A Small Autonomous UAV for Detection and Action in Precision Agriculture

by

Bilal Hazim Younus ALSALAM
B.Sc. in Electrical Engineering (Power & Machines)

Submitted in fulfilment of the requirements for the degree of

Master of Applied Science

School of Electrical Engineering and Computer Science
Queensland University of Technology

2017

Keywords

UAV, Remote Sensing, UAS, Remote Piloted Aircraft Systems, Low Altitude Remote Sensing, Weed Mapping, Weed Detection, Airborne Vision System, Vision Based Navigation, Aerial Robots, Guidance System.

Abstract

A phenomenal increase in the development of Unmanned Aerial Vehicles (UAVs) has been observed in a broad range of applications across various fields of study in recent years. Multirotor UAVs enable a broad range of applications due to their special flight characteristics such as vertical take-off and hovering over the target of interest to perform an action for different applications. Precision agriculture is one of the emerging fields of interest for UAV applications for monitoring crop health. UAVs can provide higher spatial resolution at lower operational costs and have the potential to facilitate site-specific invasive weed control treatments in crop fields, as opposed to manned aircraft or satellite remote sensing.

This thesis describes a framework for off-board and on-board systems which include the UAV platform, sensor payload and post-processing pipeline customised to detect an ArUco marker, a colour and ,specifically, invasive spear thistle weed. In the off-board system stage, the UAV collects the data and a computer vision algorithm checks whether spear thistle weed can be found in the image. The system creates GPS tabulated locations for weed detection. Results have shown that the sensitivity and selectivity of the algorithm depends upon both the flight height and the season of the weed. The sensitivity is the ability of the algorithm to identify and detect the true positive target while the selectivity is the capability of the algorithm to filter out the false negatives for detection targets. Furthermore, the system utilises false positive rates and false negative rates to achieve accurate results. False positives are when a classification indicates presence, however the weeds are absent, and false negatives are the point at which a classification demonstrates the weeds are absent, but are truly present. Results have shown a 95% sensitivity and 98% selectivity when the height above the ground is 5 m, 90% sensitivity and 94.5% selectivity when the height above the ground is 7 m and 80% sensitivity and 85% selectivity when the height above the ground is 15 m. The task was complex due to the difficulty in differentiating the spectral properties and general appearance of the highly correlated invasive spear thistle weeds and zoysia grass at two stages of growth and difficulties due to seasonal variability and sun angle.

After first applying the system for weed detection off-board, on-board decision making

was then implemented. A modular and generic system for the UAV using vision based navigation for on-board decision making with a focus on agriculture and remote sensing application was developed. A decision making approach similar to the Observation, Orientation, Decision and Action (OODA) loop was implemented. Position control based on the Robotic Operating System (ROS) was used to ensure the tracking of waypoints. The detection of an object of interest is facilitated by computer vision functionality. This allows the UAV to change its planned path accordingly in order to approach the target, apply pesticide or collect additional images at higher resolution. The results showed that the on-board system using the ROS operation system is capable of object detection and close the OODA loop framework. Results showed that the on-board system is capable of detecting an ArUco Marker with 99% sensitivity and 100% selectivity at a height of 5 metres above ground level. Furthermore, the on-board system is capable of detecting a red target with 96% sensitivity and 99% selectivity at the same height. The real time on-board detection and action algorithm for invasive weed needs to be improved to achieve better sensitivity and selectivity. The system is capable of detecting the invasive weed to 33% sensitivity and 67% selectivity. The low sensitivity and selectivity for weed detection is mainly due to limitations in the algorithm related to the sun angle and the season of the weed. This system has potential applications in the field of precision agriculture such as crop health monitoring and in particular, plant pest detection, which impacts on crop yields and results in financial losses if not noticed and addressed at an early stage.

Contents

List of Figures

List of Tables

List of Abbreviations

AC	Alternative Current
AF	Air Frame
APS	Autopilot System
ARCAA	Australian Research Centre for Aerospace Automation
BSD	Berkeley Software Distribution
DC	Direct Current
ESC	Electronic Speed Controller
GUI	Graphical User Interface
GCS	Ground Control Station
GPS	Global Positioning System
HLO	High Level Objective
HSV	Hue, Saturation and Value
I	Current
IDE	Integrated Development Environment
IPS	Image Processing System
OBIA	Object Based Image Analysis
OODA	Observation, Orientation, Decision, and Action
OpenCV	Open source Computer Vision
PA	Precision Agriculture
PPS	Power and Propulsion System
QUT	Queensland University of Technology
R	Resistance
RC	Remote Control
RGB	Red, Green and Blue
ROS	Robotic Operation System
TCS	Telecommunications System
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UBEC	Ultimate Battery Eliminator Circuit
USB	Universal Serial Bus
UTM	Universal Transverse Mercator
V	Voltage

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

QUT Verified Signature

Signed: __

3 March 2017

Date: ___ _

Acknowledgments

First, I would like to acknowledge the continual support from my supervisors and colleagues. I am grateful to Associate Professor Felipe Gonzalez and Professor Duncan Campbell for their invaluable insights, guidance and constructive feedback throughout this research. I would also like to acknowledge my colleague Kye Morton for his support during the system development and the flight test. Thank you also to all the students, research colleagues and administrative staff at their associated research centres including the Australian Research Centre for Aerospace Automation (ARCAA). In particular, I would like to acknowledge Dr Jonathan Kok and Dr Aaron Mcfadyen for their support during this research. I would like to extend a special thanks to pilot Steven Bulmer for his assistance during data collection and to Mr Michael Cahill the owner of the farm at Christmas Creek for his assistance while collecting the data and for allowing the flight test to occur on his farm. Thank you also to Chandrama Sarker for your support during the writing of this thesis.

Second, I would like to thank the Australian Research Centre for Aerospace Automation (ARCAA), Robotic and Autonomous System (RAS) and Queensland University of Technology (QUT) for the use of all the facilities and the research labs in which I conducted a significant portion of my research. I would also like to thank the Higher Committee for Educational Development in Iraq (HCED) for their financial support during my research.

Lastly, I would like to acknowledge my family and friends. To my father, my mother, brothers and sisters, I am grateful and thankful for all your support during my journey with this research. And to my friends, I am grateful for your support.

Bilal Hazim Younus ALSALAM

Queensland University of Technology

March 2017

CHAPTER 1

Introduction

1.1 Background and Motivation

Unmanned Aerial Vehicles (UAVs) are being used in several remote sensing applications including search and rescue, ecology, wildlife and precision agriculture [1, 2, 3]. Remote sensing in precision agriculture can assist farmers to assess plant yield, plant health and disease [4, 5]. Different types of sensors such as: thermal cameras, multispectral cameras, NIR cameras and digital cameras have been used on-board UAVs to collect data or monitor plant health [6, 7, 8, 9]. Computer vision and image processing have also been applied to remotely sensed images to make decisions in agricultural applications. These decisions could be carried out off-board after collecting and processing the data or UAV on-board while the UAV is flying [7, 6, 10]. There are cases where it is desirable to make an on-board decision to minimise the amount of data that is stored on-board. As an example, the UAV could be flying at a specific height, decide on a potential issue, descend and capture a high resolution image, or perform a closer inspection or apply pesticides or release insect bugs. The aim of this research is to develop and implement off-board and on-board systems for real time precision agriculture and plant pest detection.

Traditionally, farmers apply pesticides, herbicides or release insect bugs all over the field which is costly. The motivation for using UAVs with the structure of the OODA (Observation, Orientation, Decision and Action) loop framework is to detect invasive weed and implement an action such as applying herbicides. Autonomous on-board detection and action will target the specific location of the weed or pest for herbicide application instead of applying it on the whole field.

1.2 Research Objective

The aim of this research is to develop and flight test a multi-rotor airborne system which includes: UAV, microcomputer advanced processing interface with camera and GPS with OODA loop approach for off-board and on-board decision making. The system will be applied in the context of remote sensing in precision agriculture.

The main objectives for this research are as follows:

1.2.1 Objective 1

To investigate the use of optimisation techniques and develop algorithms which can detect invasive weeds, and to implement an algorithm for invasive weed detection and mapping for agricultural purposes. The invasive weed map will provide the geo-location of the detected weed which can give information to the farmer to assist them to apply the treatment or uproot the weed.

1.2.2 Objective 2

To apply an OODA loop for on-board decision making to detect a target of interest (a colour or specific type of weed, for example) and use the developed system to complete an action such as spraying or capturing high-resolution imagery from a lower height. To develop a position control based on navigation using the ROS (Robotic Operating System) to ensure a tracking of waypoints and the target location.

1.3 Research Problem and Research Plan

The use of UAVs for off-board and on-board decision making in precision agriculture is now an active and developed field of research. There are, however, limitations to supporting off-board and on-board decision making in the detection algorithm and theoretical framework. There are several different UAV platforms and computer vision algorithms that have been used and most of the research works focus on image analysis and classification executed after the UAV lands. There are, however, cases in which there is a need for rapid assessment with on-board image analysis and decision-making. The OODA loop method can assist in this effort, but it is limited by the resources and constraints of on-board computers that make the task more

challenging. This research experiments will answer the following questions:

1. What are the current limitations of image analysis, classification methods and vision based navigation with electro-optical sensors for on-board decision making?

2. What are the challenges in the practical applications of off-board and on-board systems in the context of precision agriculture and plant biosecurity where vegetation characteristics, such as texture, colour or shape, are posing significant challenges for existing image classification algorithms?

1.4 Research Contribution

The main outcomes and the contribution of this research is as follow:

- Application of computer vision to differentiate and classify a particular species of invasive weeds, i.e. spear thistle, amongst soil and zoysia grass and then provide GPS locations (geolocation) or weed map information that can assist farmers in determining weed locations and apply treatment.

- To develop an Unmanned Aerial System (UAS) capable of on-board target recognition for invasive weeds and autonomous decision making using PID based position control with the OODA loop concept. Such a system should be capable of deploying a UAV on a designated flight plan, detect an object of interest and then autonomously perform a manoeuvre based on its recognition of the target, such as descend to a lower flying height and spray the target or to take a high resolution image.

1.5 Research Methodology

The overall research framework can be divided into fives stages to allow for progressive development. The phases have been planned to span the suggested 1.5-2 years cycle.

1.5.1 Stage 1: Literature Review

The first stage mainly focused on a survey of relevant literature in the field of agriculture using UAVs, on-board decision making using the OODA loop method for remote sensing and identifying different computer vision techniques suitable for this research. The concept of an

OODA loop was also studied to use the method for on-board decision making after the target is detected.

1.5.2 Stage 2: Initial Data Collection and Algorithm Development for Off-board System

In this stage, an initial data collection campaign was conducted at Christmas Creek, Beaudesert in Queensland, Australia. This data was used to develop an algorithm for weed detection and mapping. After collecting the data, MATLAB was used to process the data and to develop a suitable weed detection and mapping method. The algorithm checked whether or not weed is in the image and create GPS tabulated locations of detected weed.

1.5.3 Stage 3: Software System Development for On-board System

This stage was divided into two parts:

A- In the first part, an existing system was modified to include software jointly developed by the authors of this paper "Open Source Computer-Vision Based Guidance System for UAVs On-Board Decision Making "[1]. During this part the software was modified and the algorithm was refined in order to be used in this research.

B- In the second part, the Robotic Operating System (ROS) was used to build a new system to provide more flexibility to control the different nodes in the system: a node for navigation, a node for target detection and one for ultrasonic sensor to control the altitude etc. Moreover, a simulation through the ROS environment was developed and used to ensure the system works well before the actual flight tests.

1.5.4 Stage 4: Hardware System Development

This stage consisted of developing and testing all hardware components including a quad copter UAV (3DR IRIS), an autopilot (Pixhawk), a single board computer (Odroid U3), a camera and a GPS receiver. AutoCAD is used to design the weed hardware enclosures and spraying mechanism. During this stage the wiring connection for the electronics devices such as ultrasonic sensor and the spraying motor was connected to the single board computer. The spraying tubes and the connection from the tank to the motor was also achieved as part of this stage.

1.5.5 Stage 5: Integration and Testing the System

This stage involved the integration of the software system and the detection algorithm from stage 3 and the physical hardware from stage 4. The complete on-board decision and action system was implemented on-board with a microcomputer such as (Raspberry Pi 2B or Odroid U3) for on-board decision making using the concept of OODA loop firmware. The on-board system was tested through simulation within the ROS and through the actual flight test and comparisons was made between them multiple times to determine the accuracy of the system and the type of the environment suitable for this algorithm.

1.6 Publications, Submission and Accepted Papers

Publications stemming from this work and the related works are listed in chronological order below:

- **PUBLISHED PAPERS**

1. H. Choi, M. Geeves, **B. Alsalam**, and F. Gonzalez, "Open Source Computer-Vision Based Guidance System for UAVs On-Board Decision Making", *IEEE Aerospace conference*, Big Sky, Montana, March 5–12, 2016.

2. S. L. Ward, J. Hensler, **B. H. Y. Alsalam**, and L. F. Gonzalez, "Autonomous UAVs Wildlife Monitoring and Tracking Using Thermal Imaging and Computer vision," *IEEE Aerospace conference*, Big Sky, Montana, March 5–12, 2016.

- **UNDER -REVIEW PAPER**

3. **Bilal Alsalam,** Duncan Campbell, and Felipe Gonzalez, "Invasive Spear Thistle Weed Detection and Zoysia grass Mapping Using Aerial Imagery", *Weed Research Journal,* 2017.

- **ACCEPTED PAPER**

4. **B. Alsalam**, K. Morton, D. Campbell and F. Gonzalez, "Autonomous UAV with Vision Based On-board Decision Making for Remote Sensing and Precision Agriculture", *IEEE Aerospace conference*, Big Sky, Montana, March 4–11, 2017.

Figure 1.1 illustrates the link between the different papers and the research objectives. The first two papers helped in generating a low cost system with on-board image processing using a Raspberry Pi 2B microcomputer interface with a camera [1, 2]. The techniques learnt from this paper helped to understand the process of connecting autopilot through a serial connection with the Raspberry Pi 2B. This allowed for the full control of the UAV while the UAV was flying to a target of interest [1]. A deeper understanding of computer vision and image processing for detecting a target and a method to transfer the target location from the camera frame to the global location GPS was acquired through the team research covered in this paper.

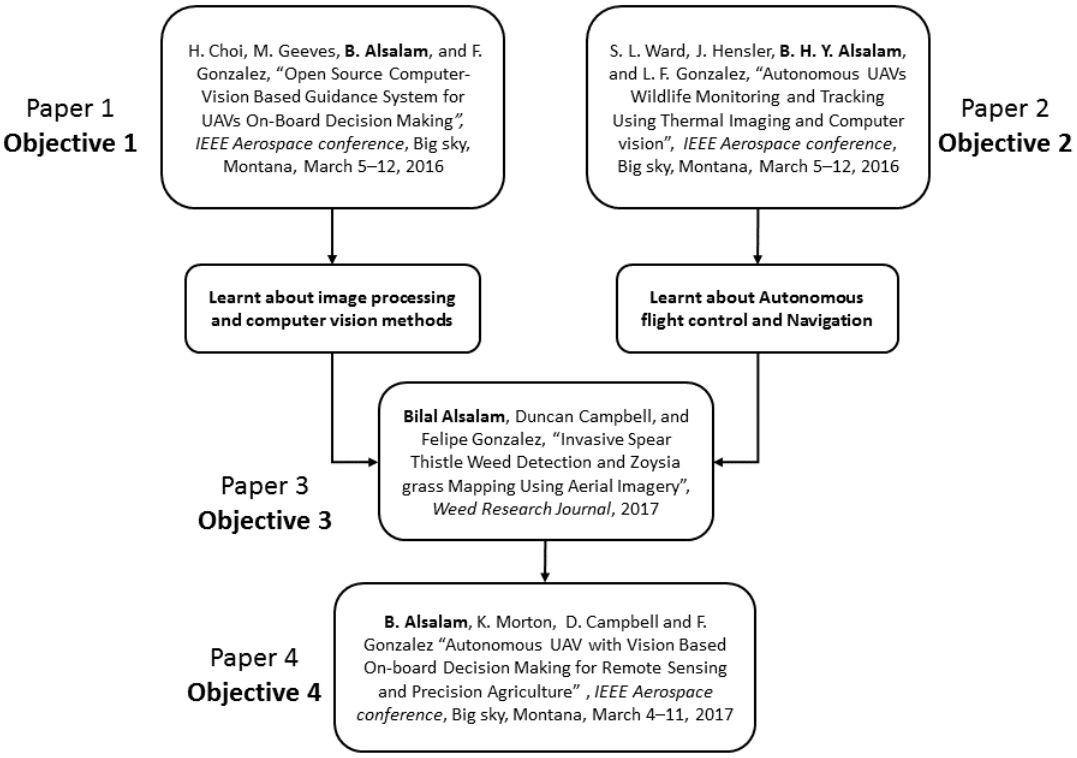


Figure 1.1: Papers related to the research.

The second paper on the other hand, assisted in gathering experience developing and coding the navigation system and autonomous control for the UAV. The paper described a system that can be used for a prediction dynamic application with a Raspberry Pi 2B microcomputer and a thermal camera [2]. The autonomous control algorithm that uses Python and a basic concept to control the flight path was developed from both papers. The previous papers assisted in determining suitable methods for navigation and the computer vision application which are helped to achieve the objectives of the research and used to develop the methods

presented in Chapter 5 (paper 3) and Chapter 6 (paper 4).

1.7 Outline of Thesis

The thesis presents a framework for off-board and on-board decision making for remote sensing and precision agriculture applications.

Chapter 2 presents a literature review in relation to using remote sensing, UAVs and computer vision for agriculture applications. An overview of the theoretical aspects of image processing and computer vision used for precision agriculture and remote sensing is covered. The theoretical aspects of the OODA loop and the OODA loop for on-board decision making are also presented.

Chapter 3 introduces the hardware system architecture for data collection and off-board analysis and mapping for invasive weed. The Chapter is divided into two parts, the first of which is the hardware design that includes the physical connection for the system and the 3D printing hardware for the system. The second part covers software design that includes Mission Planner software, Python scripts and the invasive weed detection method see Appendix B.1 and B.2.

Chapter 4 introduces the hardware system architecture for on-board decision making and action which is divided into two parts. The first part presents the system architecture (the hardware and the software) using a Raspberry Pi 2B and MAVProxy see Appendix B.2. The second part covers a system architecture (the hardware and the software) using Odroid U3 and the Robotic Operating System (ROS) see Appendix B.3, B.4, B.5, B.6, B.7 and B.9.

Chapter 5 describes the application of the off-board system for weed mapping and off-board decision making. This chapter is divided into three parts: (i) covering the type of weed, study side and the flight imagery; (ii) presenting the results and discussion; and (iii) providing a summary of the chapter.

Chapter 6 presents the autonomous UAV with vision-based navigation. This chapter is also divided into three sections: (i) covering the test cases including the marker, colour and invasive weed detection; (ii) the results analysis and the discussion; and (iii) a summary of the chapter.

Chapter 7 presents the conclusions of the work conducted, the limitations of this research and areas for future work.

CHAPTER 2

Literature Review

2.1 Overview

There is growing interest in developing and building UAVs with the capability of on-board decision making utilising computer vision, waypoint guidance and vision based control techniques [11, 12, 13]. Computer vision methods are utilized for civilian, military and farming applications [14] and this chapter presents a literature review of research in this field. The first section of the literature review will provide a brief discussion of remote sensing in precision agriculture as well as the state of art on the use of UAVs for Precision Agriculture. Computer vision for aerial imagery and OODA theory for on-board decision making will also be discussed. Furthermore, literature on previous research on invasive weed mapping using aerial imagery and vision based control will be presented. The literature will research and summarised on the type of application, the sensors type, microcomputer types and if the image processing was conducted on-board or off-board the UAV.

