the parking place, raising to 20 meters height looking around in person point-of-view. Then look down, switching to Bird-Eye-view, searching for the swimming pool and displacing over it. While raising the drone, Figure 39 has some False positives, and these do not come into account because the UAV Dataset is for Bird-Eye-view only.



Figure 39 – UAV Dataset False Positives in Person Point-of-view

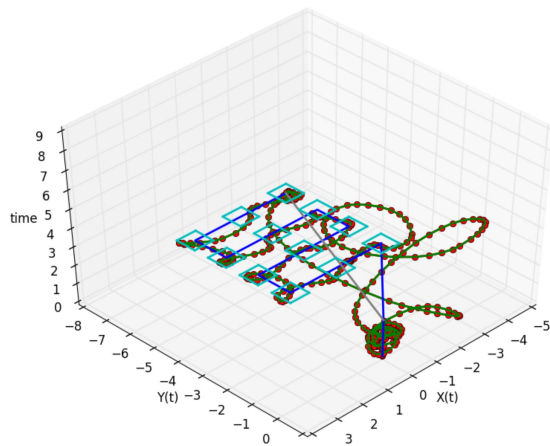
Once it switched to bird's-eye view, the detection of both False Positives as Silver Car Roof Tops and True Positives Swimming Pool came out on screen in Fig.40a and Fig.40b. However, in every image frame, the probability of the Swimming pool object was higher than the Car rooftops. In contrast, only the True Positive was object detected in many frames, showing the consistency of the trained model with the swimming pool classifier and a need for reinforcement training for the car rooftop classifier due to similarities in color and shape.



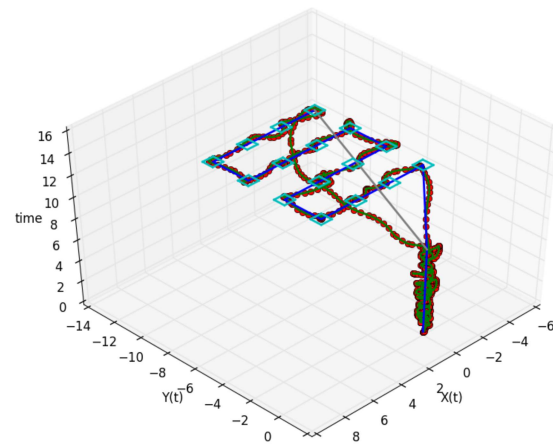
Figure 40 – UAV Dataset True and False Positives in bird's-eye view

<sup>1</sup> Youtube video: <[https://www.youtube.com/watch?v=fKb\\_Imok1gw](https://www.youtube.com/watch?v=fKb_Imok1gw)>

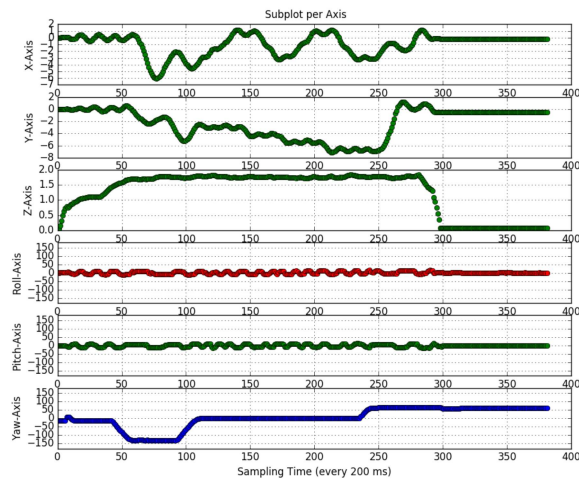
## 6.2 Experiment 2: UAV Flight Yaw Orientation vs Fix Orientation



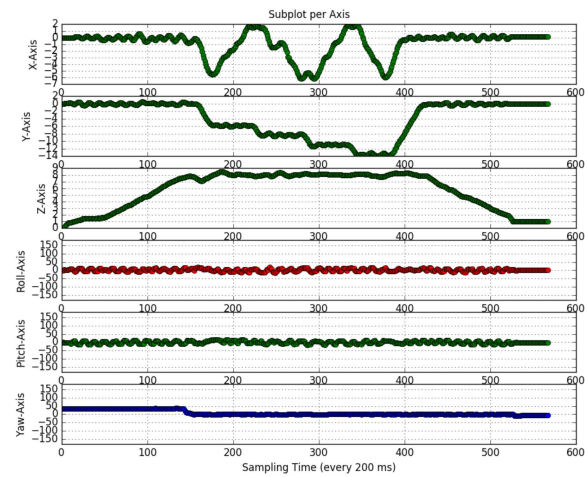
(a) 3D Graph Variable Yaw Orientation



(b) 3D Graph Fix Yaw Orientation



(c) Photo



(d) Photo

Figure 41 – UAV Flight Yaw Orientation Issue

As presented in chapter 2.1.1.1, the Bebop 2 Modeling work at  $\pm 5^\circ$ . Furthermore, the Roll and Pitch movements increased to  $\pm 10^\circ$  to overcome the outdoor winds. This change influenced the YAW movement Figure 41a, by introducing a position error into the x-axis and y-axis while the UAV changes its orientation in YAW Figure 41c from Sampling Time 40 to 110. Once established the PSI angle, the drone corrects itself and carries on with the path-planning task step-by-step up to a point where it finished the creeping line and orientates PSI towards home, missing the landing spot and regaining position control once it stops YAW rotation.

Even though the linear axes stability, the Yaw axial movement suffers from such model change from 5 to 10-degree limitation. Therefore, a fixed North Orientation instead of a Yaw movement bypass this issue, as show on Figure 41b and 41d.