2.2 Remote Sensing in Agriculture

Remote sensing is the assembling of data about an object or phenomenon with no physical contact with the item [15, 16, 17]. Remote sensing in agriculture is a broad area which includes: crop categorization, crop mapping, weed forecasting, yield predictions, photosynthetic pigment content, etc. [18, 19, 20, 21, 22]. A critical necessity for providing reasonable remote sensing items in agriculture is the capacity to combine high spatial resolution and speedy turnaround times. There are numerous different types of platforms which can be used for remote sensing in precision agriculture such as manned aircraft, satellites and UAVs. Satellite based remote sensing technologies are used to identify agricultural problems because they have capacity to continuously monitor the Earth's surface. However, issues such as

optimum spatial and spectral resolution, the reversal time, repeat cycle and data gaining costs and sometimes the weather are the main factors influencing the usefulness of the satellite based techniques. [23, 24, 25]. Remotely sensed images captured from aeroplane and satellite platforms have been used to produce several examples of weed mapping in late growing periods [26, 27, 28, 29]. However, for many crops, the ideal treatment time for weeds is long before this point in the growth season when both the weeds and the crop are in their seedling stages [30]. Because of the significant impact of weeds in the agricultural sector, several researches have explored and used remotely sensed images captured from aircraft or satellite platforms which has resulted in several examples of weed mapping in late growth periods [26, 27, 28, 29]. Identifying small seedlings with airborne and satellite imagery is problematic due to the typically insufficient spatial resolution of aircraft or satellite platform imagery or the possible presence of cloud cover in satellite imagery[22, 29, 27].

Manned aircraft or satellite remote sensing platforms are very useful for weed detection at late phonological stages (e.g. flowering or senescence)[31][32]. UAV imagery, on the other hand, is especially useful for very early weed detection (e.g. seedling stage) [33, 34, 35]. As such, each platform has its own utility according to the scale of the problem, with UAVs displaying a remarkable opportunity for pre-emtive weed detection and mapping. Torres-Sánchez et al. [33] and Peña et al. [35] discussed an OBIA algorithm method in which their data was captured using six-band multispectral sensors (visible camera and a multispectral camera). Their study consisted of weed mapping in sunflower and maize crops, and the results show the effect of using different segmentation parameters in the detection methods. An OBIA for weed detection based on machine learning methods and combined with pattern and feature selection techniques was discussed by Pérez-Ortiz et al. [31]. Their results showed that the proposed methods for pattern selection were suitable and could lead to construction of robust set of data. Moreover, Laliberte and Rango [34] used OBIA with lightweight off the-shelf digital cameras to classify images of foliage and determine the optimal texture features for each segmentation scale. The classification results for the image were high, with an overall accuracy of 90%. Furthermore, Laliberte et al. [22] used image segmentation and object-based classification to monitor vegetation changes over time. Their study used 11 aerial images that were taken between 1937 to 2003. Their results show that the shrub cover increased “from 0.9% in 1937 to 13.1% in 2003, while grass cover declined from 18.5% to 1.9%”. Due to the image resolution obtained, the shrub and grass cover was estimated in sections >2 m^2, in which about

87% of the shrub cover was accurately detected. Laliberte et al. [29] expanded on this by using object-based recognition instead of pixel-based image information as input to classify trees for mapping arid land vegetation, and they used satellite imagery to segment at four different scales. Their combination of multi-resolution image segmentation and decision tree analysis facilitated the selection of input variables that helped to determine the appropriate image analysis scale.

2.3 UAVs for Precision Agriculture

Remote sensing using UAVs is a growing field of research that can assist farmers to assess plant health [4, 5, 36, 37]. UAVs can fly in a controlled autonomous path at very low heights and produce images at relatively high spatial resolutions (<2 cm) [3, 1, 2, 38, 39]. Remote sensing using UAVs can provide a low-cost option to deal with the basic prerequisites of spatial and dynamic resolutions in contrast to satellite and manned aircraft [18, 19, 20, 40]. A considerable number of studies in the field of precision agriculture have been carried out with either direct or indirect UAV application of remote sensing and computer vision. As a result, unmanned aerial platforms represent a remarkable opportunity for weed detection and mapping. Remote sensing techniques using thermal and multispectral imaging sensors at different heights were discussed by Han [3], Gonzalez-Dugo et al. [8], and Salami et al. [41]. Their research concentrates on filling the gap between the application prerequisites and the qualities of the various selected tools, payloads and platforms. A digital camera using a paraglider UAV was used by Dunford et al. [42] to oversee the evaluation of a riparian landscape and vegetation units, and furthermore, to distinguish standing dead wood. The results demonstrated an estimation of standing dead wood units and an average precision with omission and commission errors of 80% and 65%, respectively. McFadden et al. [43] evaluated UAVs for use in plant biosecurity. The research provided recommendations for the applicability of UAVs and on-board sensor technology for plant biosecurity and pest detection. Puig E et al. [44] illustrated the combination of UAV, remote sensing and machine learning techniques for biosecurity and applications in precision agriculture. They implemented an algorithm based on K-means clustering to control high-frequency components present in the feature space. Nebiker et al. [45] illustrated the gap between satellite-based remote sensing and ground-based sensing. Their study showed the benefits of remote sensing applications

using a very high resolution micro UAV platform. Hung et al. [19] applied high resolution aerial images from a digital camera for weed classification, collecting images at 5–10 metres. Their results showed 94% accuracy. A study by Knoth et al. [15] utilised either a panchromatic or shading infrared calibrated small frame digital camera to generate high resolution images of the re-established swamp surface utilising. Their system had an overall accuracy of 91% using UAV. Lelong et al. [7] focused on the combination of a single digital camera with spectral filters using four spectral bands matching to red, green, blue (RGB) and near infrared (NIR). The results of their trials showed an expected precision level of 15% in the biophysical parameters approximation. Rieke et al. [6] described precise position data by integrating the real time kinematic position of the UAV. The system has the potential to achieve an accuracy of 1-3 cm, which can be measured for direct geo-referencing for aerial imagery.

Berni et al. [46] focused on the most proficient method to produce quantitative remote sensing data on agriculture utilising two types of UAVs, rotary-wing and fixed wing, with thermal and narrowband multispectral sensor cameras. Their research was undertaken using thermal images in the "7.5–13-μm region (40 cm spatial resolution) and narrow band multispectral images in the 400–800 nm spectral region (20 cm spatial resolution)". The results illustrate that a low cost UAV system for vegetation monitoring applications has comparable estimation capabilities to the traditional manned airborne sensors [46]. Guo et al. [47] used a micro-helicopter UAV with a multispectral camera to develop a framework to process images for precision agriculture. Kelly [32] used a md4-1000 quadcopter UAV with a multispectral camera (MCA-6 camera) for weed mapping. The mini-MCA-6 camera is produced by Tetracam company (Chatsworth, CA , USA). Kelly's outcomes demonstrated that it is possible to create quantitative mapping products, for example, crop stress maps, from UAV images [32]. Von Bueren and Yule [10] used a hexakopter and quadcopter (MikroKopter) with different Multispectral Camera Arrays (MCAs) and digital camera sensors (Canon camera) to detect near objects in infrared light for precision agriculture. Their results demonstrated the successful application of an UAV with a multispectral imaging system to create high quality multispectral images [10]. Bryson et al. [48] used a small UAV with an on-board computer (PC104) for accurate image registration to detect weeds. Their results demonstrated that their approach to classification, which depends on generic colour and texture descriptors, can be utilised to distinguish between various sorts of vegetation. Furthermore, Laliberte et al. [38] used a small UAV with a digital camera and a procedure suitable for handling a large number

of UAV images. Their overall classification accuracies for the two collected image mosaics were 83% and 88%. The results of their study show that UAVs can be used successfully to obtain imagery for rangeland monitoring, and that a UAV-based remote sensing approach can either complement or replace some ground-based measurements. Figure 2.1 shows some examples of the UAVs which are used in agricultural research.

Figure 2.1: Examples of UAVs used for precision agriculture and plant biosecurity applications [10] [23] [34].

Table 2.1 lists some examples of UAVs used in agricultural applications (detailing authors, year of implementation, the UAV used, the sensor type, operational efficiency and the computer vision technique). High spatial resolution imaging in near real time and efficient onboard processing for precision agriculture applications have not been completed [23]. As a result, designers are continually looking for ways to develop UAV platforms with high resolution, near to real time imagery and high level decision making over extended periods of time.

2.4 Vision Based Control

Computer vision is a field which contains numerous methods for obtaining, analysing, processing and understanding images and high dimensional information captured in the different

Table 2.1: Summary of the literature review listing the applications, the UAV, the sensors, operational efficiency (on-board or off-board classification capacity).

No	Author's name	Year	The application	UAV used	Sensor used	Operational efficiency	Computer-Vision Method
1	Felderhof et al. [9]	2008	Monitoring plant health	UAV Glider CropCam	Digital camera (Sony 10MP) NIR camera	Off-board	Panoramic software AutoPano Pro and PTGui
2	Lelong et al. [7]	2008	Wheat crop	L'Avion Jaune's powered glider	Digital camera Canon EOS 350D	Off-board	Establish relationship between: leaf area and NDVI
3	Nebiker et al.[45]	2008	Monitoring crop status	Mini UAV (Zurich) and md4-200 quadcopter	MultiSpectral (MSMS) MicroSensor (Canon EOS 20D.)	Off-board	NDVI method
4	Han [3]	2009	Water management and agriculture applications	AggieAir UAV, called GhostFoto	Multispectral (Pentax Optio E10 camera) & thermal infrared (TIR) (Canon PowerShot SX100 IS)	On-board Using Gumstix computer	Algorithms based on NIR imagery and NDVI to detect river and vegetation
5	Dunford et al. [42]	2009	Classify and map riparian vegetation	A paraglider UAV	Digital cameras Canon Powershot G5 (5 MP&12 MP)	Off-board	Object-oriented analysis
6	Berni et al. [18]	2009	Coffee crops monitoring	Quanta-H rotary wing UAV and Quanta-G fixed wing UAV	Multispectral camera (Model USS-2000C) thermal camera (Thermovision A40M)	Off-board	SIFT algorithm Leica Photogrammetric Suite
7	Bryson et al. [48]	2010	Vision Based Mapping and Classification	J3 Cub UAV	Colour monocular camera	Off-board	Machine learning vision approach

8	Hunt et al. [37]	2010	Crop Monitoring	Vector-P UAV	FinePix S3 Pro UVIR camera	Off-board	GNDVI
9	Guo et al. [47]	2012	Mapping crop status	Micro-helicopter UAV	Multiple spectral cameras	Off-board	Lucas-Kanade method to detect feature points
10	Von Bueren and Yule [10]	2013	Monitoring agricultural paddocks	Hexakopter and Quadkopter (Mikrokopter)	Multispectral sensor camera (MCA) And digital camera	On-board	N/A
11	Gonzalez-Dugo et al. [8]	2013	Water status for fruit trees monitoring	Wingspan fixed wing	Thermal camera (MIRICLE 307)	Off-board	Crop water Stress Index (CWSI) method
12	Kelly [32]	2013	Weed mapping in maize fields	md4-1000 quadcopter UAV	Multispectral camera MCA-6 camera	Off-board	Object Based Image Analysis (OBIA)
13	Puig et al.[44]	2015	Assessment of crop insect damage	DJI S800 EVO	Sony NEX-5R high-resolution camera	Off-board	Unsupervised machine learning K-means clustering algorithm

- **N/A**: Not Available

situation in order to create numerical or typical data [49, 50]. Computer vision methods are utilised for military, civilian and farming applications [51]. The main objective of a vision based control technique in robotics is to control a robot (ground or aerial robot) to perform a predefined task using visual feedback, such as approaching an object or obstacle avoidance [12, 13, 52]. Liu and Dai [53] discussed visual servo techniques for on-board control for UAVs. Their research was based on aerial surveillance, "vision based navigation and airborne visual simultaneous localization and mapping". An approach for real time vision based landing for UAVs was designed and implemented by Saripalli S et al. [54]. Their research focused on navigation based on GPS data and computer vision and demonstrates that their navigation algorithm with computer vision can produce accurate results. Yang et al. [49] and Raja [55] discussed landing strategies for UAVs utilising visual control algorithms in order to detect landmarks. The authors utilised re-enacted flight video to check the precision of the system. Fu et al. [56] discussed vision based tracking algorithms for UAVs landing on an arbitrary field. Their real-time vision-based tracking algorithm was assessed with airborne pictures from auto landing of flights utilising a manually classified ground truth database. Their outcomes showed that the algorithm is very strong in tracking the helipad and accurate for closing the loop of vision based control. The Lucas-Kanade technique which is exceptionally useful to evaluate optical streams or movement between two successive images was utilised by Guo et al. [47]. An image based visual servo using a tracking parallel linear image feature for vertical take-off and landing was presented by Robert and Tarek [57]. Their controller designed for a small UAV is capable of quasi-stationary flight. Markus et al. [58] presented a framework for on-board vision based control in unknown environments (indoor and outdoor). Their method uses monocular vision to solve the estimation of the metric visual scale from an air pressure sensor. Choi et al. [1] and Ward et al. [2] discussed using a UAV with "computer vision based guidance system for on-board decision making". Their results showed that their algorithm can accurately recognise "99% of the object of interest" and the UAV is able to navigate and perform on-board decision making.

2.5 Off-board and On-board Hardware for Image Processing and Computer Vision

In order to apply computer vision, either on-board or off-board, the selection of a suitable computer on which to perform the processing is the main requirement. There are several types of computers and micro controllers which can be used for computer vision and remote sensing such as a desktop PC, laptop or FPGA, a GPU or microcomputers such as Odroid, Raspberry Pi etc. A suitable choice for the computer is based on several factors including: data type, the speed of processing and ease of use for the farmer. Kelly [32], for instance, used an off-board OBIA procedure to compute multiple sets of data and statistics derived from the classification outputs in order to have a suitable process to generate processed imagery. The results of his research demonstrated 86% overall accuracy, that 23% of the area was free of weeds, and 47% of the area had low amounts of weed and the resultant treatment recommendation showed a high potential to reduce herbicide application or other weed control processes.

2.6 Decision Making using the OODA Loop Framework

There are several approaches for on-board decision making, where one possible approach is use to the Observation, Orientation, Decision, and Action (OODA) loop method in the context of UAVs. OODA loop theory was proposed by John Boyd [59] and it has since been applied in models used in defence to describe decision making [60, 61]. Parasuraman et al. [62], for example, discussed a model based on decision making with four classes: data procurement, data investigation, decision and action choice, and lastly, action implementation. Each class can be applied as a manual activity or completely autonomous. Peng et al. [63] discussed the difficulties and the challenges for multiple networked UAVs that were analysed based on the OODA model. They discussed three key technologies for the UAVs: cooperative control, cooperative information sensing and the mission decision under a dynamic network, and multiple autonomous vehicles. Decision making based on the OODA loop method varies between an off-board and on-board implementation due to the fact that the decision and the action in the OODA loop will be passive in off-board decision making but active in the on-board case. Figure 2.2 illustrates the concept of onboard decision making. The OODA loop

has been applied in the context of UAVs, however, the application of off-board and on-board decision making for precision agriculture is limited to date.

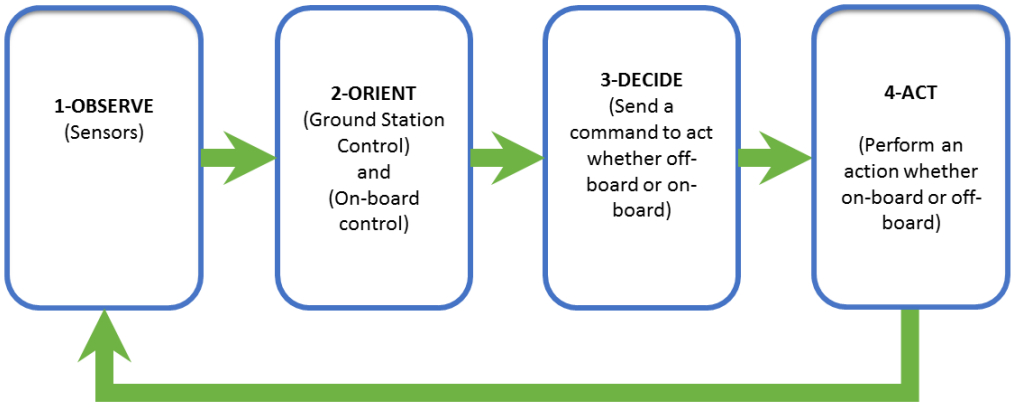


Figure 2.2: Simplified version of Boyd's OODA Loop.

2.7 Summary

The purpose of this chapter was to review the previous research experiments in this field and determine the most appropriate methods to achieve the task of on-board decision making for agricultural applications. This chapter presented a literature review of remote sensing in precision agriculture as well as materials and methods which include the state-of- the-art methods for the use of UAVs in precision agriculture. Computer vision and vision based control have also been discussed for onboard decision making. Furthermore, an investigation of the OODA loop method has been presented.

In this work, both off-board and on-board, the OODA loop will be implemented and applied in the context of precision agriculture.

The next chapter describes the system architecture for data collection and off-board analysis for invasive weed. The chapter will cover the hardware system including electrical electronic integration and 3D printing as well as the software integration.

CHAPTER 3

System Architecture For Data Collection and Off-board Analysis and Mapping

3.1 Overview

As described in Chapter 1, the purpose of this research is to investigate and develop a system for off-board and on-board decision making using UAVs in the context of precision agriculture.

Chapter 2 discussed the literature review and research objective and also the materials and theoretical methods applied in the research such as OODA loop theory and vision-based control.

This chapter presents the design of the system architecture for data collection and off-board analysis and mapping. It is divided into two parts, the first of which covers the hardware design which includes the physical connection for the system and the 3D printing hardware. The second part covers software design and includes Mission Planner software, Python scripts and the invasive weed detection method.

3.2 Hardware Design

The system architecture for data collection and off-board analysis consists of two parts as shown in Figures 3.2 and 3.1. The on-board system consists of a quadcopter UAV (3DR IRIS), an autopilot (Pixhawk), a microprocessor Raspberry Pi 2B, a 5 MP Raspberry Pi camera and a GPS 3DR (GPSKIT0003). The Raspberry Pi connected to the ground station uses the Wi-Fi network. A 915 MHz radio control is also used to connect the ground control station system to the Pixhawk. The ground station consists of 3DR telemetry, WiFi adapter and FR-Sky receiver.

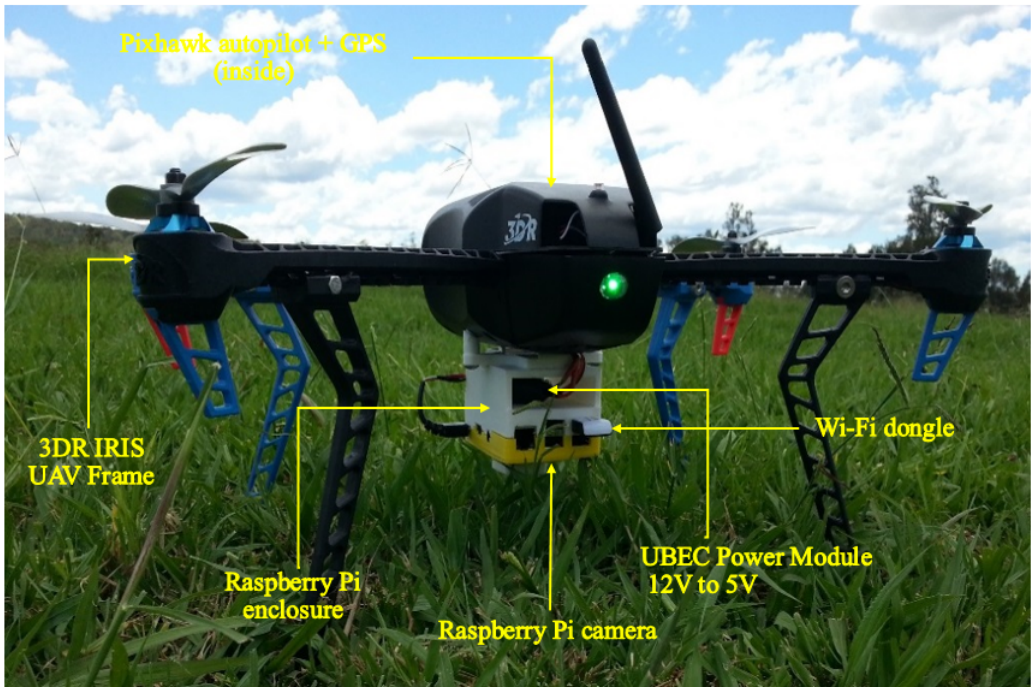


Figure 3.1: On-board system for collecting data.