### 6.3 Experiment 3: UAV Flight Different Overlap at same height

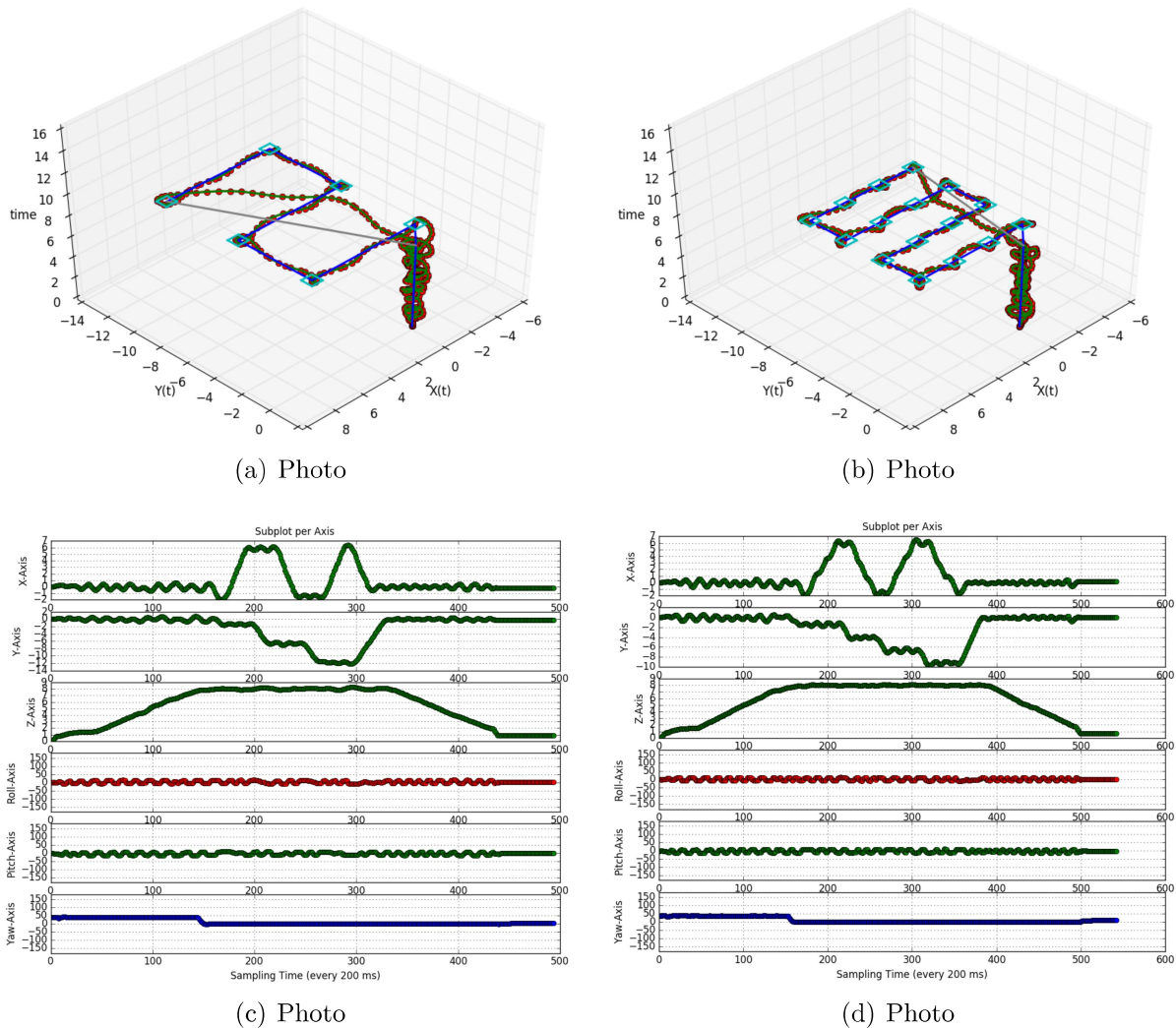


Figure 42 – UAV Flight Different Overlap at same height

As presented in chapter 5.6, the path planning will have a different outcome Figure 42c and Figure 42d for different overlaps. The closer the overlap is to 0%, the shorter the overlap, the greater the distance between photos on the path planning. On the other hand, the closer the overlap is to 100%, the greater the overlap, the shorter the distance between photos on the path planning. It is possible to observe this by comparing Figure 42a at overlap forward 40% and lateral 30% has six photos desired positions to cover the same area as the Figure 42b at overlap forward 80% and lateral 65% has 16 photos desired positions. This number of photos will directly affect the orthophoto resolution. The standard for the ODM app is overlapped 80% and 65% for a high-quality orthophoto.

## 6.4 Experiment 4: UAV Flight at 20 meters height over Parking lot

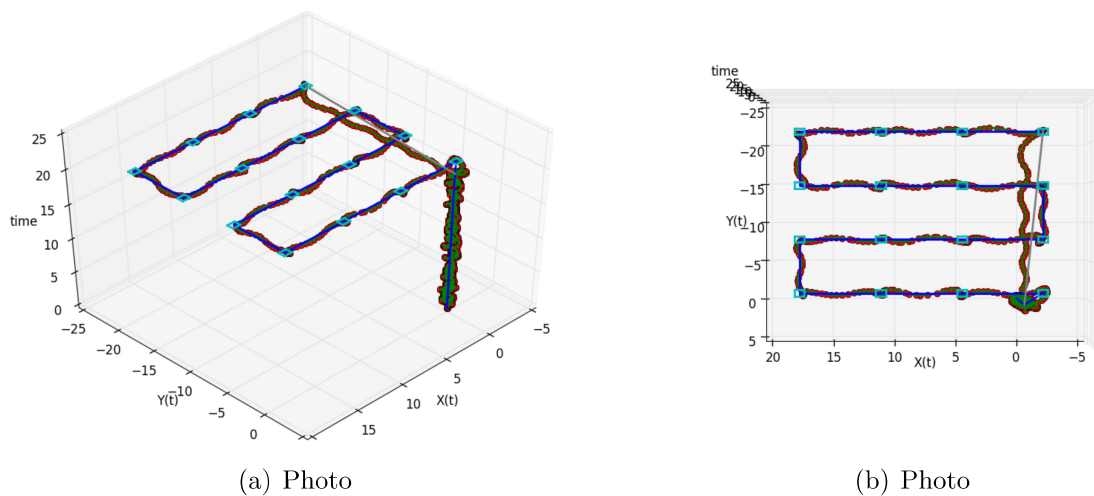


Figure 43 – UAV Flight at 20 meters height over Parking lot

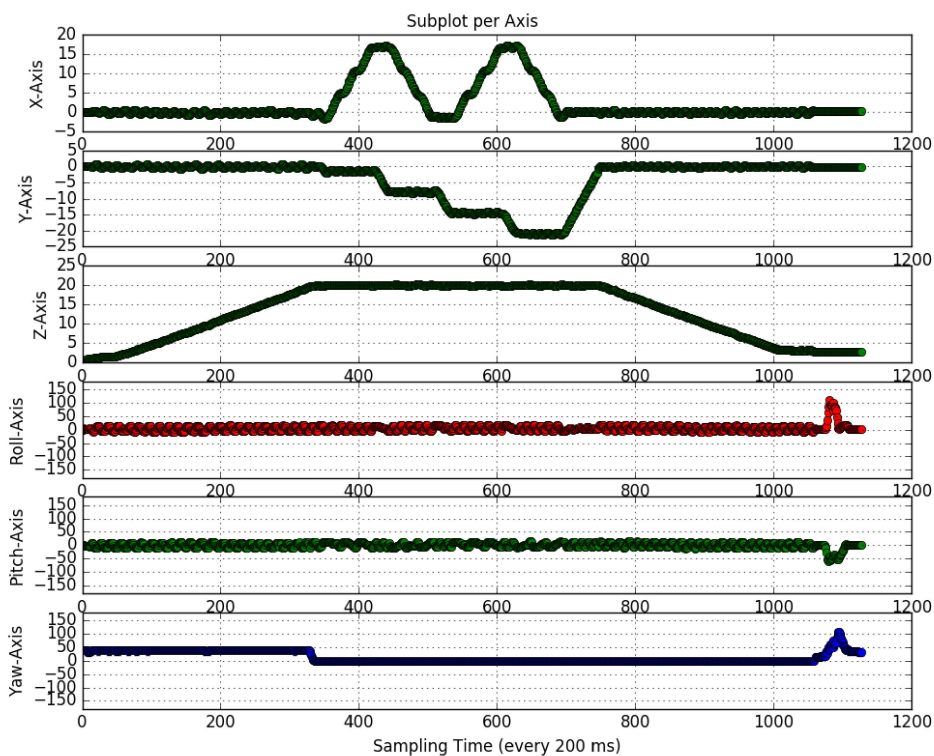


Figure 44 – 3D Overview of Flight at 20 meters

This experiment runs at a higher surveillance level, with longer distances for a longer duration. By comparing Figure 43 with the previous two experiments, the first noticeable trait is that the delimitation square for the photo desired position has decreased in proportion to the size of the path planning. The delimitation square is a fixed  $0.5 \times 0.5 \times 0.5$  m cube that determines this space as the boundaries of the photo's desired position equal to

the desired position when the UAV enters this air space. For longer flight dimensions, the UAV flies smoother Figure 44, and this is observed in-camera with more stable images as Figure 45 shown in the order taken. At the end of the flight, there are picks on the UAVs angular positions because the accumulative odometry error in the Z-axis increased due to a more extended flight, and the vehicle calculated the ground lower than it was, but the drone altitude sensor and the internal controller kept from colliding to the ground until it land.