3.2.1 3DR-IRIS UAV Frame

The 3DR IRIS UAV frame is suitable for the purpose and relatively low in cost (< $2000 AUD, 2016) compared to other platforms. The 3D Robotics IRIS UAV has a payload capacity of 425 grams. The airframe is made of strong and resistant plastic for the crash. The dimensions of the frame are 550 mm from motor to motor and a height of 100 mm from the centre of the frame [64].

3.2.2 AC2830-358 850Kv Motors and 10x4.7 propellers

The UAV uses four AC2830-358 850Kv motors and four 10x4.7 propellers to produce thrust. The AC2830-358 850Kv motor is small in size, and is designed to produce high thrust. Two motors run counter clockwise and another two run clockwise. Specifically so as to give pitch, roll and yaw control [64].

3.2.3 Pixhawk Autopilot

The IRIS uses a Pixhawk flight controller; an open source autopilot designed and manufactured by 3D Robotics which has extensive online documentation and support. This level of customisation makes it suitable for developing a system that can be modified to suit various

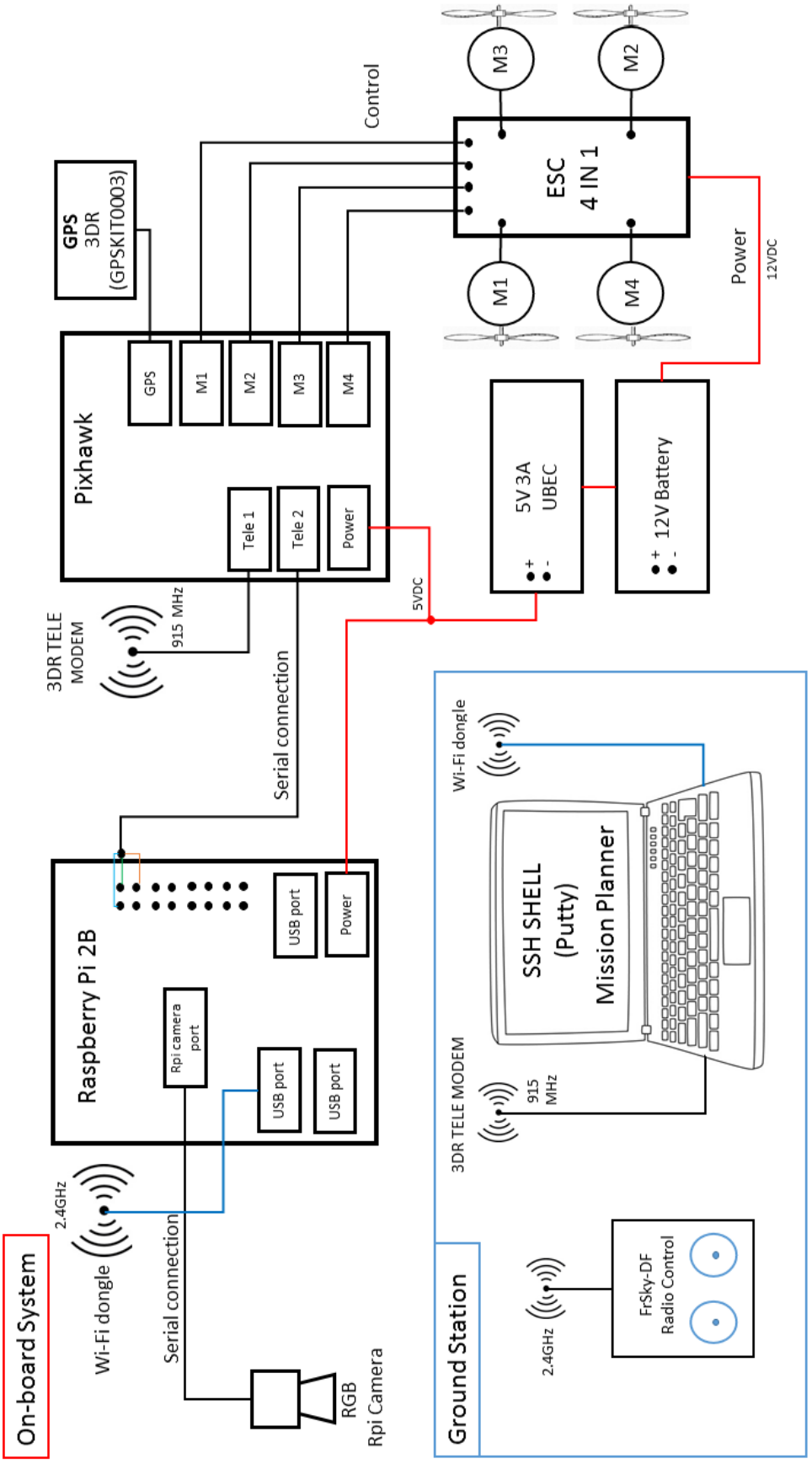


Figure 3.2: System Architecture for Data Collection and Off-board Analysis and Mapping.

applications. The Pixhawk autopilot includes both software and hardware components to interface with all other subsystems. Pixhawk contains an in-built accelerometer, gyroscope, barometer and magnetometer [65].

3.2.4 GPS Compass Module

A 3DR (GPSKIT0003) 6H compass/GPS module is used on the system. This is a stand alone type GPS operating system at 2.7 to 3.6 voltage. This GPS has sensitivity of -162 dBm, and the velocity and heading accuracy are 0.1 m/s and 0.5 degrees, respectively [51]. The GPS can be used with APM and Pixhawk flight control systems.

3.2.5 4 in 1 ESC/Power Module

The system architecture uses a 4 in 1 ESC/Power module which is inbuilt with the 3D IRIS UAV. Each ESC is rated at 20amps capacity and is running the SImonK firmware for enhanced response and stability. The ESC power module provides regulated power to the flight control board, furthermore it monitors both the voltage and the current. Using 4 in 1 ESC can replace several components including a power regulation board, a power distribution and four ESCs compared with similar UAV with 4 ESC [51].

3.2.6 5000 mAh 3S 30C Lipo Pack Battery

The system architecture uses a 5000 mAh 3S 30C, 11.1 V DC Lipo battery which has a capacity of 5000 mAh and a 30C discharging rate. The battery allows for approximately 15–20 minutes flight time without any payload which is sufficient to complete the test [51].

3.2.7 FrSky-DF Radio Control (Tx/Rx)

The system can also be controlled manually using a FrSky DF 2.4 GHz transmitter and receiver link which transmits signals to the autopilot [64].

3.2.8 3DR TELE Radio Modem and Ground Station Control

The system uses two 3DR radio modems which are responsible for transmitting signals between the on-board sensor and the ground control station. The telemetry is sent via

communication between the two units at an operating frequency of 915 MHz. The ground station computer (GCS) runs using Mission Planner software [64].

3.2.9 WiFi connection

a wifi network is used to connect the on-board microcomputer (i.e. Raspberry Pi 2B, Odroid U3^{+}) with the ground control station in order to upload commands via an SSH connection to the on-board system.

3.2.10 Microcomputer (Raspberry Pi 2B)

The Raspberry Pi 2B is a single board microcomputer of small dimensions (85 x 49 mm). The board is priced between $40–$55 AUD (July 2016). The Raspberry Pi 2B hardware uses input/output calls (GPIO) in order to receive/transmit signals from/to the sensor. The microcomputer can be operated using different software such as: Raspbian, OSMC (Open Source Media Centre), Linux and Windows 10 IoT. The Raspberry Pi 2B board weighs 45 grams which makes it suitable to use on an UAV as a small on-board computer (see Appendix A.1) [66]. A script running on the Raspberry Pi 2B allows the system to take images every second to collect data for off-board analysis and mapping.

3.2.11 Raspberry Pi Camera

The Raspberry Pi camera is a 5 megapixel fixed focus camera which supports 1080p 30, 720p 60 and VGA90 video modes. The camera's dimensions are 25 x 20 x 9 mm and it attaches via a 15 cm ribbon cable to the CSI port on the Raspberry Pi 2B. The camera can be accessed through the Picamera Python library or the ROS environment in order to capture frames for image processing [66].

3.2.12 Universal Battery Elimination Circuit 5V-3A (UBEC)

A UBEC is a switch mode DC regulator which takes voltage of 12 V DC from the main battery for the UAV and converts it to 5 V DC in order to power the Raspberry Pi 2B or Odroid U3. The power input wires of the UBEC are joined into the power input wires of the ESC while the power yield wires of the UBEC receiver are connected to the required device (the Raspberry Pi 2B or the Odroid U3).

3.2.13 Ground Station Computer (GSC)

The ground station consists of a laptop which runs Mission Planner software and via an SSH shell (PUTTY) allows the system to send and receive commands for the mission.

3.2.14 3D Printing for Off-board System

Figure 3.3 shows a 3D printing of an enclosure designed to install, secure and protect the Raspberry Pi 2B, the Raspberry Pi camera and the UBEC. The enclosure was designed using AutoCAD software. Once each surface has been coated with acetone, the surfaces are brought and clamped together. The enclosure is mounted as close as possible to the centre of gravity of the airframe.

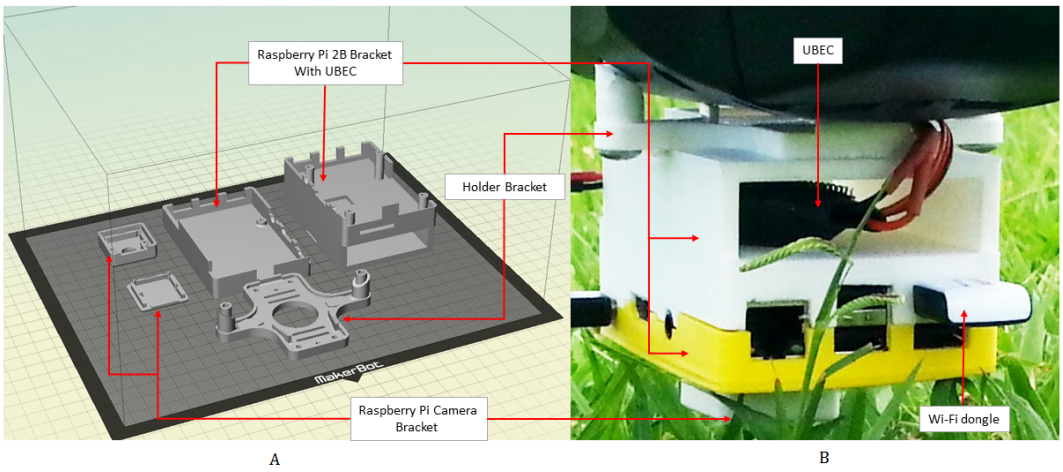


Figure 3.3: A: 3D printing design for the Off-board System. **B**: The enclosure system for the Off-board System and Data Collection.

3.3 Detection and GPS Mapping Approach

Figure 3.4 describes the process flow of information developed in this work. Steps 1–8 consist of planning a series of waypoints for a maximum area of coverage with a 40% overlap. The Raspberry Pi receives commands from the ground station using Putty to record video and to take images of the region of interest at 1 frame per second. In steps 9 and 10, the images and videos are uploaded post flight and processed together with the GPS waypoints logs. Once this is completed, the next step (step 11) compares the time in the image with GPS with the UAV time logs; the algorithm takes into account the date and time the image was created and finds the matching date and time from the flight log.

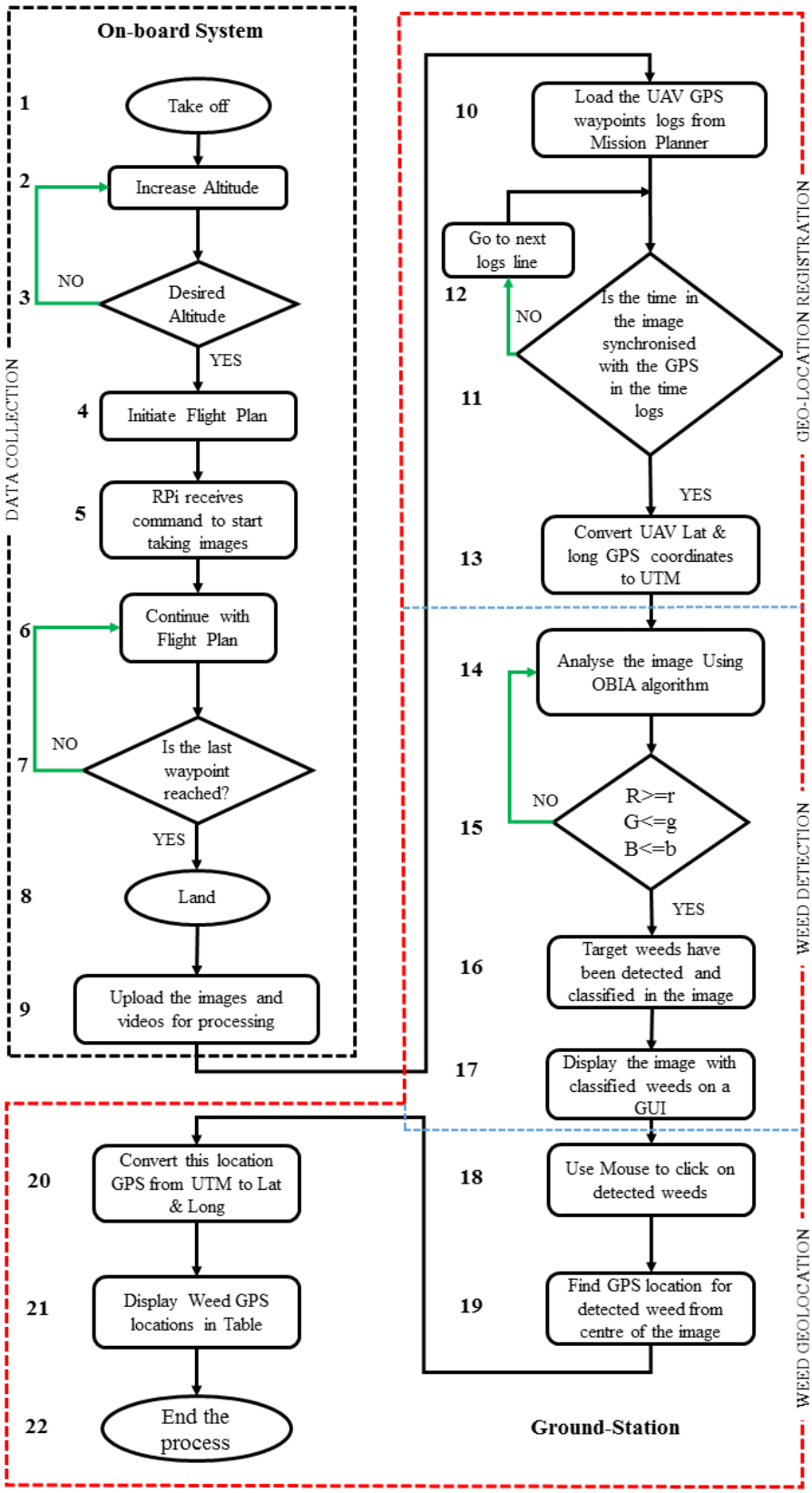


Figure 3.4: Image processing flowchart.

If the time does not match, the next step is to go to the next line in the log (step 12), and once a match is found, the geo-location of the point of interest in degrees is converted into Universal Transverse Mercator (UTM) format and saved (step 13). In step 13, the centre GPS coordinates of the image which correspond to the UAV relocated coordinates are used to calculate the x and y distance in metres from the centre of the image to any point on the image. In steps 14–16, an OBIA and threshold selection method algorithm (see section 3.4.3) are applied to the image to detect if there are any weeds in the image. After the weeds have been detected in steps 14–16, the processed single image is displayed (step 17). In steps 18 and 19, the user can via a mouse pointer select any of the detected weeds on each individual or stitched image to create a weed map with GPS coordinates. In step 20, the UTM coordinates for each weed are converted into GPS coordinates in degrees. Lastly, in step 21, the GPS location of the detected weed is displayed in tabular form.

3.4 Software Design

The software used for this system consists of: Mission Planner software to create the waypoint flight path, Python scripts to record video and take images and a weed detection and mapping method.

3.4.1 Mission Planner

Mission Planner software is used on the ground control station for the UAV. Mission Planner is used to select waypoints for the mission for data collection which is then used for off-board analysis and mapping [67]. Mission Planner can be also used to load firmware into the autopilot (Pixhawk), and set up configure and load an autonomous mission into the autopilot. Furthermore, Mission Planner can be used to save and analyse the mission logs which contain information about the mission. Figure 3.5 shows an example of waypoints that are used to cover the area of interest.

Figure 3.5: Mission Planner software.

3.4.2 Python Script

The second type of software used are Python scripts which run on the Raspberry Pi 2B. The Raspberry Pi camera takes one image every one second with a 40% overlap in order to cover the entire area of interest. Before commencing image capture, the clocks in the Raspberry Pi 2B and the GSC need to synchronised and the following script is run *(sudo date –s Time)* to achieve this purpose. To capture images from Raspberry Pi camera for 4 minutes, that is from 1 second (1000 ms) to 4 minutes (240000 ms), the following script is run:

raspistill -o Image_name_%d.jpg -tl 1000 -t 240000

3.4.3 Invasive Weed Detection Method

An invasive weed detection algorithm using an OBIA algorithm and a threshold selection method was developed. Initially, several images of the target (i.e. spear thistle weed) of different size and/or stages of growth were collected from 1 and 3 metres above the ground. Each image is analysed using an OBIA algorithm and a threshold selection method with the true colour and texture image data using RGB triplet (Figure 3.6). Each RGB triplet defines a colour for one pixel of the image in three layers. The first layer of the 3D array (colours band) contains red components, the second layer green components and the third layer blue components. In order to detect the weed, the threshold number must be chosen for each colour component and a set of colour thresholds is generated. The optimum threshold values

were achieved followed by manual tuning to obtain the greatest sensitivity and selectivity.

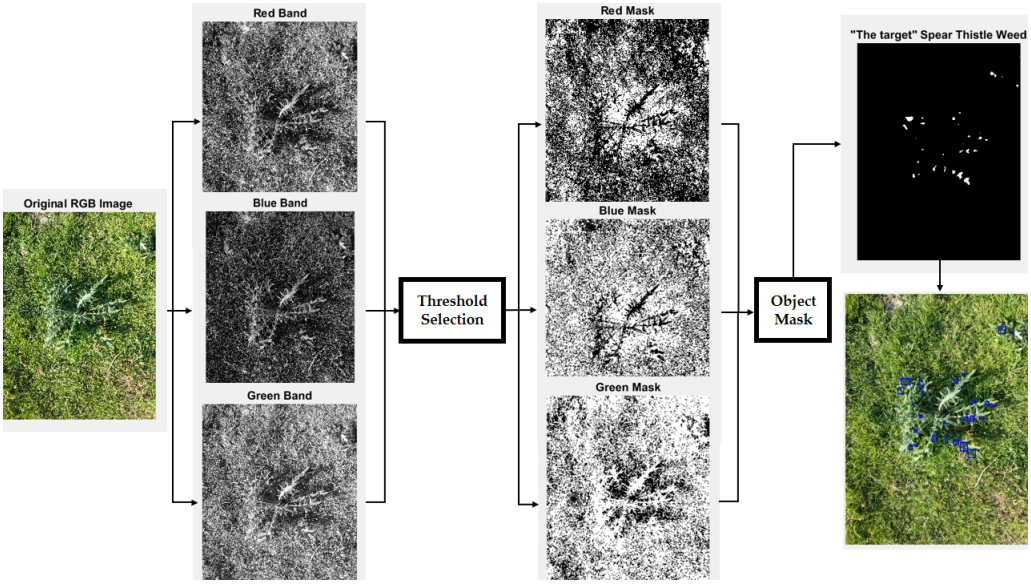


Figure 3.6: Image processing stages.

An invasive weed that is mostly green in colour will have the following colour combination: (green band > green threshold), (blue band < blue threshold) and (red band < red threshold). The result of these is masked RGB. The channels are combined to give the binary targeted image. However, an invasive weed that is mostly brown in colour features will have the following colour combination: (red band > red threshold), (green band < green threshold) and (blue band < blue threshold). Figures 3.7A and 3.7B show the images taken at 1 and 3 metres above ground respectively. The data collection process was repeated with a RPi camera mounted on the UAV at 5, 7 and 15 meters height from the ground level (See 5.5 Results and Discussion). The algorithm is also capable of detecting weeds in early age as well as when the part of the weed is senescent. Appendix B.6 provides the code for the invasive weed detection algorithm.