Figure 45 – Search Pattern Photos Composition

Execute the steps on Chapter 5, then upload into the ODM app environment. It asks to create a new project, upload the images, select the preferences, but for orthophoto and point cloud, keep it on default, press enter, and way for some time that will depend basically on the number of images and their resolution. The software has been an open-source project in development for the last seven years.

## 6.4.1 Experiment 4: ODM Image Results: 2D Orthophoto and 3D Point Cloud

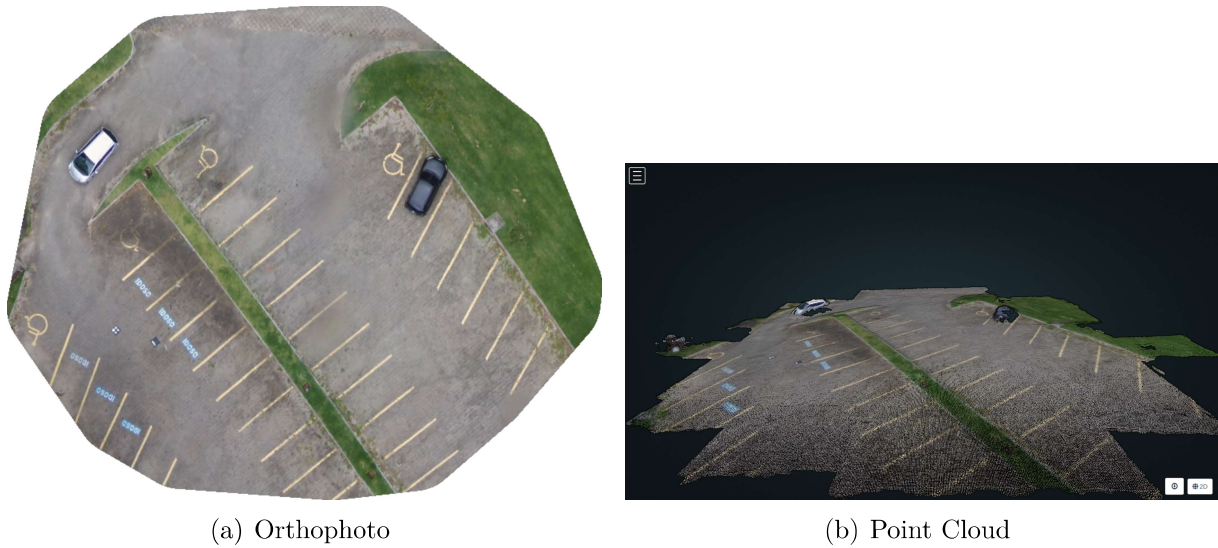


Figure 46 – ODM Image Results: 2D Orthophoto and 3D Point Cloud

Table 8 – ODM Quality Report: Dataset and Processing Summary

Date	29/12/2021 at 15:35:42
Area Covered	0.000398 km <sup>2</sup>
Processing Time	13.0m:44.0s
Capture Start	28/11/2021 at 15:03:56
Capture End	28/11/2021 at 15:07:38
Reconstructed Images	16 over 16 shots (100.0%)
Reconstructed Points (Sparse)	19899 over 19941 points (99.8%)
Reconstructed Points (Dense)	230,816 points
Average Ground Sampling Distance (GSD)	1.7 cm
Detected Features	8,773 features
Reconstructed Features	3,764 features
Geographic Reference	GPS
GPS errors	0.86 meters

This subsection presents the result of ODM with a orthophoto Figure 46a, a point cloud 46b and a quality report on 8, that show the processing time of over 13 minutes for these 16 photo.

## 6.4.2 Experiment 4: Georeference Objects in Orthophoto



Figure 47 – ODM Orthophoto Georeference

Table 9 – Georeferencing Objects on image:

Object:	Silver Car	Black Car
Bounding Box Type:	DOTA	YOLO
Perimeter:	10.82 m	12.98 m
Area:	6.56 m <sup>2</sup>	10.32 m <sup>2</sup>
Volume:	2.55 m <sup>3</sup>	4.34 m <sup>3</sup>
GPS (latitude, longitude):	(-20.275017, -40.302640), (-20.275028, -40.302628), (-20.275052, -40.302651), (-20.275040, -40.302664)	(-20.275037, -40.302458), (-20.275071, -40.302459), (-20.275070, -40.302432), (-20.275037, -40.302432)

Based on each image's GPS information, latitude, longitude, and altitude, the ODM app process the orthophoto by comparing similarities and features on the image. It also statistically calculates the GPS coordinates on each image and inputs a calculated GPS coordinate on each pixel. This way, it is possible to use the measurement tool of the software to select the lines or bounding boxes for georeferencing of the selected objects. Figure 47 shows the false positives car rooftop georeferenced in two different techniques Table 9 for demonstration, DOTA (XIA et al., 2018) on the left and YOLO on the right.

## 7 Conclusion

The objective of this thesis was to show that it is possible to create a unique solution to deal with the problem at a custom level. The first step was to create a customized object detection system, using the YOLO v3 convolutional neural network as the tool to analyze images collected by a UAV that uses visual references as landmarks for navigation. The project contributes with a quick introduction to computer vision tasks, machine learning models, and deep learning algorithms in a single project. As a case study, it was considered to detect swimming polls in a certain neighborhood, searching for possible breeding grounds for the *Aedes Aegypti* mosquito, a great problem in the city of Vitória, ES, in particular in the Ilha do Boi neighborhood, for the diseases it causes. Such a system corresponds to the `Beboi_lai` ROS package, written in Python and Cpp, to control the UAV to autonomously fly over a given inspection area collecting aerial images. Such a package is available as an open-source code, with the side objective of encouraging other people to follow this same path, bringing computer vision into play in mobile robotics, especially regarding UAVs, thus showing the effectiveness of the initial proposition of this work.

In the sequel some possible directions to continue this work are given. They are:

1. To increase the database of captured images, focusing on improving the precision of the model, as well as to expand the number of classes the system is capable of identifying, thus reducing the number of false positives and allowing to search for other possible breeding grounds for the mosquito.
2. To redo the modeling of the Bebo 2 in an outdoor environment for making possible more aggressive attack angles in different wind conditions.
3. To use a different GPS, with higher precision and access to a more satellites, thus allowing increasing the sampling rate associated to the control loop from 5 Hz to 20Hz, to provide a smoother UAV motion control.
4. To design and assembly a proper outdoor UAV, able to transport a payload of at least 1kg, to make possible to install more hardware on board the UAV, such as a high-resolution camera with a Jetson Xavier single-board computer, to allow processing the images on-board, for instance.
5. Other interesting hardware to install on board, in such a case, are two auxiliary stereo cameras, to make possible to implement SLAM and obstacle avoidance, for instance.
6. To implement a surveillance system using a multi UAV system, able to fly as a formation, implementing the necessary path-planning considering the multiple UAVs and how they are distributed.



# Bibliography

ALSADIK, B.; REMONDINO, F. Flight planning for lidar-based uas mapping applications. *ISPRS International Journal of Geo-Information*, v. 9, p. 378, 06 2020. Citado na página 56.

AMORIM, L. A.; VASSALLO, R. F.; SARCINELLI-FILHO, M. Building mosaics using images autonomously acquired by a uav. In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. [S.l.: s.n.], 2019. p. 306–312. Citado na página 20.

ARNOLD, R.; YAMAGUCHI, H.; TANAKA, T. Search and rescue with autonomous flying robots through behavior-based cooperative intelligence. *Journal of International Humanitarian Action*, v. 3, 12 2018. Citado na página 39.