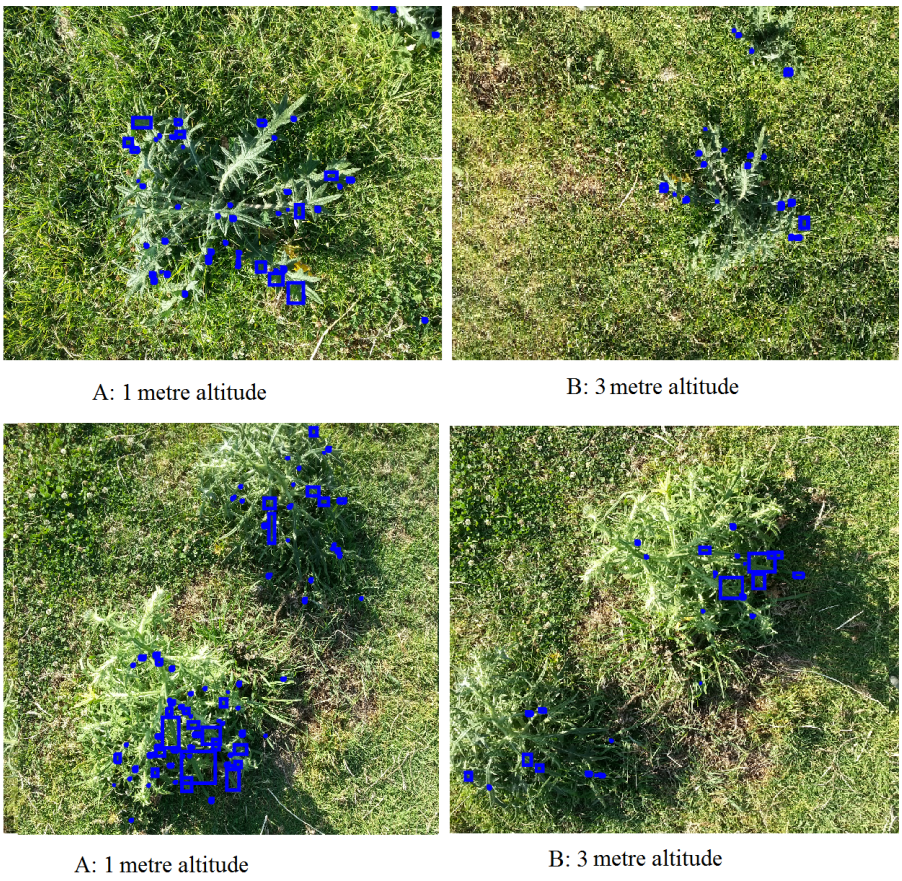


Figure 3.7: Spear thistle weed detection and classification at 1 and 3 metres height at different stages of growth. (November 2015).

3.5 Summary

This chapter introduced a clear framework for the system architecture design for the off-board system (analysis and mapping) using a Raspberry Pi 2B. A detailed description of the off-board decision making including the platform and different subsystems have been presented. An additional set of software design including Mission Planner, a Python Script and an Invasive Weed Detection Method were also introduced.

The next chapter describes the system architecture design for on-board decision making and action (the hardware and the software) using a Raspberry Pi 2B and MAVProxy or using an Odroid U3 and Robotic Operating System (ROS).

CHAPTER 4

System Architecture Design For On-board Decision Making and Action

4.1 Overview

As described in Chapter 1, the purpose of this research is to investigate and develop a system for off-board and on-board decision making using UAVs in the context of precision agriculture.

Chapter 2 discussed the literature review and research objective and also the materials and theoretical methods applied in the research such as OODA loop theory and vision-based control.

Chapter 3 presented the design of the system architecture for data collection and off-board analysis and mapping including the hardware and software design.

This chapter covers the design of the system architecture for on-board decision making and action. The chapter is divided into two sections: The first section will cover hardware and software system architecture for on-board decision making using Raspberry Pi 2B and MAVProxy. The second section describes hardware and the software using Odroid U3 and Robotic Operating System (ROS).

4.2 System Architecture Using Raspberry Pi 2B and MAV Proxy

The system architecture is similar to the system architecture for data collection and off-board analysis and mapping described in Chapter 3, section 3.2, however, two elements have been added for on-board decision making.

These two new elements are an ultrasonic sensor (HC-SR04) and a 5 V DC relay (SR-05VDC) to run the 12 V DC motor for spraying. Figure 4.1 shows the various elements of the on-board system architecture and the ground station while Figure 4.2 shows the hardware elements for on-board decision making and action (chemical spray).

The hardware consists of the same hardware used in Chapter 3 for data collection and off-board analysis and mapping. It consists of a 3DR-IRIS UAV frame, AC2830-358 850 KV motors and 10x4.7 propellers, Pixhawk Autopilot, GPS Compass Module, 4 in 1 ESC/Power Module, 5000 mAh 3S 30C Lipo Pack Battery, FrSky-DF Radio Control (Tx/Rx), WiFi connection, microcomputer (Raspberry Pi 2B), UBEC, and GSC with the following added component:

4.2.1 Ultrasonic Sensor (HC-SR04)

The HC-SR04 ultrasonic sensor is used to measure the flight hight by emitting an ultrasonic wave in one direction and starting timing from when the ultrasonic wave is launched. The ultrasonic spread velocity is 340 m/s and is based on the timer. The distance can be calculated between the obstacle and transmitter [68]. The ultrasonic is connected to the Raspberry Pi 2B to the GPIO connection pins (see Figure 4.5), or connected to Arduino which is connected to Odroid U3 in the second system (see Figure 4.13). The HC-SR04 Ultrasonic Sensor Specifications can be found in Appendix A.2.

4.2.2 HD Webcam Logitech C270

The HD Webcam Logitech C270 is used to capture frames for image processing. In addition, the Raspberry Pi camera is used as a backup and to record video. The HD Webcam Logitech C270 Specifications [69] can be found in Appendix A.3. The large Signal to Noise Ratio (SNR) for webcam are based on two factors: the ISO speed and the exposure.

4.2.3 Relay (SRD-05VDC)

The Relay SRD-05VDC has a capacity of 5 A is used to connect the 5 V GPIO Raspberry Pi terminal to the 12 V spraying pump.

4.2.4 Spraying-Pump

The 12 V DC motor drives a spraying pump to perform an action on the target (see Figure 4.2**B**).

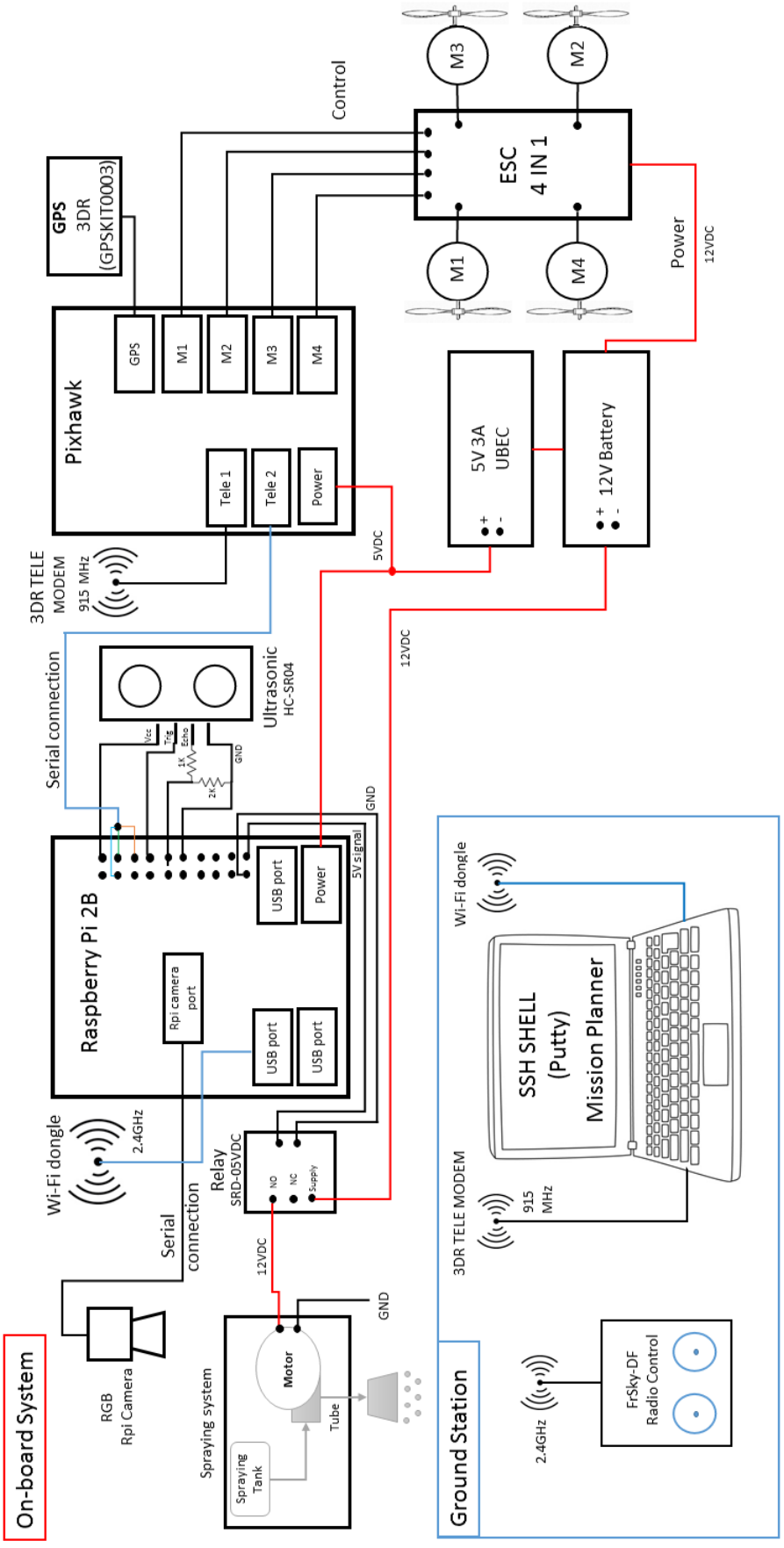


Figure 4.1: System Architecture for On-board Decision Making and Action Using a Raspberry Pi 2B and MAVProxy.

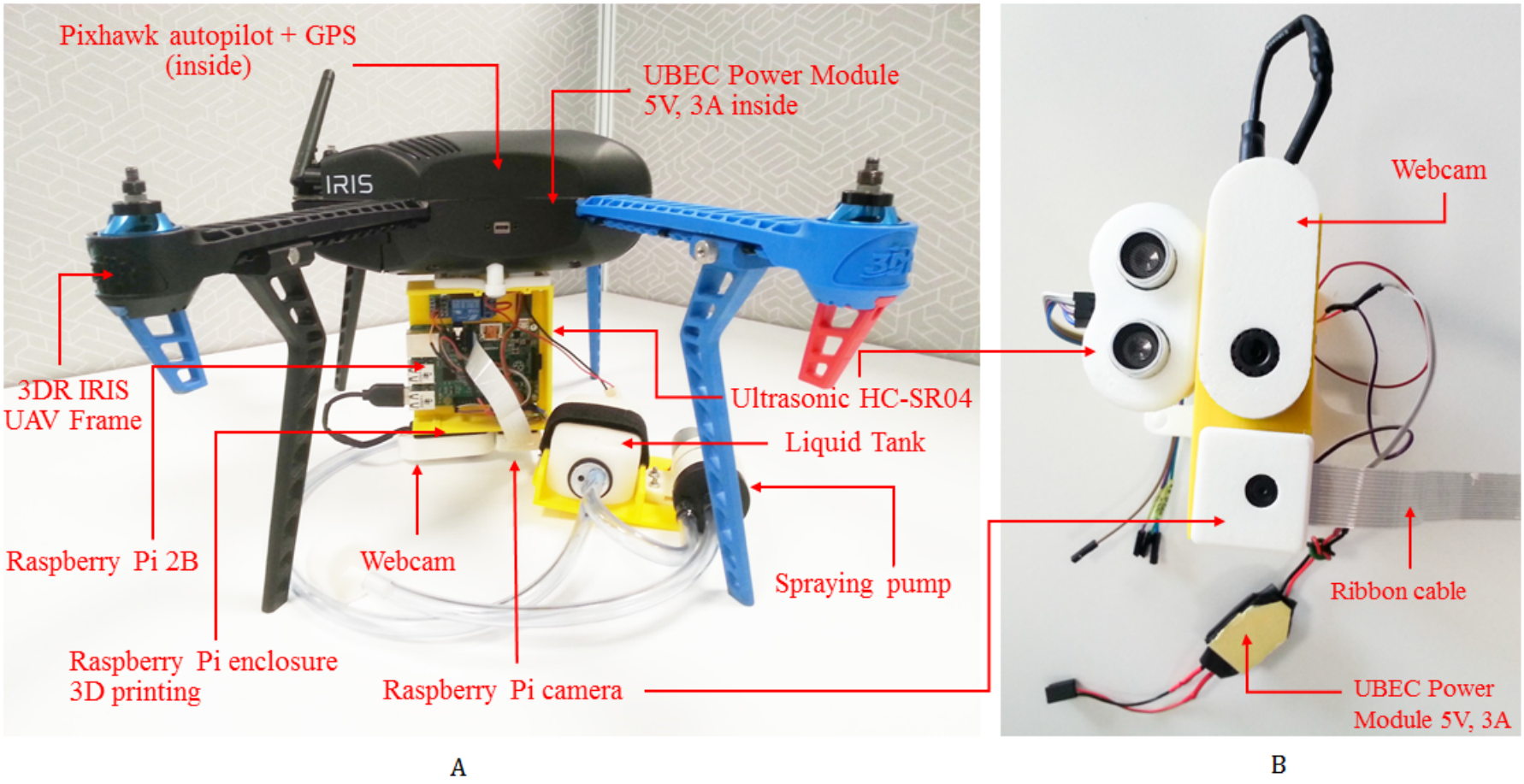


Figure 4.2: **A**: Hardware System Architecture for On-board Decision Making and Action Using a Raspberry Pi 2B and MAVProxy. **B**: Top view to the payload.

4.2.5 Liquid-Tank

A 200 ml capacity tank (4.2**B**) is used to store the liquid (e.g. pesticide) to be sprayed on the target.

4.2.6 3D Printing for the On-board System

Figure 4.3 shows the design for the system requires a 3D printing design for the spraying system bracket and holder. AutoCAD was used to design the required parts. The 3D printed parts consist of a holder, an ultrasonic cover, a cover for the Raspberry Pi 2B and the relay, a cover for the Raspberry Pi camera and for the webcam, and an upper cover is designed to hold the chemical tank.

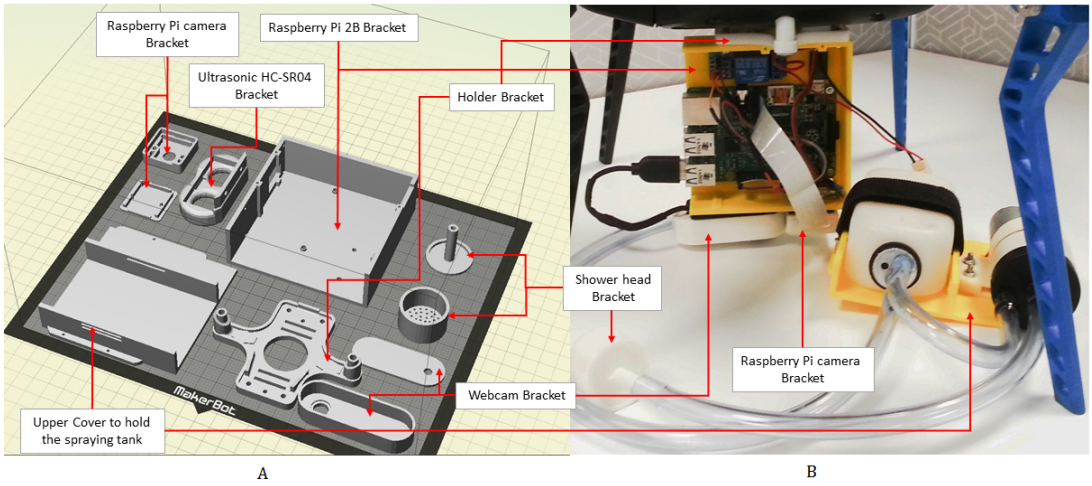


Figure 4.3: A: 3D Printing Design for the On-board System. **B**: The Enclosure System for On-board Decision Making and Action.

4.2.7 Electrical Integration

4.2.7.1- Pixhawk – Raspberry Pi 2B Serial Interface

A customised serial cable was implemented to interface between the Raspberry Pi 2B and the autopilot as shown in Figure 4.4 [64]. The customised cable was a spare telemetry cable with a 6-Postion DF13 plug which was spliced and soldered with individual PCB female jumper wires. The DF13 plug is inserted into the secondary telemetry port (Telem2) of the Pixhawk. Port Telem1 remains for the 3DR radio modem such that both Mission Planner and the Raspberry Pi can receive telemetry data as a fail-safe protocol. After connecting to the Pixhawk, the female jumper wires are connected to the Raspberry Pi serial port pins and ground (Tx,Rx,Gnd) [64].

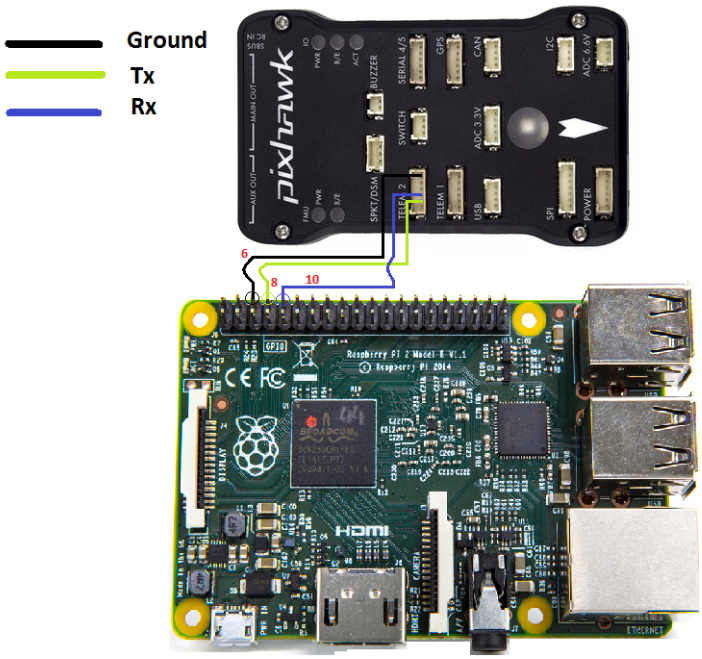


Figure 4.4: Connecting the Pixhawk to Raspberry Pi 2B using a customised serial connected cable.

4.2.7.2- Ultrasonic Sensor – Raspberry Pi 2B Serial Interface

Four GPIO pins (1, 16, 18 and 20) from the Raspberry Pi 2B are used to connect the HC-SR04 ultrasonic sensor to the Raspberry Pi: these are Echo Pulse Output (ECHO), ground (GND), 5V Supply (Vcc) and Trigger Pulse Input (TRIG) as shown in Figure 4.5. The ultrasonic sensor can be powered by the Raspberry Pi 2B to send the signal to the TRIG pin in the ultrasonic sensor. The pulse waves bounce off any adjacent objects and are reflected back to the ultrasonic sensor. The sensor recognises these arrival waves and measures the time between the trigger and returned pulse, and then a 5 V signal will be sent to the ECHO pin [68].

In order to connect the HC-SR04 ultrasonic sensor to the Raspberry Pi 2B, two resistances (1 kΩ and 2 kΩ) must be connected to increase the voltage from 3.3 V to 5 V for the pin 18 for GPIO of the Raspberry Pi 2B (Vin). A voltage divider is used because the output signal of the sensor (ECHO) on the Ultrasonic module (HC-SR04) is evaluated at 5 V, and the input pin 16 on the Raspberry Pi 2B GPIO is evaluated at 3.3 V.