BJELONIC, M. *YOLO ROS: Real-Time Object Detection for ROS*. 2016–2018. <[https://github.com/leggedrobotics/darknet\\_ros](https://github.com/leggedrobotics/darknet_ros)>. Citado na página 32.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. <https://arxiv.org/abs/2004.10934>. Citado na página 45.

CAI, Z.; VASCONCELOS, N. Cascade r-cnn: High quality object detection and instance segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 43, n. 5, p. 1483–1498, 2021. Citado na página 45.

CHEN, S.; LAEFER, D. F.; MANGINA, E. State of technology review of civilian uavs. *Recent Patents on Engineering*, v. 10, n. 3, p. 160–174, 2016. Citado na página 20.

CHENG, G. et al. Learning rotation-invariant and fisher discriminative convolutional neural networks for object detection. *IEEE Transactions on Image Processing*, v. 28, n. 1, p. 265–278, 2019. Citado na página 49.

COGLIATTI-CARVALHO, L. et al. Water volume stored in bromeliad tanks in brazilian restinga habitats. *Acta Botanica Brasilica*, v. 24, p. 84–95, March 2010. Citado na página 19.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*. San Diego, CA, USA: IEEE Computer Society, 2005. (CVPR '05), p. 886–893. ISBN 0769523722. Citado na página 45.

EVERINGHAM, M. et al. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, v. 111, p. 98–136, 2014. Citado na página 48.

GAUEN, K. et al. Comparison of visual datasets for machine learning. In: *2017 IEEE International Conference on Information Reuse and Integration (IRI)*. San Diego, CA, USA: [s.n.], 2017. p. 346–355. Citado na página 48.

GIRSHICK, R. Fast r-cnn. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: [s.n.], 2015. p. 1440–1448. Citado na página 45.

- GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2014. v. 1, p. 580–587. ISSN 1063-6919. Citado na página 45.
- GREGOIRE, M. D: Introduction to uml. In: \_\_\_\_\_. [S.l.: s.n.], 2018. p. 1083–1086. ISBN 9781119421306. Citado na página 33.
- HARIKUMAR, K.; SENTHILNATH, J.; SUNDARAM, S. Multi-uav oxyrrhis marina-inspired search and dynamic formation control for forest firefighting. *IEEE Transactions on Automation Science and Engineering*, v. 16, n. 2, p. 863–873, 2019. Citado na página 20.
- HUNT, D. Gang of four design patterns. In: \_\_\_\_\_. [S.l.: s.n.], 2013. p. 135–136. ISBN 978-3-319-02191-1. Citado na página 33.
- JIAO, Z. et al. A yolov3-based learning strategy for real-time uav-based forest fire detection. In: *2020 Chinese Control And Decision Conference (CCDC)*. [S.l.: s.n.], 2020. p. 4963–4967. Citado na página 21.
- KARAMANOU, A. et al. Supporting search and rescue operations with uavs. In: . [S.l.: s.n.], 2018. Citado na página 39.
- LENZ, M. *Drone Swarms for Firefighting: the Future of Fire Supression?* April 28, 2021. DroneLife Blog. Accessed: January 9, 2022. Disponível em: <<https://dronelife.com/2021/04/28/drone-swarms-for-firefighting-the-future-of-fire-supression/>>. Citado na página 20.
- Li, D. et al. A richly annotated pedestrian dataset for person retrieval in real surveillance scenarios. *IEEE Transactions on Image Processing*, v. 28, n. 4, p. 1575–1590, 2019. Citado na página 47.
- LIN, T.-Y. et al. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 42, n. 2, p. 318–327, 2020. Citado na página 45.
- LIN, T.-Y. et al. Microsoft coco: Common objects in context. In: *3th European Conference on Computer Vision (ECCV)*. Zurich, Switzerland: [s.n.], 2014. p. 740–755. Citado na página 48.
- LINDEBERG, T. Scale Invariant Feature Transform. *Scholarpedia*, v. 7, n. 5, p. 10491, 2012. Citado na página 45.
- LIU, W. et al. Ssd: Single shot multibox detector. In: LEIBE, B. et al. (Ed.). *The 14th European Conference on Computer Vision (ECCV 2016)*. Amsterdam, The Netherlands: Springer International Publishing, 2016. p. 21–37. ISBN 978-3-319-46448-0. Citado na página 45.
- MALTA, V. et al. Criadouros de aedes (stegomyia) aegypti (linnaeus, 1762) em bromélias nativas na cidade de Vitória, ES. *Revista da Sociedade Brasileira de Medicina Tropical*, v. 38, 05 2005. Citado na página 19.

- MERINO, L.; DIOS, J. R. Martínez-de; OLLERO, A. Cooperative unmanned aerial systems for fire detection, monitoring, and extinguishing. In: \_\_\_\_\_. *Handbook of Unmanned Aerial Vehicles*. Dordrecht: Springer Netherlands, 2015. p. 2693–2722. ISBN 978-90-481-9707-1. Citado na página 20.
- NITSCHKE, M. et al. Whycon: An efficient, marker-based localization system. In: *IROS Workshop on Open Source Aerial Robotics*. [S.l.: s.n.], 2015. Citado na página 33.
- PANG, J. et al. Libra r-cnn: Towards balanced learning for object detection. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE Computer Society, 2019. v. 1, p. 821–830. Citado na página 45.
- PINTO, A. O.; CIARELLI, P. M.; SARCINELLI-FILHO, M. Creating a yolov3 custom model for aerial surveillance. In: *2021 XIX Workshop on Information Processing and Control (RPIC)*. [S.l.: s.n.], 2021. p. 1–6. Citado na página 21.
- PINTO, A. O. et al. High-level modeling and control of the bebop 2 micro aerial vehicle. In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. Athens, Greece: [s.n.], 2020. p. 939–947. Citado 6 vezes nas páginas 19, 21, 23, 24, 25, and 43.
- POLKA, M.; PTAK, S.; KUZIORA, L. The use of uav's for search and rescue operations. *Procedia Engineering*, v. 192, p. 748–752, 12 2017. Citado na página 39.
- QU, H. et al. A pedestrian detection method based on yolov3 model and image enhanced by retinex. In: *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. [S.l.: s.n.], 2018. p. 1–5. Citado na página 21.
- RADOGLUO-GRAMMATIKIS, P. et al. A compilation of uav applications for precision agriculture. *Computer Networks*, v. 172, p. 107148, 2020. ISSN 1389-1286. Citado na página 20.
- REDMON, J. et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. <https://arxiv.org/abs/1506.02640>. Citado 4 vezes nas páginas 13, 20, 45, and 46.
- REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, USA: [s.n.], 2017. p. 6517–6525. Citado na página 45.
- REDMON, J.; FARHADI, A. *YOLOv3: An Incremental Improvement*. 2018. <https://arxiv.org/abs/1804.02767>. Citado 3 vezes nas páginas 20, 32, and 45.
- REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*. Montreal, Canada: [s.n.], 2015. p. 91–99. Citado na página 45.
- RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, v. 115, p. 211–252, September 2014. Citado na página 48.
- SANTANA, L. V. *Sistemas de Navegação e Controle para Veículos Aéreos Não Tripulados e suas Aplicações*. Tese (Doutorado) — Universidade Federal do Espírito Santo, 2016. Citado na página 25.