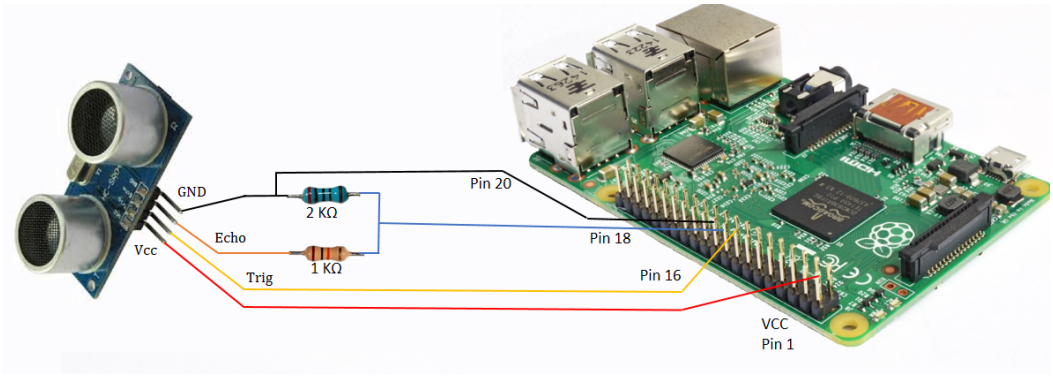


Figure 4.5: Ultrasonic Sensor – Raspberry Pi Serial Interface.

4.2.7.3- Relay – 12 V DC motor – Raspberry Pi 2B Serial Interface

The circuit shown in Figure 4.6 is used for spraying system. The spraying pump uses a 12 V DC motor which is powered by the same 11.1 V DC battery that powers the UAV. The 5 V DC relay (SRD-05VDC) is used to run the 12 V DC Raspberry Pi 2B as shown in Figure 4.6. The 5 V DC relay isolates the 5 V and 12 V. The GPIO signal voltage is 3.3 V and the motor can be run at 10–12 V.

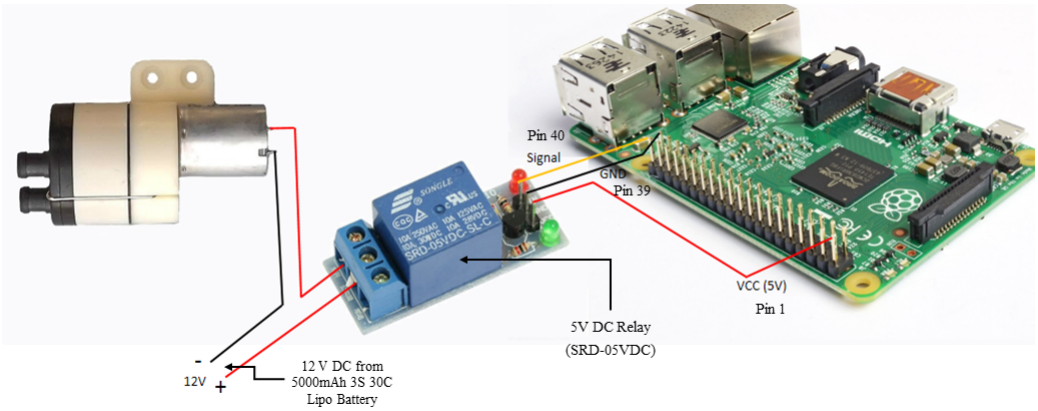


Figure 4.6: Interface Raspberry Pi 2B with Relay and 12 V DC Motor.

4.2.8 Software Design Using Raspberry Pi 2B and MAVProxy

The on-board software system using a Raspberry Pi 2B consists of: a MAVproxy configuration and a Python script.

4.2.8.1- MAVProxy Communication

MAVProxy is used to connect the Ground Control Station (GCS) to the UAV. The GCS for the system supports MAVLink protocol in order to make connection between the Pixhawk (with APM firmware) and the GCS. The MavProxy commands are written using Python script which allows MavProxy to be interfaced with other library software such as OpenCV [70]. A WiFi network is used to connect the Raspberry Pi 2B to the GSC. A remote SSH terminal [71] can be established to initiate serial communications between the Raspberry Pi 2B and the Pixhawk. The following commands are executed [72] as shown in Figure 4.7.

sudo -s: This command runs a new shell as root with higher privileges than the standard runtime environment. The shell user prompt will change to

(root@Raspberrypi:/home/pi#).

Following this, Mavproxy can be executed by typing [72].

MAVProxy.py –master=/dev/ttyAMA0 –baudrate 57600 –aircraft MyCopter

After executing this command, the Raspberry Pi 2B starts serial communications with the Pixhawk through the customised cables discussed in section 4.2.7.1. The prerequisite for running this command is to set the telemetry band rate for the Pixhawk 57600. If the band rates do not match between the Raspberry Pi 2B and the autopilot, the information will not be able to be transferred.

The information related to the Pixhawk autopilot is displayed when the communication has been successfully established (e.g. Mode STABILIZE), followed by a number of parameters received (see Figure 4.7). The 3D IRIS UAV was powered on with the mode switched to STB or “Stabilize” in the RC control. To confirm whether or not the connection with the autopilot is successful, toggle the mode into LTR or “Loiter” mode and read the output of the terminal; this is confirm that the Raspberry Pi 2B is successfully connected to the autopilot through Mavproxy.

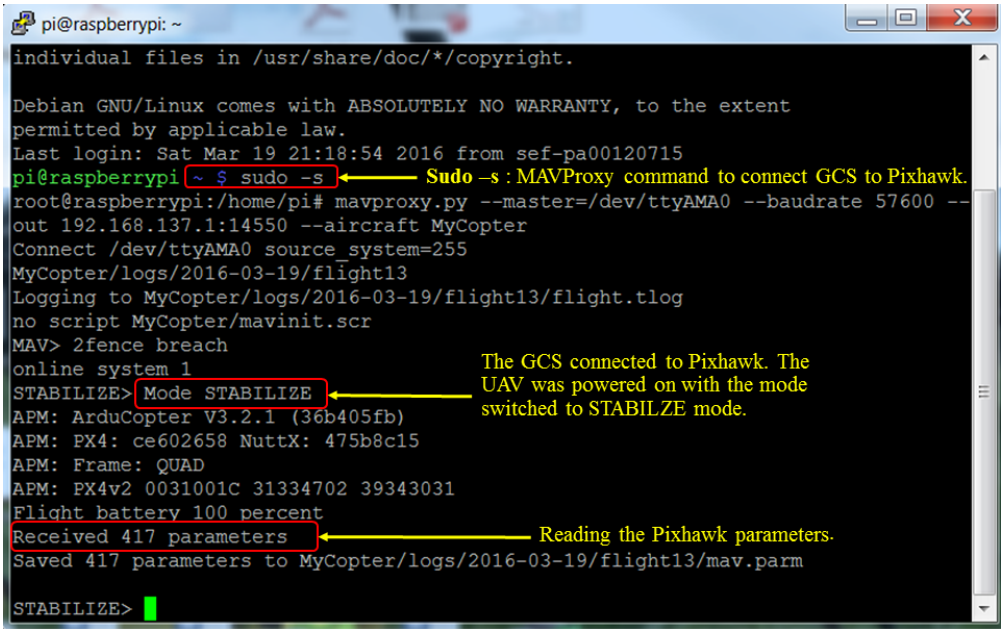


Figure 4.7: Remote SSH terminal accessing the Raspberry Pi 2B using SHH Putty and running sudo -s, Mode STABILIZE executed.

After completing the previous steps, the Raspberry Pi 2B (microcomputer) is connected to the Pixhawk, similar to connecting Mission Planner on a windows machine to the Pixhawk. Figure 4.8 shows arming the UAV through the SSH terminal on the Raspberry Pi 2B.

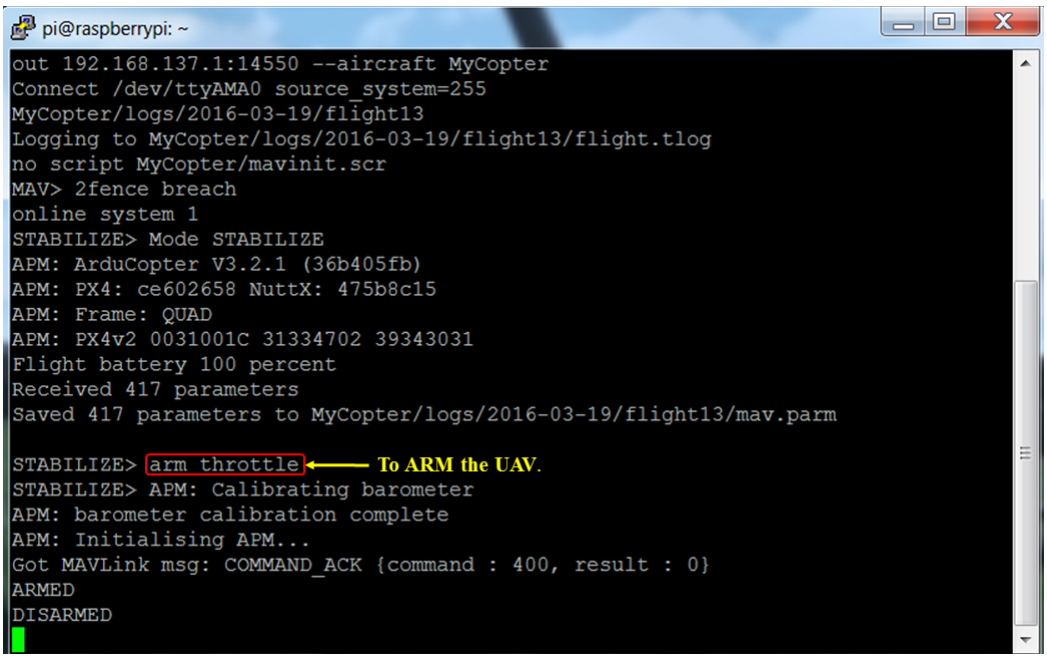


Figure 4.8: Arm the UAV through the SSH terminal on the Raspberry Pi 2B.

4.2.8.2- Python Scripts

In order to execute any Python scripts using MAVProxy, the DroneKit API must first be installed and initiated (Figure 4.9). This command (*module load droneapi.module.api)* should be run when the STABILIZE mode is achieved.

After completing the previous steps, a Python script *(e.g. spray_weed.py*) can be run through the MAVLink environment by using the following command (Figure 4.9). A complete copy of the Python code can be found in Appendix B.2.

api start [The Name of The Code].py

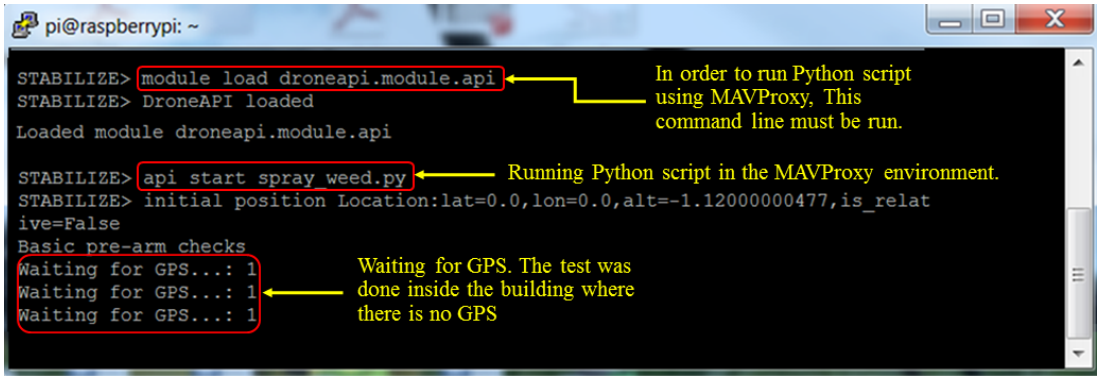


Figure 4.9: Running the Image Processing Code through MAVProxy.

The software design using Rasperry Pi 2B and MAVProxy was tested in the field at Christmas Creek (Queensland, Australia). Even though the system is useful but it has several limitations.

1. A long delay (>60 seconds) to connect the operating computer to the on-board system.
2. The connection between the Raspberry Pi 2B and the MAVproxy frequently crashes after the Python script runs to fly the UAV, resulting in loss of control of the UAV.
3. The Python code is written as a single script in order to control the entire system which makes it harder to control each part of the UAV separately.
4. It is hard to store the flight data (such as logs, videos and images) in order to use this later for analysis.

These drawbacks lead to the necessity look for another more modular system to eliminate these problems and also to have more freedom to control the system.

4.3 System Architecture Using Odroid U3+and Robotic Operating System (ROS)

The system architecture of the second on-board system consists of several components. Figure 4.11 shows the entire hardware system. The onboard system consists of the same hardware used in the system architecture in Chapter 3 for data collection and off-board analysis and mapping and the system architecture for on-board decision making and action using the Raspberry Pi 2B as shown in Figure 4.10. The system hardware comprises a 3DR-IRIS UAV frame, an AC2830-358 850 KV motor and 10x4.7 propellers, a Pixhawk Autopilot, a GPS Compass Module, a 4 in 1 ESC/Power Module, a 5000 mAh 3S 30C Lipo Pack Battery, a FrSky-DF Radio Control (Tx/Rx), a WiFi connection, a UBEC, a GSC, an ultrasonic sensor (HC-SR04), a HD Webcam Logitech C270, a relay (SRD-05VDC), a spraying pump and a liquid tank, an Odroid U3 and a micro Arduino. The HC-SR04 ultrasonic sensor is controlled by a micro Arduino which is connected via a USB cable to the Odroid U3. The micro Arduino has C++ code to run the ultrasonic measurements (see Appendix B.10).

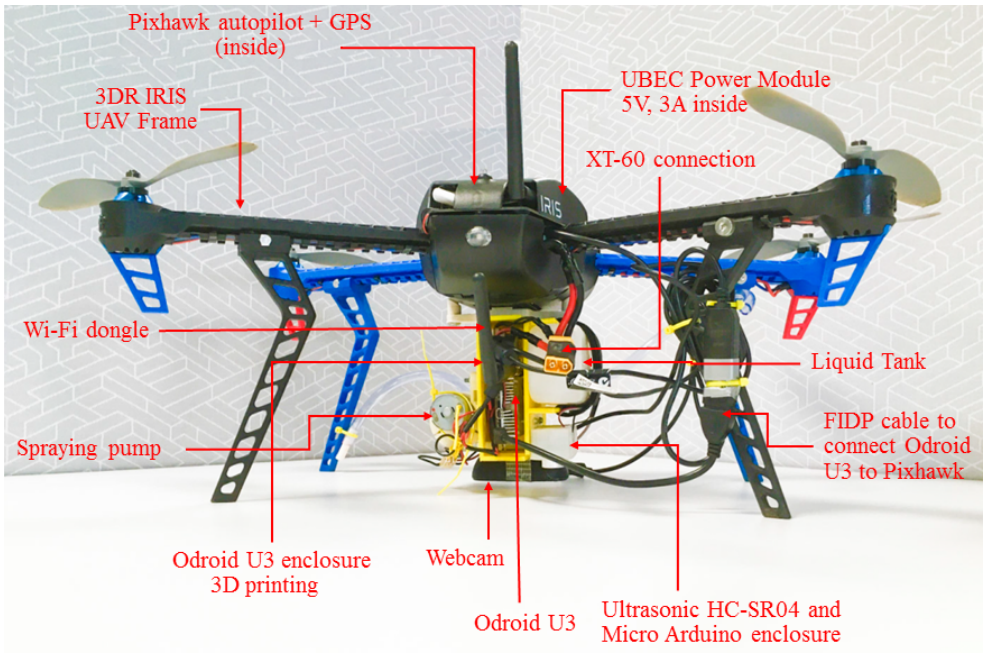


Figure 4.10: On-board system for on-board decision making using an Odroid U3 and ROS.

The Odroid U3 receives the measurements through the USB cable and uses these measurements to control the system through a node in ROS. The micro Arduino is also responsible for controlling both the ultrasonic sensor and the relay which controls the motor for the

spraying system. The Pixhawk is connected to the Odroid U3 through a serial cable called an FIDP cable.

4.3.1 Odroid U3

The Odroid U3 is a powerful Linux single board microcomputer used for on-board decision making and action. The Odroid U3 is faster than the Raspberry Pi 2B (2 GB VS 1GB) and allows near real time image processing (see Appendix A.4). The Odroid U3 microcomputer can be run using different software including Android and Linux. The board is priced between $70–$95 AUD [73] (July 2016).

4.3.2 Micro Arduino

A micro Arduino is a microcontroller used for building digital devices. The micro Arduino provide sets of digital and analog input/output pins to interface to various expansion boards or other circuits. The micro Arduino company provides an "Integrated Development Environment (IDE)" based on the C++ programming language [74].

4.3.3 Electrical Integration

4.3.3.1- Pixhawk – Odroid U3 Serial Interface

The Pixhawk is connected to the Odroid U3 using an FTDI cable as shown in Figure 4.12. A 6-Postion DF13 plug is soldered to the FTDI cable. The benefit of using the FTDI cable is to make the connection between the microcomputer and the autopilot faster than the serial connection with the GPIO.

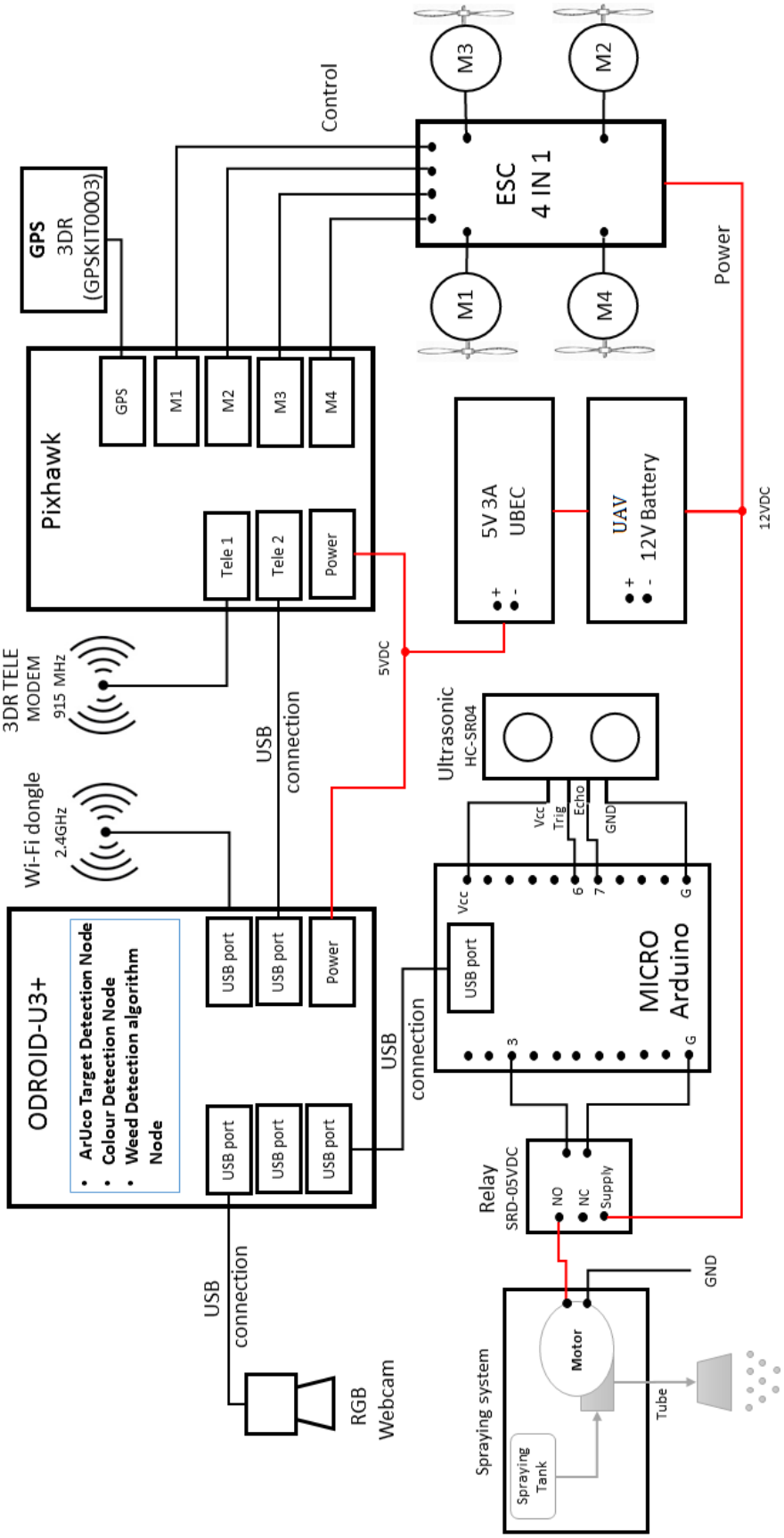


Figure 4.11: System architecture for on-board decision making using an Odroid U3 and ROS.

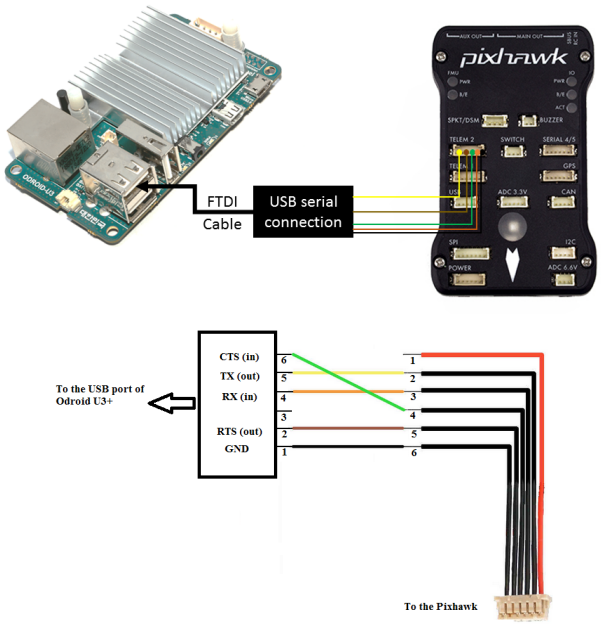


Figure 4.12: Pixhawk Connection to Odroid U3+ through USB serial.

4.3.3.2- Ultrasonic Sensor & Relay – Micro Arduino Serial Interface

Figure 4.13 shows the physical connection of the ultrasonic sensor (HC-SR04) to the micro Arduino and also the connection of the motor to the relay (SRD-05VDC) to micro Arduino. The micro Arduino uses C++ code (see Appendix B.10) which controls both the motor and the ultrasonic sensor. The micro Arduino is powered and connected to the Odroid U3 through a USB connection. A ROS node was also created for the ultrasonic module to receive all the measurement data from the ultrasonic sensor and to check the UAV flight height.

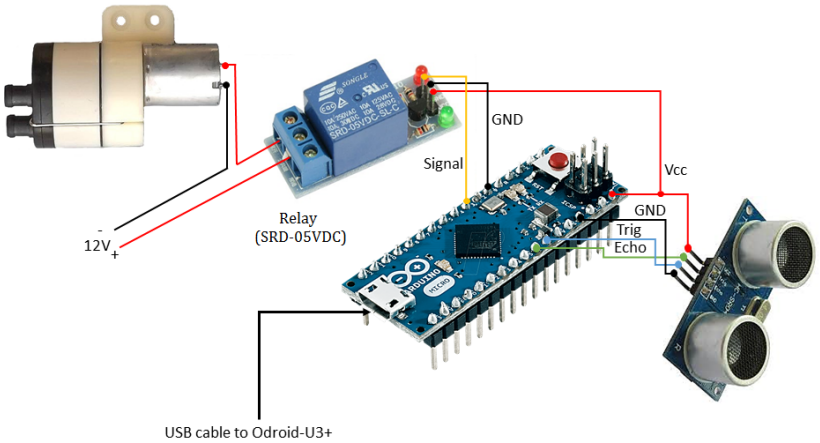


Figure 4.13: Interface between ultrasonic sensor (HC-SR04), relay (SRD-05VDC) and micro Arduino.

4.3.4 Detection and On-board Decision Making Approach

The OODA loop described in Chapter 2 is a closed loop control method used as a framework for decision making [60, 61]. Figure 4.14 shows the flowchart system in the concept of a OODA loop for on-board decision making developed in this work. The observation is based on using the sensors such as ultrasonic sensors and a camera. The ground station receives messages from the on-board computer to check the status of the mission while the UAV is flying. After sending the command to start the mission, the different ROS nodes are executed as shown in Figure 4.15. While the UAV is flying, the detection algorithm will be checking if the target is in the frame or not. When the target is detected, the UAV is automatically commanded to fly to the target. The onboard decision making focuses on the decision and the action part of OODA loop. After the UAV reaches the new location (above the target), the action will be for the UAV to descend to a lower height just above the ground (e.g. 45 cm) and to run the spray pump to spray the target. When the spraying is complete, the UAV either go to the next waypoint looking for new targets or fly home and land.

The spraying tank was calculated with the payload for the UAV (3D IRIS) for this system to do spraying task for maximum of two targets. As the experiment in this research for on-board decision making and action based on OODA loop detecting one target in each flight. This system can be modified by using bigger UAV platform and bigger tank for longer flight time and detecting multi-targets.

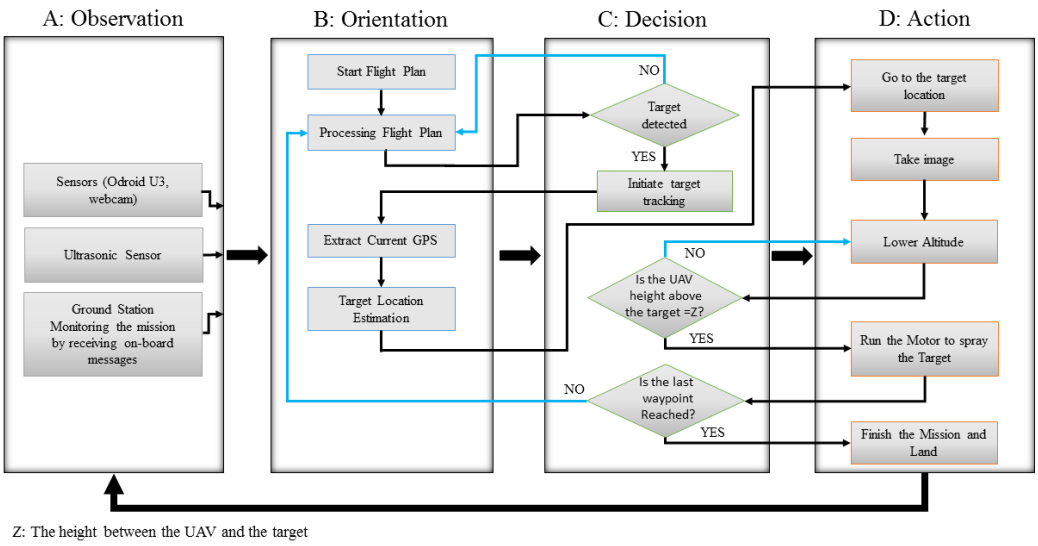


Figure 4.14: OODA loop flowchart for on-board decision making.

4.3.5 Software Design for On-board Decision Making and Action Using Odroid U3$^+$

Two different types of software were used for the on-board decision making and action system running on the Odroid U3. These are Robotic Operating System (ROS) and the JMavSim Simulation.

4.3.4.1- Robotic Operating System (ROS)

ROS is open source software which provides libraries and tools and an easy environment in which to develop robotic applications [75]. ROS provides an operating system service with common functionality, including message transformations between processes, and package management [75]. ROS packages consist of several nodes. A node is basically an executable that when called with *rosrun* or through *roslaunch* will start running. In addition, ROS supports simulations to represent a graph architecture platform to run the processes in nodes that receive, post and multiplex sensor, control, actuator and other messages.

An on-board downward facing camera attached to the 3D IRIS quad-rotor and connected to the Odroid U3$^+$ microcomputer is used for vision based navigation. Several nodes were developed including a navigation node, a camera node, a detection node (to detect features such as ArUco Markers, colours and specific types of weed), a rotation matrix node and a transfer node. All the nodes connect together (see Figure 4.15). The camera is connected to the Odroid U3 via a USB cable and the micro Arduino is connected through another USB cable. The camera node is continuously capturing frames and passing them to the detection node through *OpenCV.cvBrridge*. The detection algorithm is progressively checked so see if the target is in the frame or not. If the target is within frame, the target location in pixels (u,v) will be passed to the transfer node in order to change the pixels (u,v) location to the target location in metres (x, y). In order to have the camera frame in the same direction as the body frame, a rotation matrix node was used to correct the direction.

The ultrasonic sensor (HC-SR04**)** and spraying system are connected to the micro Arduino as described in Figure 4.13. The ultrasonic data is received by the Odroid U3 through via a USB serial connection. The ultrasonic data is used to correct the height (z) of the UAV using a Python node (see Appendix B.8).

Once the navigation node receives the location of the target from the rotation matrix node

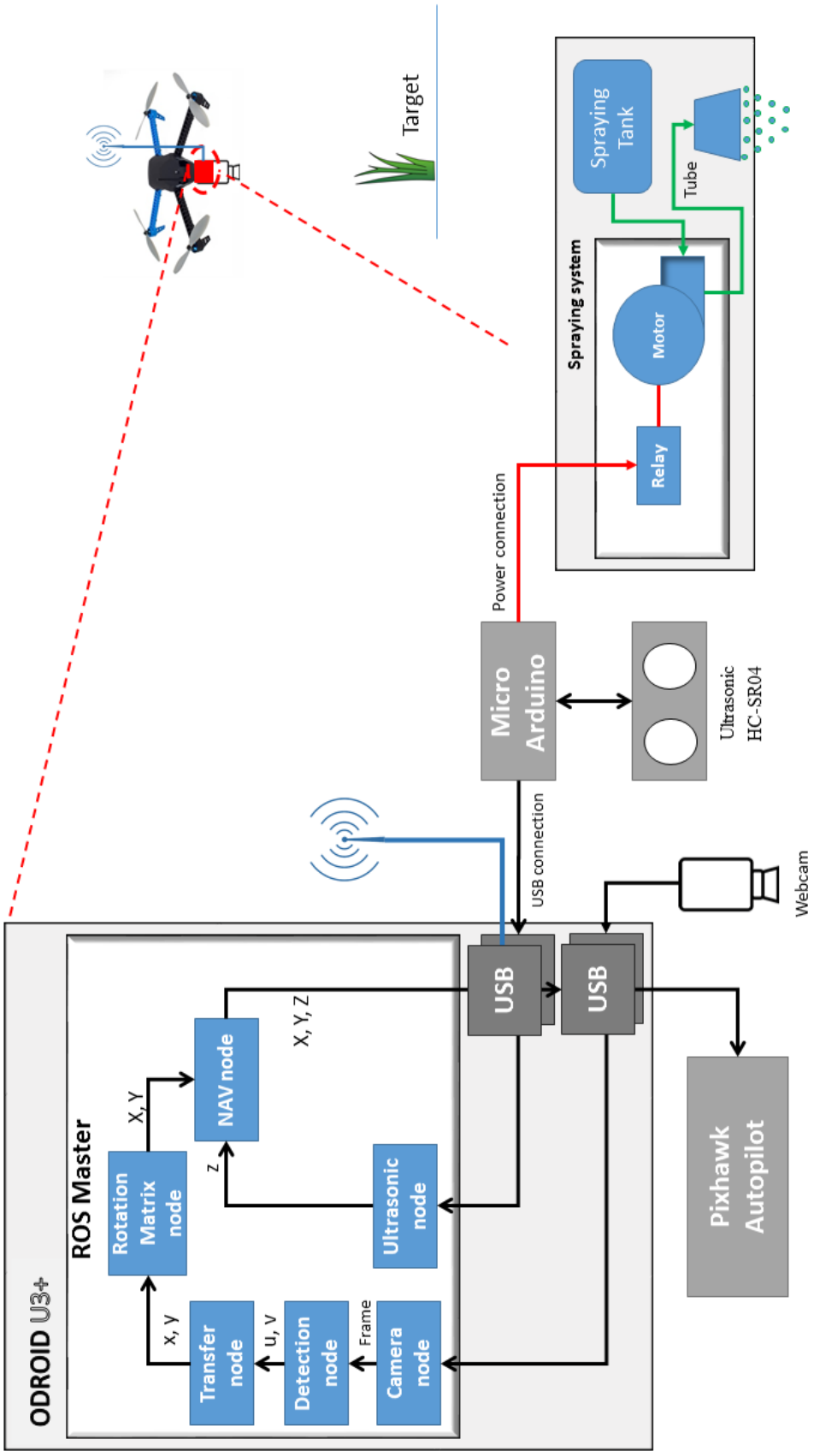


Figure 4.15: ROS nodes for vision based navigation control.

(x,y) and the height from the ultrasonic node (z), the new local position (x,y,z) is sent to the autopilot (Pixhawk) through the navigation node to direct the UAV to go to the new location. When the new location is reached, the navigation node sends a new message to the autopilot to hover above the target at, for example, a height of 45 cm. Once the target reaches that height (e.g. 45 cm), the micro Arduino sends a message to the motor to start spraying on the target for a period of x seconds (e.g. 3 seconds). After completing the task, the UAV resumes flight and continues to fly to the next waypoint or returns to the starting location. Logs of the whole operation can be saved using *rosbage* file and analysed by running it using Rviz (3D Robot Visualizer).

The algorithms for each of the nodes in the system can be found in Appendix B.7, B.8 and B.9.

The control architecture is depicted in Figure 4.16. The function of the vision based navigation is to first estimate the position of the target centre in the inertial frame (u,v). The position controller on-board will control the platform to fly to a position directly above the target, while the PID tuning is done inside the autopilot (Pixhawk). The pseudo code for the position estimation needed for the vision based navigation is as follows:

Algorithm 4.1 A pseudo code for position estimation for the vision based navigation.

- *Find the centre point of the target perspective projection in pixels (u,v).*

$$u=\frac{max(u_i)+min(u_i)}{2},\ v=\frac{max(v_i)+min(v_i)}{2}$$ .... where i= {1,2,3,4}

- *Find the centre of the target for normalised coordinates in the camera frame (x_1, y_2).*
- *Transfer the position of the target from the camera frame to the inertial frame (local position X,Y,Z).*

A) *Camera Model and Pixel Distance*

The camera model used in the system is assumed to be fixed to the UAV platform. The camera frame (e.g. u= 640, v=480 pixels) is projected into the image plane as a 2D point with coordinates (u, v) as shown in Figure 4.17. Where u and v represent the coordinates in pixel units of the point in the plane, while x and y represent the distance in metres in the real world after converting the pixel units to metres. The camera has a 60^o field of view (FoV).

The flight height (z) is received from the ROS message (*geometry_msgs.msg /PoseStamp*)

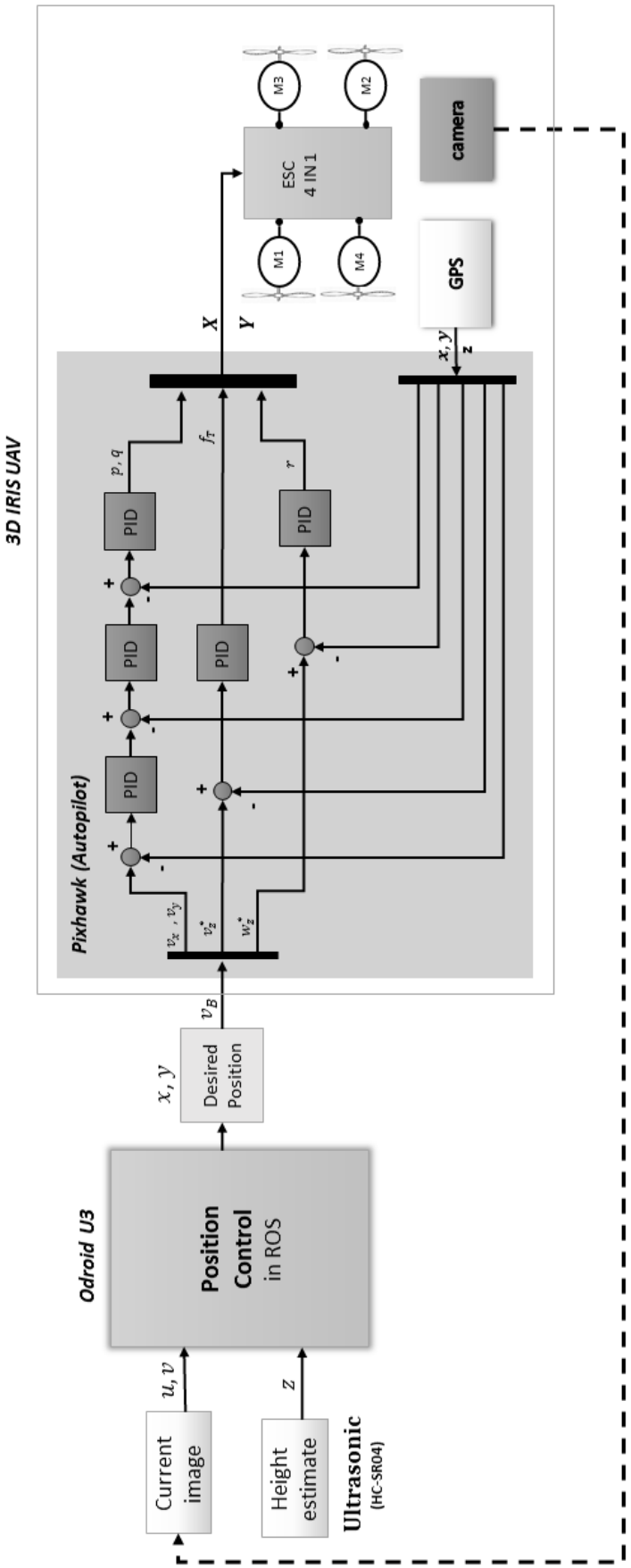


Figure 4.16: Position control architecture for on-board system.

as well as from the ultrasonic node which is connected to navigation node. The following equations are used to calculate the x and y position in the real world.

$$z = current\_pos.z$$

$$x_1 = z * tan\ \frac{\phi}{2} \text{ ..... where is Ø=60}^o \text{ ...(1)}$$

$$y_1 = z * tan\ \frac{\phi}{2} \text{ ..... where is Ø=60}^o \text{ ...(2)}$$

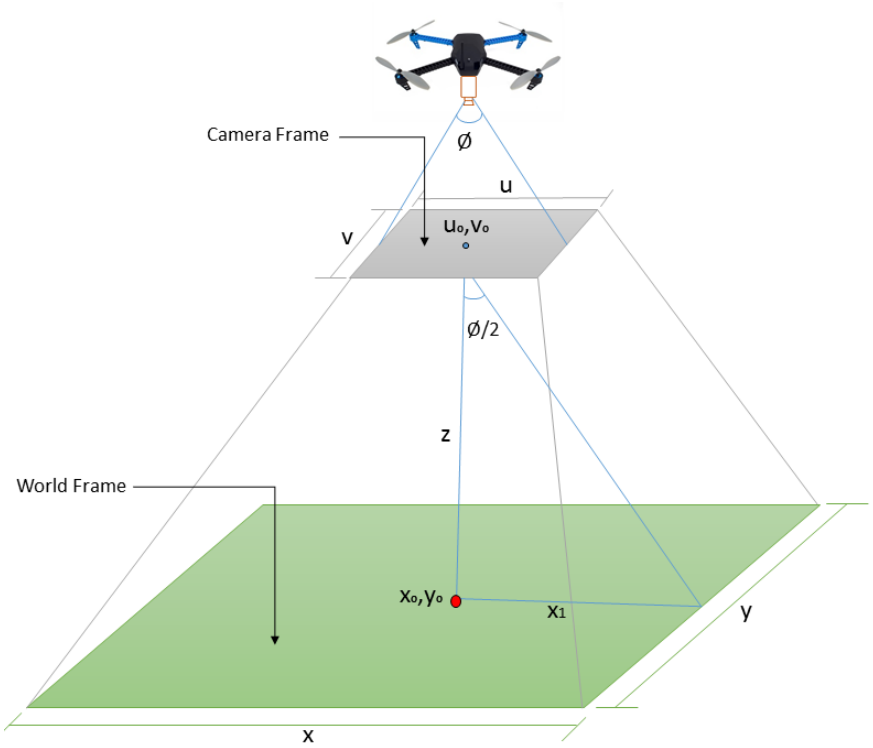


Figure 4.17: Mapping the image frame from the camera to the world frame

B) *Rotation Matrix*

A rotation matrix is used to rotate the coordinates of points in 2D or 3D. There are different type of rotation matrices, however, a basic rotation was applied in this system. A fundamental rotation is a turnover about one of the axes of a direction framework every time. The accompanying three fundamental turnover matrices rotate vectors by an angle ϕ about the x, y, or z axes, in three dimensions [76].