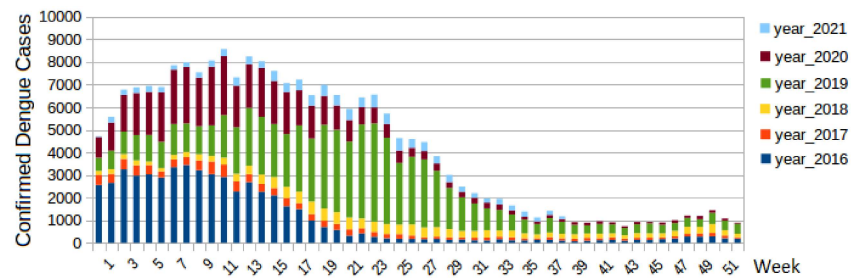
- SANTANA, L. V.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. Heterogeneous leader-follower formation based on kinematic models. In: IEEE. *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*. Arlington, VA, USA, 2016. p. 342–346. Citado na página 25.
- SANTANA, L. V. et al. A trajectory tracking and 3d positioning controller for the ar.drone quadrotor. In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. Orlando, FL, USA: [s.n.], 2014. p. 756–767. Citado 2 vezes nas páginas 25 and 43.
- SANTANA, L. V.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. Navigation and cooperative control using the ar.drone quadrotor. *Journal of Intelligent & Robotic Systems*, v. 84, n. 1, p. 327–350, 2016. Citado na página 24.
- SANTOS, M. C. et al. Indoor low-cost localization system for controlling aerial robots. *Control Engineering Practice*, v. 61, p. 93 – 111, 2017. ISSN 0967-0661. Citado na página 25.
- SANTOS, M. C. P. *Controle em Ambientes Interiores de Veículos Aéreos Não Tripulados*. Tese (Doutorado) — Universidade Federal do Espírito Santo, 2017. Citado na página 25.
- SANTOS, M. C. P. et al. An adaptive dynamic controller for quadrotor to perform trajectory tracking tasks. *Journal of Intelligent & Robotic Systems*, v. 93, n. 1, p. 5–16, February 2019. ISSN 1573-0409. Citado na página 25.
- SANTOS, M. C. P. et al. A novel null-space-based uav trajectory tracking controller with collision avoidance. *IEEE/ASME Transactions on Mechatronics*, v. 22, n. 6, p. 2543–2553, Dec 2017. ISSN 1941-014X. Citado na página 25.
- SCHULDT, D.; KURUCAR, J. Efficient partitioning of space for multiple uas search in an unobstructed environment. *International Journal of Intelligent Robotics and Applications*, v. 2, 03 2018. Citado 2 vezes nas páginas 38 and 39.
- Secretaria de Saúde do Estado do Espírito Santo. *28<sup>o</sup> boletim de chikungunya*. 2021. <<https://mosquito.saude.es.gov.br/Noticias>>. Citado na página 18.
- Secretaria de Saúde do Estado do Espírito Santo. *Aedes Aegypti: Conheça algumas particularidades do mosquito Aedes Aegypti*. 2021. <<https://mosquito.saude.es.gov.br/Noticias>>. Citado na página 19.
- Secretaria de Saúde do Estado do Espírito Santo. *Aedes Aegypti: mosquito responsável pela transmissão da Dengue, Zika e Chikungunya*. 2021. <<https://mosquito.saude.es.gov.br/Noticias>>. Citado 2 vezes nas páginas 18 and 19.
- SINHA, G. R. *Computer Vision versus Human Vision*. 2020. ACM Distinguished Speakers Program. Citado na página 44.
- STADNIK, A. V.; SAZHIN, P. S.; HNATIC, S. Comparative performance analysis of neural network real-time object detections in different implementations. *EPJ Web of Conferences*, v. 226, p. 02020, 2020. Citado na página 45.
- SUN, P.; PIAO, J.-C.; CUI, X. Object detection in urban aerial image based on advanced yolo v3 algorithm. In: *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*. [S.l.: s.n.], 2020. p. 2191–2196. Citado na página 21.

- TAN, Y. et al. Automatic inspection data collection of building surface based on bim and uav. *Automation in Construction*, v. 131, p. 103881, 2021. ISSN 0926-5805. Citado na página 20.
- TOEVIS, B. Processing of metadata on multimedia using exiftool: A programming approach in python. In: . [S.l.: s.n.], 2015. p. 26–30. Citado na página 34.
- VALENÇA, M. et al. Dynamics and characterization of aedes aegypti (l.) (diptera: Culicidae) key breeding sites. *Neotropical entomology*, v. 42, p. 311–6, 06 2013. Citado na página 19.
- VILLA, D. K. D.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. A survey on load transportation using multirotor UAVs. *Journal of Intelligent & Robotic Systems*, v. 98, n. 2, p. 267–296, 2020. Citado na página 20.
- WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2021. <https://arxiv.org/abs/2011.08036>. Citado na página 45.
- XIA, G. et al. Dota: A large-scale dataset for object detection in aerial images. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, USA: IEEE Computer Society, 2018. p. 3974–3983. Citado 2 vezes nas páginas 47 and 64.
- XIAO, J. et al. Sun database: Large-scale scene recognition from abbey to zoo. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2010. p. 3485–3492. Citado na página 48.
- ZHANG, B. Computer vision vs. human vision. In: *9th IEEE International Conference on Cognitive Informatics (ICCI'10)*. Beijing, China: [s.n.], 2010. p. 3–3. Citado na página 44.
- ZHANG, C.; KOVACS, J. M. The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, Springer, v. 13, n. 6, p. 693–712, 2012. Citado na página 20.
- ZHANG, S. et al. Single-shot refinement neural network for object detection. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT, USA: [s.n.], 2018. p. 4203–4212. Citado na página 45.
- ZHU, X. et al. Deformable convnets v2: More deformable, better results. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: [s.n.], 2019. p. 9300–9308. Citado na página 45.

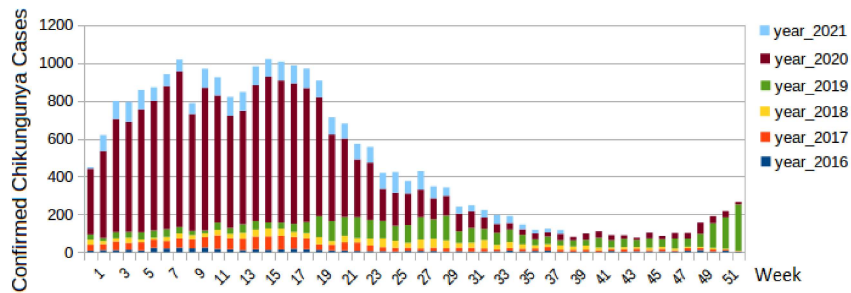
# Appendix

# APPENDIX A – Cases from 2016 to 2021 in the State of Espírito Santo

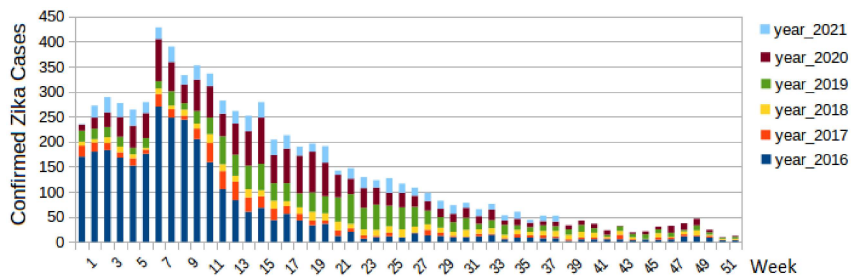
## A.1 Accumulative Cases: Dengue, Chikungunya, Zika



(a) Dengue



(b) Chikungunya



(c) Zika

Figure 48 – Histograms: Confirmed Cases per Week from 2016 to 2021

# APPENDIX B – *ROS essentials*

## B.1 Install ROS Kinetic Kame

As discussed, ROS is not an operating system, but it needs a host operating system to work. Ubuntu Linux is the most preferred OS for installing ROS.