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}$$

$$R_y(\phi) = \begin{bmatrix} cos\phi & 0 & sin\phi \\ 0 & 1 & 0 \\ -sin\phi & 0 & cos\phi \end{bmatrix}$$

$$R_z(\phi) = \begin{bmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The main reason for using a rotation matrix is to align the body frame of the UAV, the camera frame and the world frame. The relationship between the camera velocity v_B and the quadcopter velocity in the body frame v_A is shown in Figure 4.18 and can be established using the basic rotation matrix from the camera frame to the body frame.

$$v_A = R_B^A . v_B$$

The transformation of the position of the target from the camera frame to the inertial frame is as follows:

$${}^{c}X_{target} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_C = {}^{c}X_{UAV} + R_C^A . R_B^A \begin{bmatrix} x \\ y \\ z \end{bmatrix}_B$$

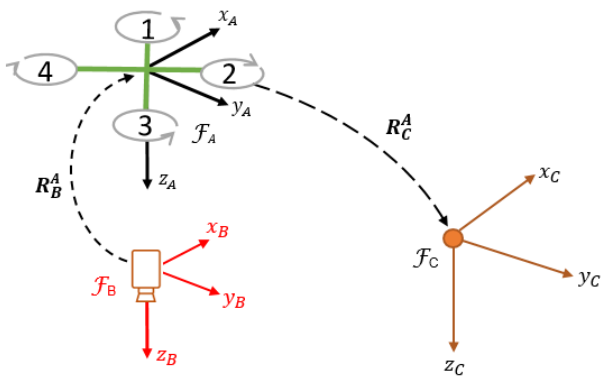


Figure 4.18: Rotation matrix for the camera frame to the body frame

Assuming that the camera azimuth angle and the elevation angle, which represent the orientation of the camera frame with respect to the body frame are small and of an order of magnitude less than the body displacement from the object, the transition matrix R_B^A can be simplified to an identity matrix $_C$ [77, 78]. This was implemented as a node in ROS as shown in Appendix B.9.

C) *Interfacing the Onboard System with the GCS with ROS*

In order to connect the the onboard computer to the ground station, a ssh protocol (*ssh username@hostname*) is used. The *hostname* for the system was changed to *image-processing* for convenience. An example of how to connect the GCS to the on-board microcomputer Odroid U3 is shown in Figure 4.19.

[bilal@sony|~]$ *ssh odroid@image-processing*

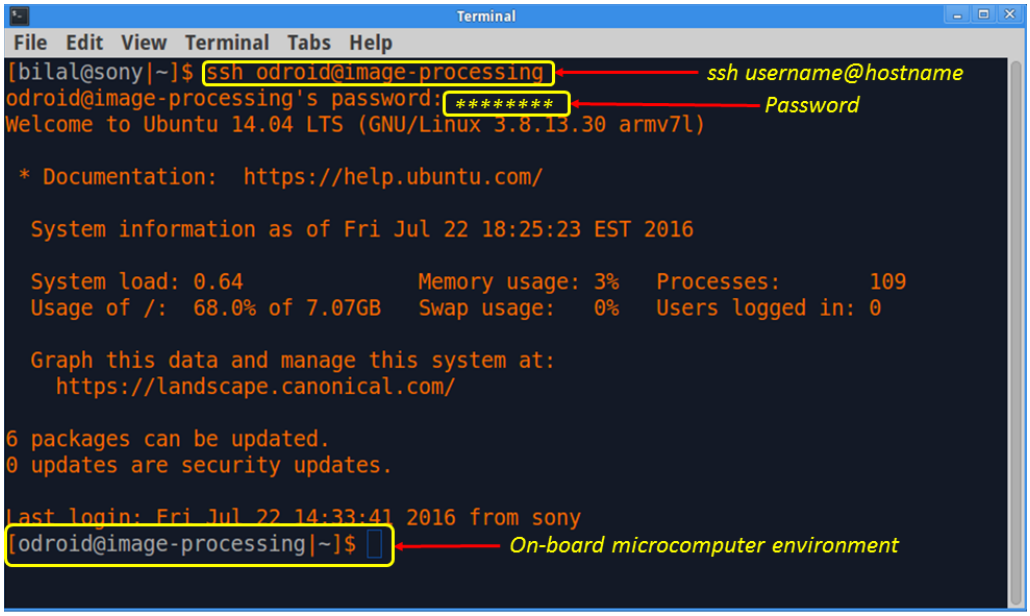


Figure 4.19: Example of connecting the GCS to the Odroid U3.

To enable publishing topics, *roscore* should be started on the onboard computer through the ground station connected via WiFi. As soon as the *roscore* is running, the nodes can run too.

[image-processing@odroid|~]$ *screen*

[image-processing@odroid|~]$ *roscore*

The next step is to choose the launch file to start running the nodes onboard the UAV. The launch file can be found in Appendix B.3.

C) *JMavSim Simulation*

The environment chosen to develop the navigation code for test flight is JMavSim simulation which is a simulation environment that is supported by ROS for simulation in the loop. The following commands are needed to run the simulation:

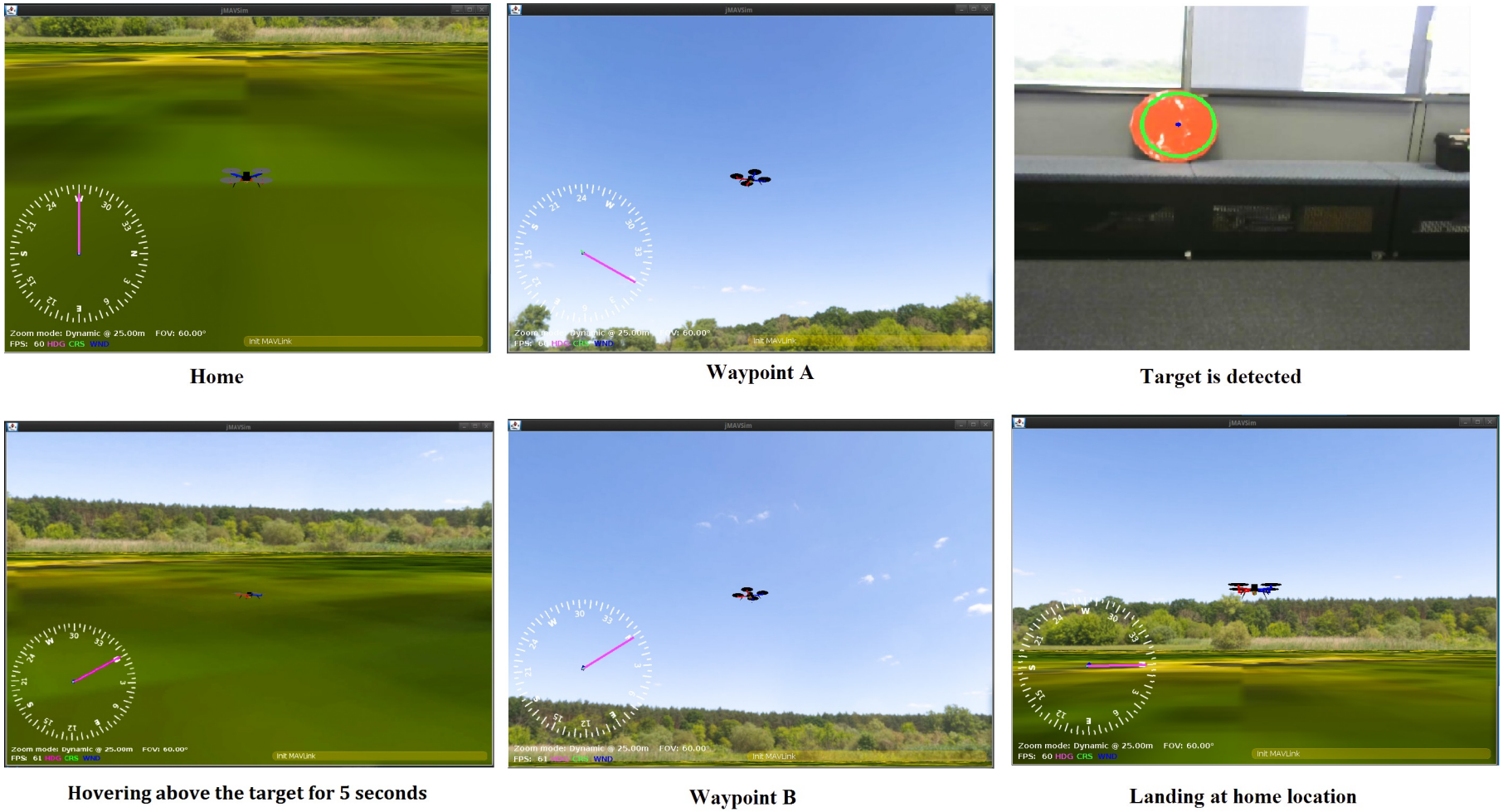


Figure 4.20: Hardware in the Loop, Simulation Flight Test With Red Target Detection.

[bilal@sony|~]$ *roscore*

[bilal@sony|~]$ *roslaunch ~/catkin_ws/launch/px4.launch*

[bilal@sony|~]$ *cd ~/px4/src/Firmware/*

[bilal@sony|~]$ *make posix_sitl_default jmavsim*

[bilal@sony|~]$ *roslaunch ~/catkin_ws/launch/spray_nav spray.launch*

[bilal@sony|~]$ *roslaunch ~/catkin_ws/launch/red_color red color.launch*

After running the commands above, the navigation code can be run to simulate a flight test to detect a specific target. Figure 4.20 shows a UAV with on-board decision making in the loop simulation at different stages. For example, the UAV mission may consist of flying between two points A and B to detect a red circle target. While the UAV is flying, the image processing is continually running to detect the red circle target. When the red circle is detected, the UAV is commanded to fly to the new location (target location) and hover on this location at a height of 45 cm for 5 seconds (see Figure 4.21).

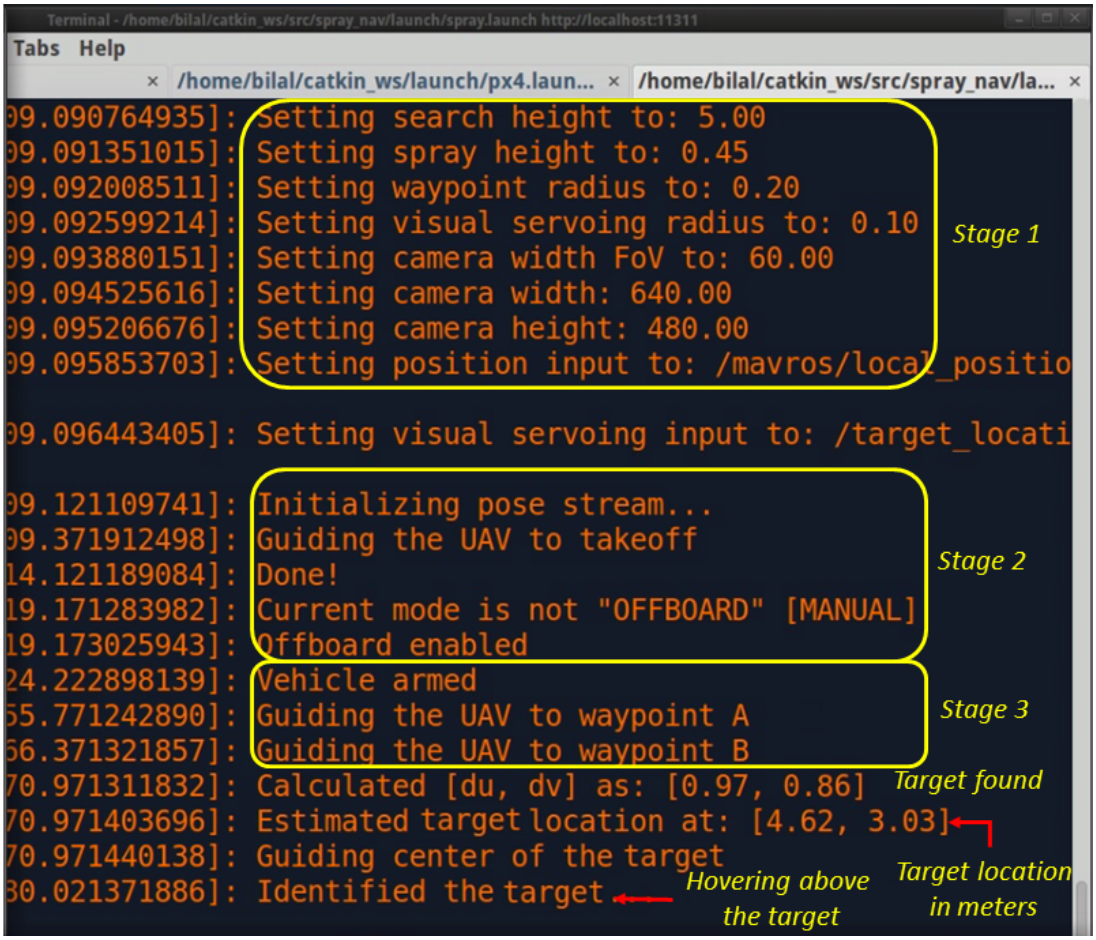


Figure 4.21: Flight description in the simulation.

D) *Flight Test Results Visualisation Using Rviz 3D Robot Visualizer*

Rviz is a 3D Robotic visualizer for displaying sensor data such as from a camera, GPS, ultrasonic sensor etc. and state information for ROS [79]. Using rviz can help to display the actual flight test data and check the flight path and correct any possible errors in the ROS nodes by tuning the Pixhawk parameters if there is an error. The combination of the simulation in Jmavsim (Figure 4.20) before the flight and the rviz (Figure 4.22) after the flight, helps fine the tune the on-board decision making and reduce the errors in the system. The flight test data is saved as baglogs using the following command:

[image-processing@odroid|~]$ *cd ~/baglogs*

[image-processing@odroid|baglogs]$ *rosbag record -a*

The following commands are used to visualise the recorded data:

[bilal@sony|~]$ *roscore*

[bilal@sony|~]$ *rosparam set use_sim_time true*

[bilal@sony|~]$ *rosbag play ~/baglogs/* [the name of the logs].*bag*

The video link below shows a visualisation of the data using the Rviz 3D Robot Visualizer:

https://www.youtube.com/watch?v=lKngX2N2lMs

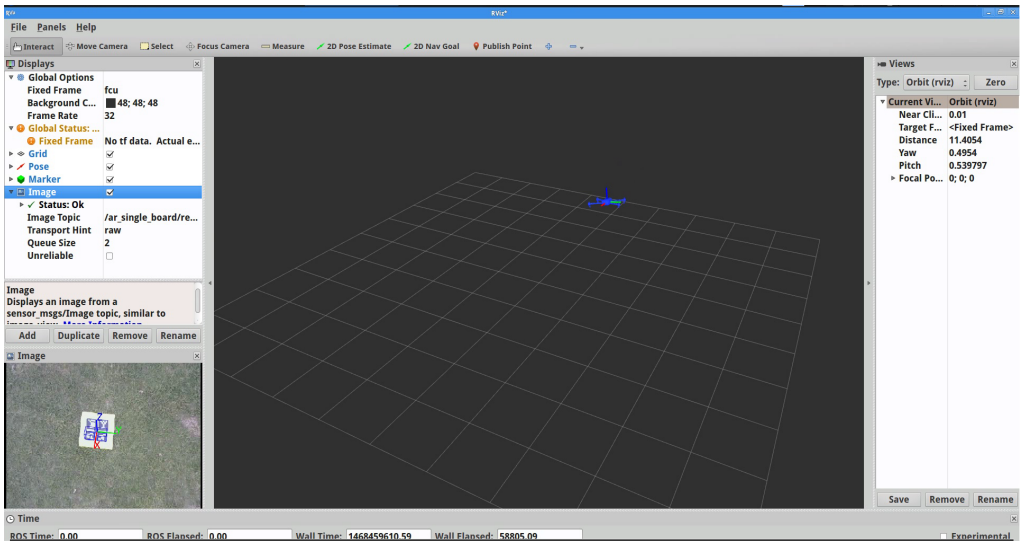


Figure 4.22: Rviz 3D visualizer for displaying the flight data after the actual flight test.

E) *Weed Detection Method with ROS*

The detection algorithm for the invasive spear thistle weed used the same threshold selection method that was explained in Chapter 3 (section 3.4.3). The code was modified into a Python ROS script from Matlab. ROS cv_Bridge is used to pass the image frame to the weed detection node. Figure 4.23 shows the weed detection algorithm test before applying it on-board the UAV with the weed detection method processing frames through the ROS node. There are several limitations to the implementation of this algorithm on-board for image processing near real time such as seasonal variations and change in solar incidence angle. An outline of the image processing node can be found in Appendix B.6.

Figure 4.23: Testing weed detection through ROS.

4.4 Summary

This chapter introduced two possible system architectures for an on-board system. The first system architecture uses a Raspberry Pi 2B microprocessor. A detailed description of the Raspberry Pi 2B on-board decision making including the platform and the subsystem was presented. The disadvantages of using Raspberry Pi 2B system in relation to its slow processing speed and only using a single script to control all actions lead to the development of a second system.

The second system architecture introduced a framework of an Odroid U3 with a ROS operating system. A detailed description of the platform and the different subsystems was presented. The electrical integration (hardware system) and ground station control (software system) for this system was also detailed. An additional set of software designs were introduced for the on-board decision making and autonomous action using the Odroid U3.

The next chapter, Chapter 5 describes the development of the weed detection and mapping algorithm after collecting data from the UAV for off-board decision making.

CHAPTER 5

Weed Mapping and Off-board Decision Making

5.1 Overview

The aim of this research is to investigate and develop a system for off-board and onboard decision making. Chapter 2 discussed the literature review and the materials and theoretical methods applied in the research such as OODA loop theory and vision-based control.

Chapter 3 described the system architecture for off-board analysis including hardware and software. The chapter details the application of the system architecture for data collection and off-board analysis and mapping for invasive weeds.

Chapter 4 described the design of the system architecture for on-board decision making and action. The chapter details the application of the system architecture to on-board decision making and action utilising the concept of an OODA loop.

This chapter presents the specific application of the UAV platform, sensor payload and post processing pipeline customised to detect and map invasive spear thistle weeds. A description of the invasive weed type, study side, UAV flight imagery and GPS mapping and weed detection algorithm are presented. The results and discussion are found in section 5.5.

5.2 Spear thistle invasive weed

Spear thistle was chosen as a target weed to demonstrate the system capabilities. Spear thistle has dark green leaves and purple flower heads (Figure 5.1). The alternative names are Black Thistle, Bull Thistle, Scotch Thistle and the scientific name is *Cirisium vulgare*. This weed belongs to the Asteraceae family. The flowers heads are 3–5 cm long and the leaves are typically be 45 cm long; this weed grows in Australia from spring to autumn [80, 81]. The seeds are smooth and typically 3–5 mm long and are gray in colour and longitudinal darker

markings. The weed is spread by movement of seeds, via wind, water or mud [80, 81]. The spear thistle weed is widely distributed in the southern and eastern parts of Australia. It is very common in south-eastern Queensland, and the southern and eastern parts of New South Wales, Victoria, and Tasmania. It is also common in the south-western parts of Western Australia and the south-eastern parts of South Australia [80, 81].