0. Go to <<http://wiki.ros.org/kinetic/Installation/Ubuntu>>

1. Open Terminal and follow installation steps

1.1 Setup your computer to accept software from packages.ros.org:

```
1 sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/
2 apt/sources.list.d/ros-latest.list'
```

1.2 Set up your keys

```
1 sudo apt install curl
2 curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add
   -
```

1.3 Update Debian package index:

```
1 sudo apt-get update
```

1.4. Run Desktop-Full Install:

```
1 sudo apt-get install ros-kinetic-desktop-full
```

1.5 Environment setup:

```
1 echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc source ~/.bashrc
```

1.6 To install tools and other dependencies for building ROS packages, run:

```
1 sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-
2 wstool build-essential
```

1.7 Install rosdep:

```
1 sudo apt install python-rosdep
```

1.8 Before using ROS tools, must initialize rosdep:

```
1 sudo rosdep init
2 rosdep update
```



## B.2 ROS Cheatsheet

Copyright © 2010 Willow Garage.

<<https://mirror.umd.edu/roswiki/attachments/de/ROSCheatsheet.pdf>>

Copyright © Clearpath Robotics Inc. All rights reserved.

<<https://clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/>>

## B.3 ROS OOP (Object Oriented Programming)

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import Int64
4 from std_srvs.srv import SetBool
5
6 class NumberCounter:
7
8     def __init__(self):
9         self.counter = 0
10        self.pub = rospy.Publisher("/num_count", Int64, queue_size=10)
11        rospy.Subscriber("/number", Int64, self.callback_number)
12        rospy.Service("/reset_counter", SetBool, self.callback_reset)
13
14    def callback_number(self, msg):
15        self.counter += msg.data
16        new_msg = Int64()
17        new_msg.data = self.counter
18        self.pub.publish(new_msg)
19
20    def callback_reset(self, req):
21        if req.data:
22            self.counter = 0
23            return True, "Counter has been successfully reset"
24        return False, "Counter has not been reset"
25
26 if __name__ == '__main__':
27     rospy.init_node('number_counter') # Initialize node
28     number_counter = NumberCounter() # Instantiate NumberCounter Class
29     rospy.spin()
```

Listing B.1 – OOP in rospy code example

```
1#include <ros/ros.h>
2#include <std_msgs/Int64.h>
3#include <std_srvs/SetBool.h>
4
5int counter = 0;
6
7class NumberCounter {
8    private:
9        ros::Subscriber sub;
10       ros::Publisher pub;
11       ros::ServiceServer svr; //reset_service
12
13       NumberCounter(ros::NodeHandle *nh) {
14           public:
15               sub = nh.subscribe("/number", 1000, CB_number);
16               pub = nh.advertise<std_msgs::Int64>("/number_count", 10);
17               svr = nh.advertiseService("/reset_counter",CB_resetCounter);
18           }
19       void CB_number(const std_msgs::Int64& msg){
20           counter += msg.data;
21           std_msgs::Int64 new_msg;
22           new_msg.data = counter;
23           pub.publish(new_msg);
24       }
25       bool CB_resetCounter(std_srvs::SetBool::Request &req,
26                           std_srvs::SetBool::Response &res){
27           if (req.data) {
28               counter = 0;
29               res.success = true;
30               res.message = "Counter has been reset";
31           }
32           else {
33               res.success = false;
34               res.message = "Counter has not been reset";
35           }
36           return true;
37       }
38}; // NumberCounter
39
40int main(int argc, char **argv){
41    ros::init(argc, argv, "number_counter");
42    ros::NodeHandle nh;
43    ros::Rate loop_rate(10);
44    NumberCounter number_counter = NumberCounter(&nh);
45    ros::spin();
46}
```

Listing B.2 – OOP in roscpp code example

## B.4 ROS Bebop Monocular Camera Calibration

To Calibrate a Monocular Camera:

Print a checkerboard with known dimensions and number of squares.

Have the monocular camera publishing images over ROS.

Compiling the driver and its dependencies.

```
rosdep install camera_calibration
```

List ROS topics to check that the images are published:

```
rostopic list
```

Check the topics are on: /camera/camera\_info and /camera/image\_raw.

Running the Calibration Node:

```
roslaunch camera_calibration cameracalibrator.py --size 8x6 --square 0.108 image:=/camera/  
image_raw camera:=/camera
```

In order to get a good calibration you will need to move the checkerboard around in the camera frame such that:

- checkerboard on the camera's left, right, top and bottom of field of view;
- X bar - left/right in field of view;
- Y bar - top/bottom in field of view;
- Size bar - toward/away and tilt from the camera;
- checkerboard filling the whole field of view;
- checkerboard tilted to the left, right, top and bottom (Skew).

At each step, hold the checkerboard still until the image is highlighted in the calibration window Figure 49.

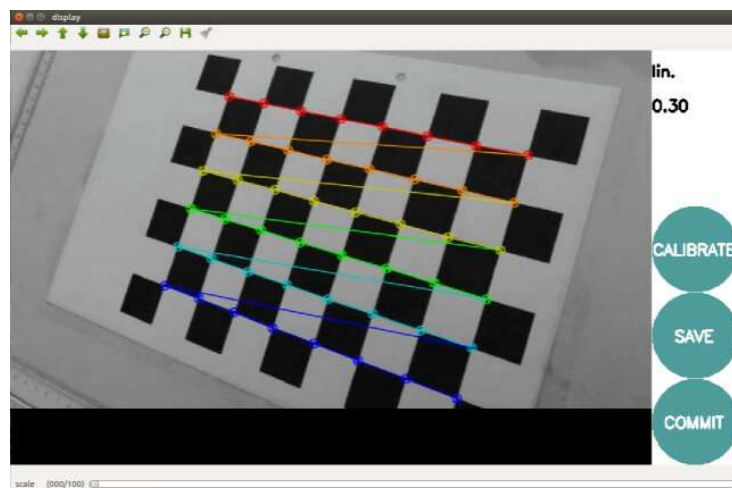


Figure 49 – ROS Camera Calibration

## Calibration Results:

```
1 # Bebop2 camera intrinsics
2 width 856
3 height 480
4
5 camera matrix
6 537.292878 0.000000 427.331854
7 0.000000 527.000348 240.226888
8 0.000000 0.000000 1.000000
9
10 distortion
11 0.004974 -0.000130 -0.001212 0.002192 0.000000
12
13 rectification
14 1.00000 0.00000 0.00000
15 0.00000 1.00000 0.00000
16 0.00000 0.00000 1.00000
17
18 projection\\
19 539.403503 0.000000 429.275072 0.000000
20 0.000000 529.838562 238.941372 0.000000
21 0.000000 0.000000 1.000000 0.000000
```

## Creating yaml file from Calibration result output for OpenCV standard:

```
1 image_width: 856
2 image_height: 480
3 camera_name: bebop_front
4 camera_matrix:
5   rows: 3
6   cols: 3
7   data: [537.292878, 0.000000, 427.331854, 0.000000, 527.000348, 240.226888, 0.000000,
8         0.000000, 1.000000]
9 distortion_model: plumb_bob
10 distortion_coefficients:
11   rows: 1
12   cols: 5
13   data: [0.004974, -0.000130, -0.001212, 0.002192, 0.000000]
14 rectification_matrix:
15   rows: 3
16   cols: 3
17   data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000,
18         1.000000]
19 projection_matrix:
20   rows: 3
21   cols: 4
22   data: [539.403503, 0.000000, 429.275072, 0.000000, 0.000000, 529.838562, 238.941372,
23         0.000000, 0.000000, 0.000000, 1.000000, 0.000000]
```

# APPENDIX C – *BebopClass*

## C.1 UML - *BebopClass*

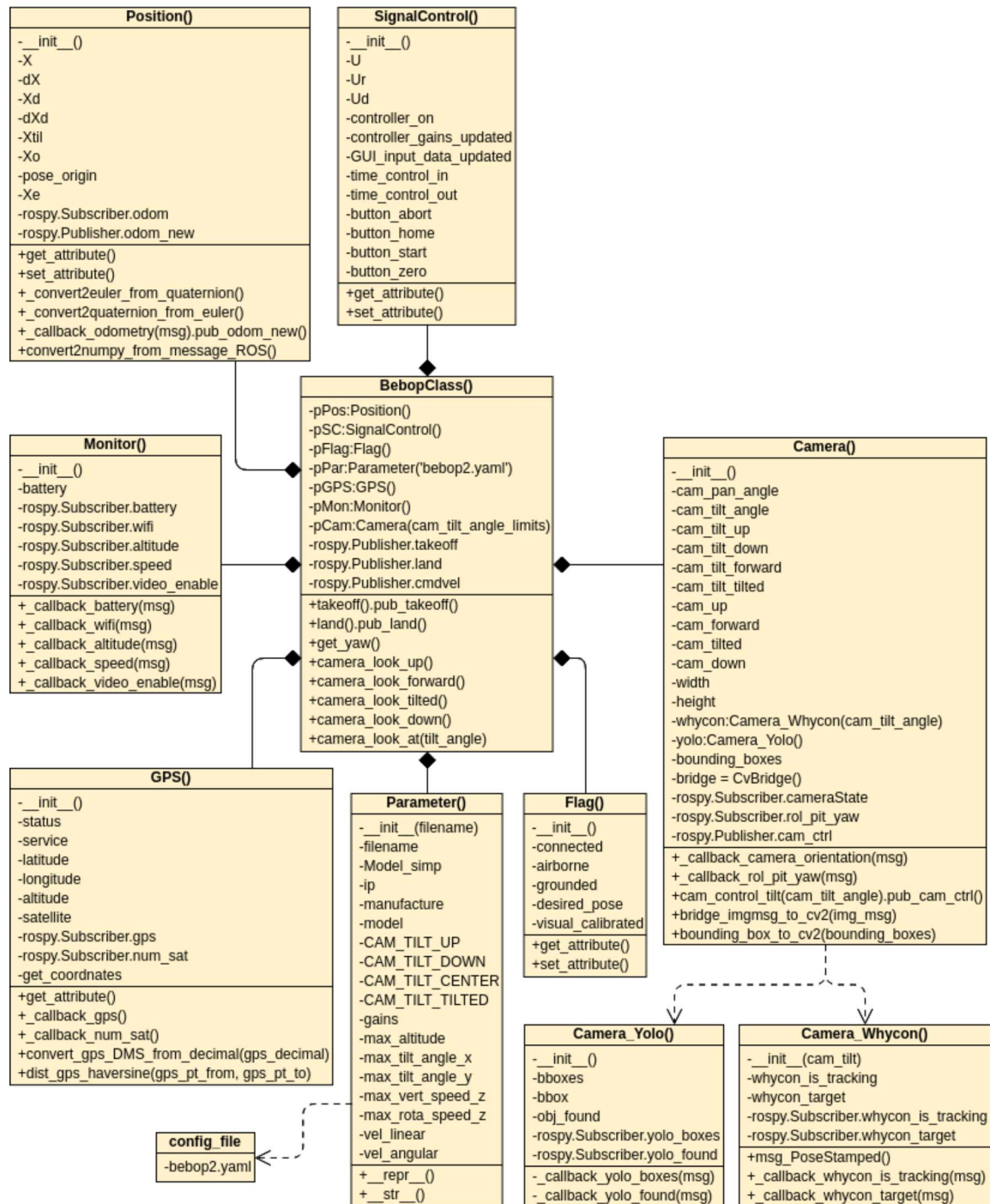


Figure 50 – UML: *BebopClass*

## APPENDIX D – *Conditional Tasking*

### *Recurrent Block Flow Chart per State*

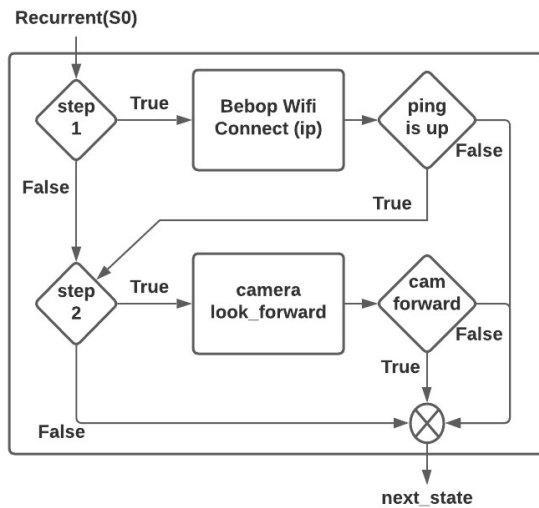


Figure 51 – Recurrent(S0) Block

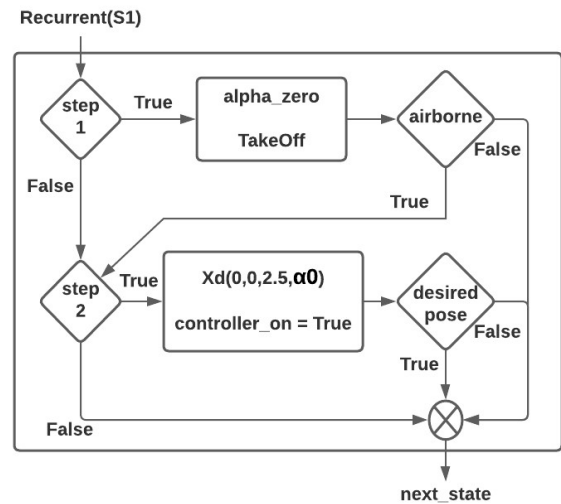


Figure 52 – Recurrent(S1) Block

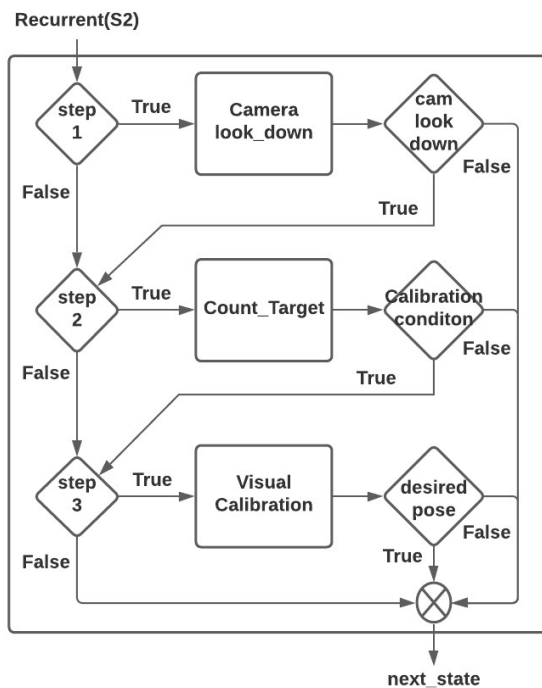


Figure 53 – Recurrent(S2) Block

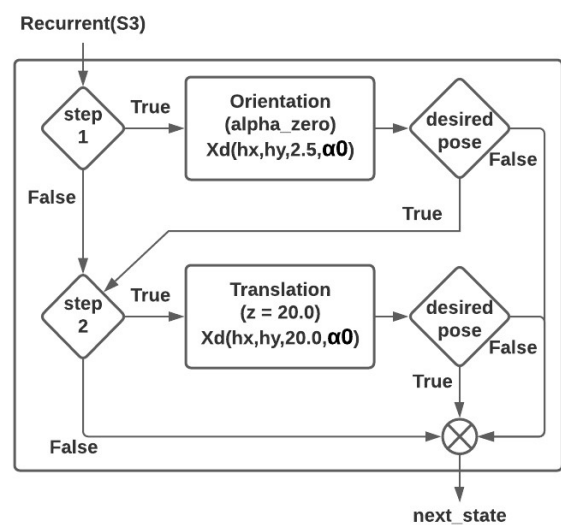


Figure 54 – Recurrent(S3) Block

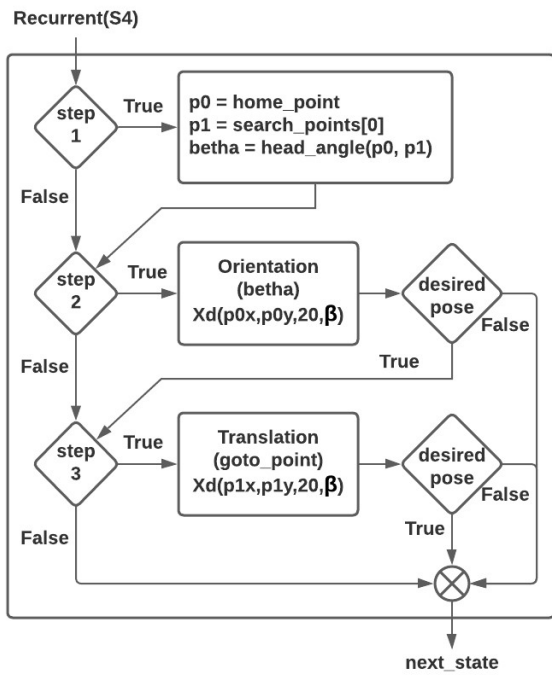


Figure 55 – Recurrent(S4) Block

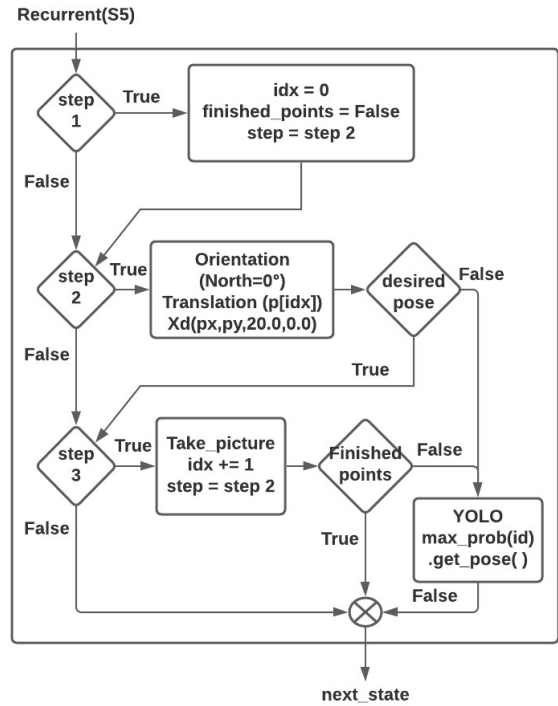


Figure 56 – Recurrent(S5) Block

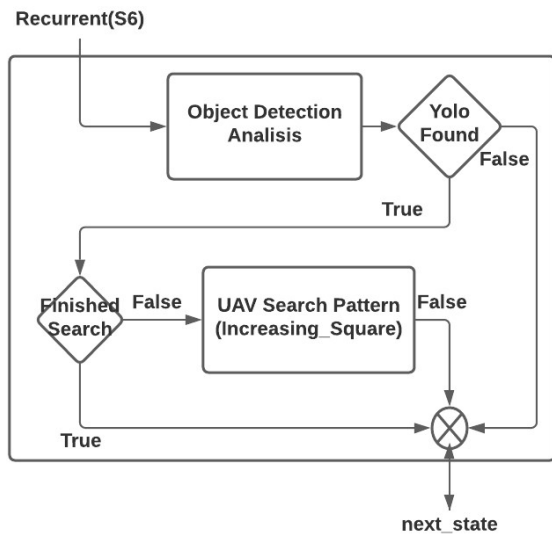


Figure 57 – Recurrent(S6) Block

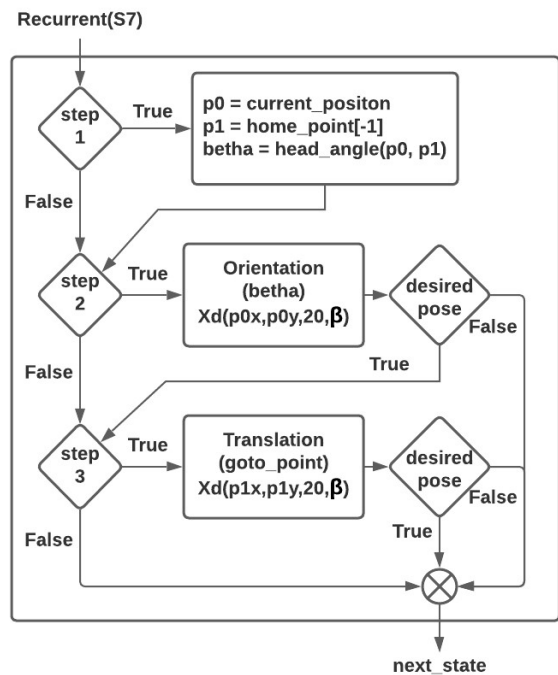


Figure 58 – Recurrent(S7) Block

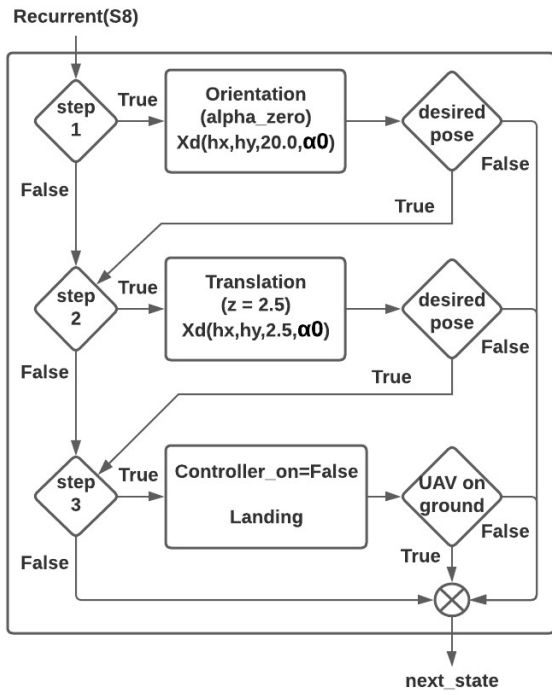


Figure 59 – Recurrent(S8) Block

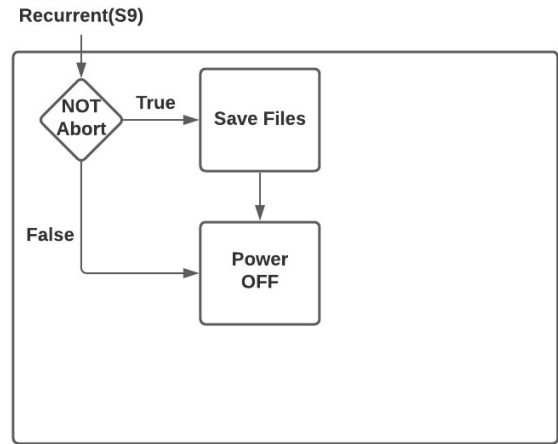


Figure 60 – Recurrent(S9) Block

*See You Space Cowboy...*

—"The Real Folk Blues" by the Seatbelts