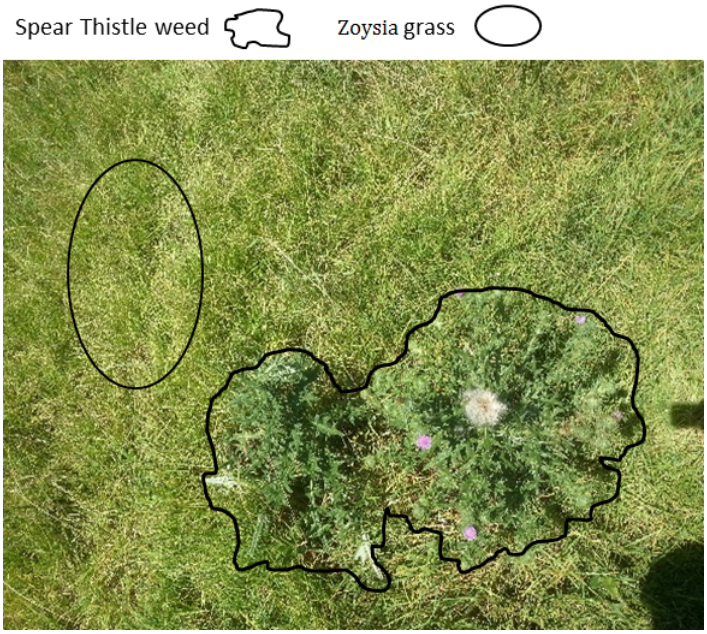


Figure 5.1: Spear thistle weed and zoysia grass.

5.3 Study side

Remotely sensed images were taken between December 21, 2015 and February, 22, 2016 on a field located at Christmas Creek (Queensland, Australia, coordinates 28^o12'19.1"S, 153^o00'08.5"E). The test flights were authorised by a written agreement letter between the farm owner and Queensland University of Technology (QUT). The field was mainly covered with zoysia grass but naturally infested with spear thistle weed. The zoysia grass was similar in size across the field, however, the spear thistle had different distributions in size and in many cases was bigger than the zoysia grass (Figure 5.2).

Figure 5.2: Aerial image for the experimental field including: zoysia grass, invasive weed and dead grass.

An experimental plot of 100x36 metres was delimited within the weed field to perform the flights (Figure 5.3A). The GPS coordinates of each corner of the flight area were collected to assist in determining the waypoints needed to plan the UAV flight, image overlap and weed geo-location.

5.4 UAV flight imagery

The UAV was flown at 5, 7 and 15 m altitude above ground level. The images were taken at a rate of one image/sec and a UAV velocity of 1.5 m/sec was selected to cover the whole experimental field with a 40% imagery forward and lateral overlap by using a single battery within a period of 15 minutes. Figure 5.3A illustrates the flight test location, showing the sector where images were taken (blue), complete UAV flight plan (red) and the area covered by the imagery (yellow). Figure 5.3B shows the UAV in flight above the experimental field.

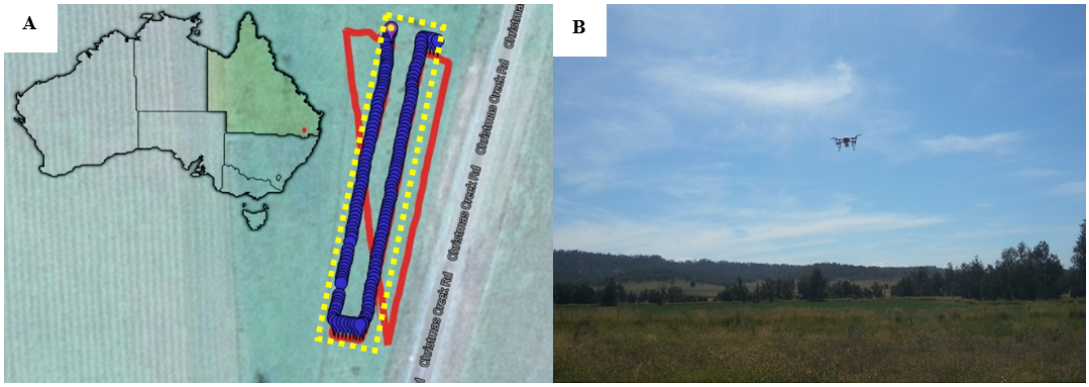


Figure 5.3: **A**: UAV Section where UAV aerial images were taken, image location in blue and the UAV flight path in red. **B**: UAV above experimental field.

5.5 Results and discussion

In order to check the sensitivity and selectivity of the invasive spear thistle weed detection algorithm, multiple images were collected at 5, 7 and 15 m above the ground level (AGL). Figures 5.5A and 5.6A show examples of the original image at 5 and 15 m AGL, respectively, Figures 5.5B and 5.6B illustrate the image after applying the detection algorithm, and Figures 5.5C and 5.6C mark the weed location corresponding to its ground coordinates.

Despite the challenges, the classification algorithm managed to provide classification results with 95% sensitivity and 98% selectivity at 5 m, 90% sensitivity and 94.5% selectivity at 7 m and 80% sensitivity and 85% selectivity at 15 m AGL, as shown in Table 5.1. The sensitivity is the ability of the algorithm to identify and detect the true positive target while the selectivity is the capability of the algorithm to filter out the false negatives for detection targets. Furthermore, the system utilises false positive rates and false negative rates to achieve accurate results. False positives are when a classification indicates presence, however the weeds are absent, and false negatives are the point at which a classification demonstrates the weeds are absent, but are truly present.

Table 5.1: Algorithm accuracy at three different altitudes.

Altitude (m)	No. Of images	Camera Resolution Ground sampling Distance (GSD)	Sensitivity (%)	Selectivity (%)
5	20	1pixel=0.1625 cm	95	98
7	20	1pixel=0.2275 cm	90	94.5
15	20	1pixel=0.4875 cm	80	85

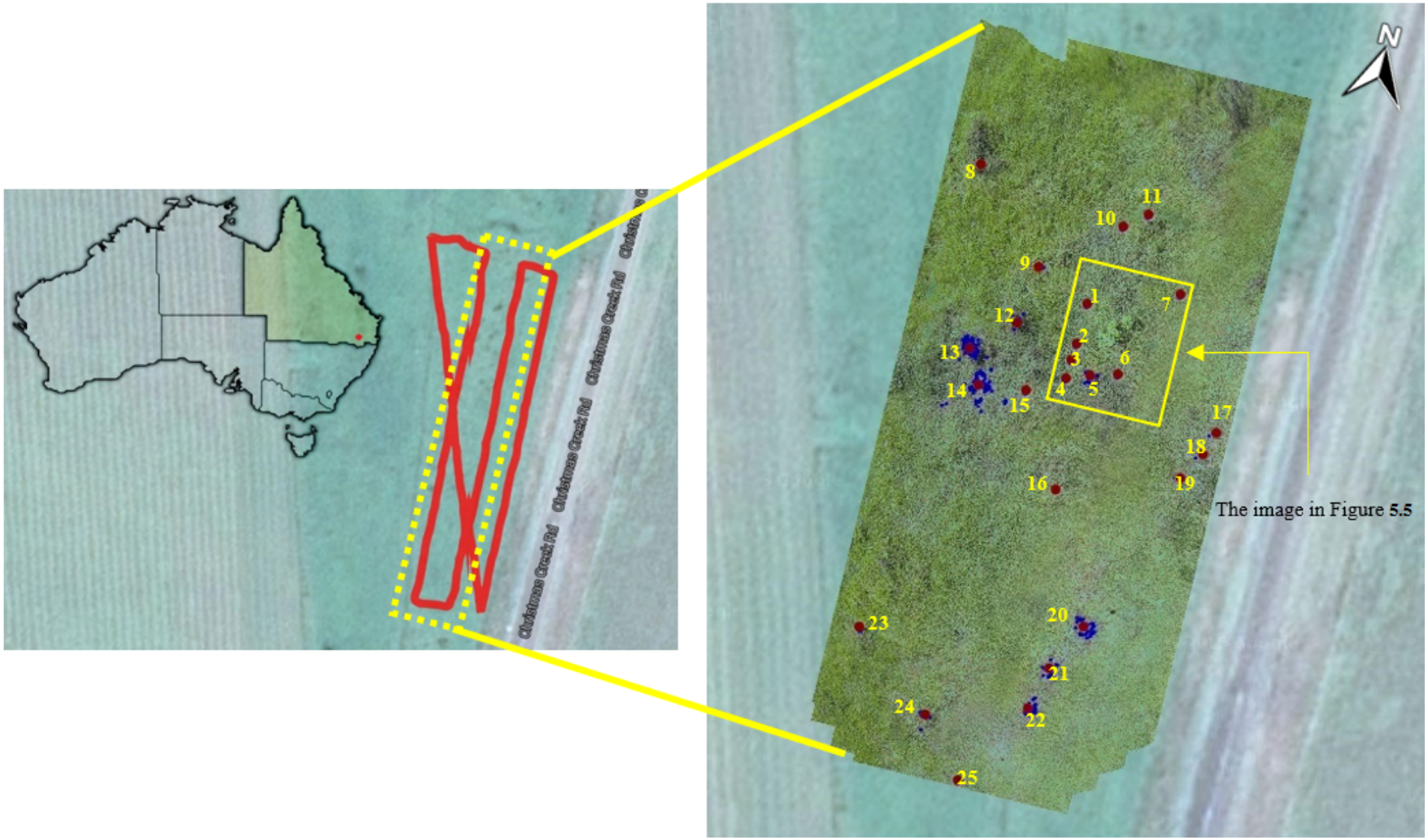


Figure 5.4: Spear thistle weeds mapping with GPS coordinate at 5 metres above ground level.

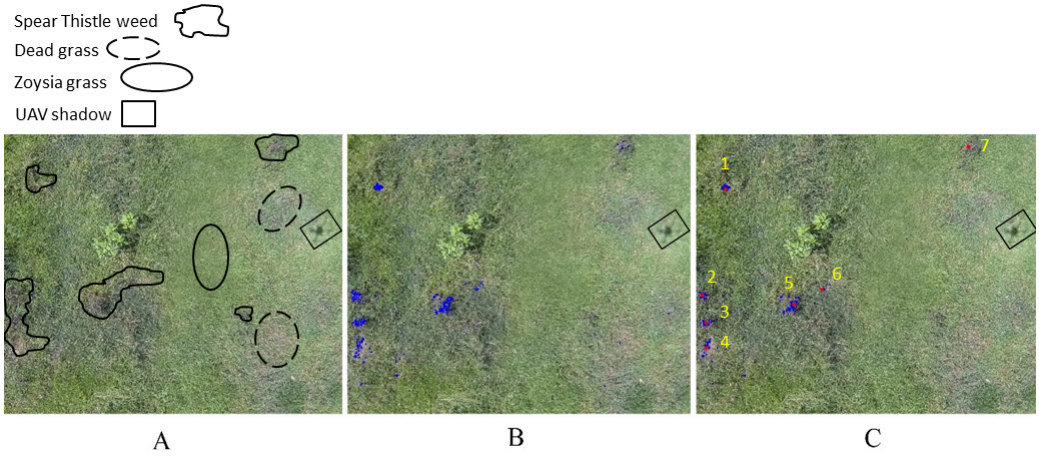


Figure 5.5: Spear Thistle weeds before and after detection at 5 metres above ground level.

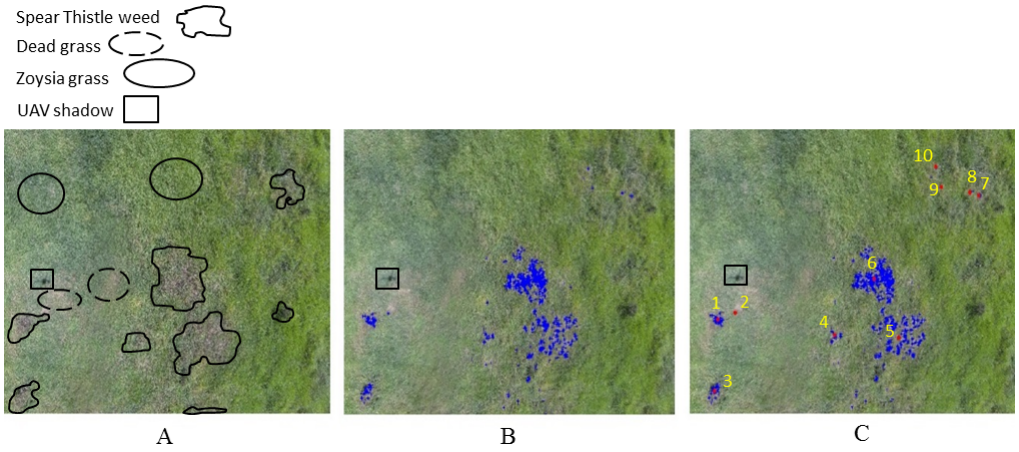


Figure 5.6: Spear thistle weeds before and after detection at 15 metres above ground level.

Figure 5.4 illustrates the stitching of imagery and mapping of the spear thistle weeds in step 21 in Chapter 3 section 3.3. The blue dots represent the spear thistle weeds and the red dots represent the GPS coordinates for the weeds as shown in Table 5.2.

Table 5.2: GPS latitude and longitude for detected weeds.

No	Latitude	Longitude
1	-28.205908014	153.0035909
2	-28.205909646	153.0035909
3	-28.205910226	153.0035910
4	-28.205910779	153.0035922
5	-28.205910503	153.0035896
6	-28.205910116	153.0035890
7	-28.205907765	153.0035884
8	-28.205915248	153.0035980
.	.	.
.	.	.
.	.	.
21	-28.205922498	153.0036027
22	-28.205922551	153.0036034
23	-28.205903546	153.0035949
24	-28.205935348	153.0036847
25	-28.20593876	153.0036410

5.6 Summary

This chapter described the application of the system architecture for data collection and off-board analysis and mapping discussed in Chapter 3, for the purpose of generating invasive spear thistle weed maps. The task was complex due to the similarities in spectral properties and general appearance of invasive spear thistle weeds and zoysia grass at two stages of growth and with difficulties created by variability and changing conditions over time in a natural environment. The algorithm developed for invasive spear thistle weed detection is effective at identifying weeds. Results demonstrated that the system is capable of target detection to 95% sensitivity and 98% selectivity at 5 m above ground level, 90% sensitivity and 94.5% selectivity at 7 m AGL and 80% sensitivity and 85% selectivity at 15 m AGL, with precise GPS mapping. This, however, highly depends on seasonal variability, stages of growth and camera resolution. A different camera should be used to cover a large area in order to increase the flight height (e.g. to 30 or 60 metres). The GPS tabulated locations of weed detected and the weed map provide useful information that can be used in decision-making systems to calculate herbicide requirements and estimate the overall cost of weed management operations.

The next chapter will involve the application of on-board system architecture using vision

based navigation for onboard target detection (such as a marker, a colour or a target invasive weed) and treatment such as spraying on the target or taking a high resolution image.

CHAPTER 6

Autonomous UAV with Vision Based On-board Decision Making

6.1 Overview

The main objective of vision based navigation in robotics (ground or aerial robots) is to control a robot to perform a predefined task using visual feedback [12, 13]. The technique applies vision feedback extracted from the camera. Chapter 4 introduced the system architecture design for the on-board system using an on-board microcomputer.

This chapter extends on that design and presents autonomous vision based on-board decision making and action. The chapter is divided into three sections starting with a description of detection and on-board decision making with the flowchart within OODA loop theory. The second section presents results of flight tests for the detection of ArUco Markers, colour detection and invasive weed detection. The third section discusses the simulation results and the actual flight test results and a comparison between Root Mean Square Error (RMSE) for the simulation and the actual flight test.

6.2 Test Cases

Several flight missions were considered. The flight mission can be modified through the navigation node by adding or deleting waypoints. The mission for this test was taken to fly between three points: Home, A and B. The microcomputer is connected to the ground-station through a WiFi network in order to send commands through the ROS environment as shown in Figure 4.19. The image processing method is programmed to process a captured frame in order to search for a target of interest. When the target is detected, the UAV will fly to new location and hover above the target at a preset altitude of 45 cm and do an action such as spraying the target with pesticide. The UAV will subsequently climb back to the planned flight

height and fly back to Home and land. Three different cases were considered to demonstrate and validate the on-board decision making system: Arco Markers, colour detection and weed detection and spraying.

6.2.1 ArUco Markers

An ArUco marker is a synthetic square marker composed of a wide black border and an inner binary matrix which makes up its identifier. The on-board camera detects the ''ArUco" markers arranged in a square pattern as shown in Figure 6.1 and transmits their position to the Pixhawk through the navigation node. The camera is connected to the Odroid U3^{+}, which runs a ROS node designed to search for the marker while the UAV is flying and also calculate the marker position and send it in metres to the Pixhawk as local position (x, y) [82] through the navigation node. This type of marker is supported by the OpenCV library [83]. The ROS node for marker detection can be found in Appendix B.4. The pseudo code for position estimation needed for the vision based navigation is as follows:

Algorithm 6.1 Pseudo code to find ArUco Marker.

1. *Find square shapes in the image that has a feature to be markers.*
2. *Analyse the inner codification and show the axes (x,y and z).*
3. *Send the target location in metres to the navigation node in order to send the UAV to the desired target.*

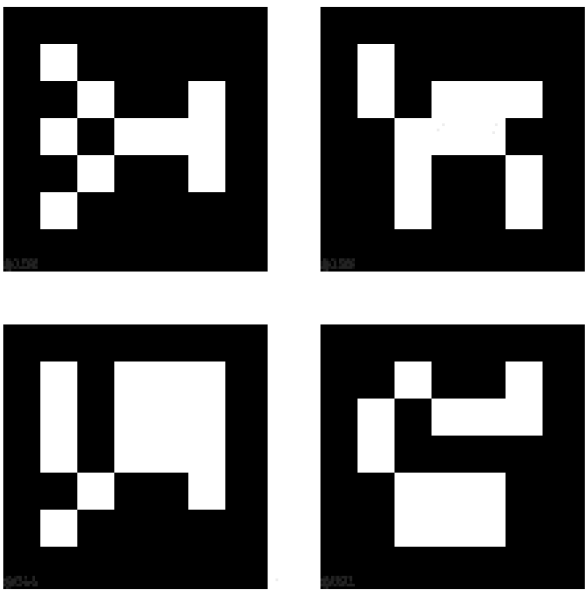


Figure 6.1: ArUco Marker.

ArUco markers are implemented in the first test to ensure the vision based control is working without any issues. The benefits of using the ArUco marker is that the ROS messages (*geometry_msgs.msg /PoseStamp*) are sent directly to the navigation node in metres so that is no need to rotation matrix node. Figure 6.2 shows the UAV hovering above the target at a preset flight height of 45 cm and detecting the target. The ArUco marker detection and navigation was conducted 10 times. The test produced on-board detection results with 99% sensitivity and 100% selectivity.

Figure 6.2: **A**: The marker target is detected. **B**: Detecting target. **C**: Hovering above the target after detection.

6.2.2 Colour Detection

A ROS node for red colour detection was implemented and run on-board the UAV with the help of the on-board computer (Odroid U3^{+}). The position and orientation of the target are

provided through *rosbag* recording data. The algorithm uses image processing using the OpenCV library to scan for the colour red [84]. The camera node publishes image frames through the CV Bridge in ROS [85]. Figure 6.3 illustrates how the images are passed using *OpenCV.cvBrridge* in the ROS library to provide an interface between ROS and OpenCV. A combination of HSV colour and circle detection were also used to consistently search for red circular object to verify the vision based navigation. The pseudo code for position estimation needed for the vision based navigation is as follows:

Algorithm 6.2 Pseudo code for colour detection.

1. *Split each image into R,G,B channels and then convert these to H, S, V channels.*
2. *Threshold the HSV image and keep only the red pixels.*
3. *Normalise each channel and convert to greyscale images.*
4. *Use the Hough transform to detect red circles in the threshold image.*
5. *Find the image feature representation of the centre using the image centre and focal length.*
6. *Send the image feature in pixels to the transfer node in order to convert from pixels to metres.*
7. *Send the target location in metres to the rotation matrix node to orient the UAV body frame in same direction as the world frame.*
8. *Send the target location from the rotation matrix node to the navigation node to direct the UAV to the desired target.*

An outline of the colour detection node can be found in Appendix B.5.

The test for on-board decision making for colour detection was conducted eight times. Results shows that the UAV flew from home towards point A and then towards B. While moving towards B the UAV detects the target colour and descends and hovers 45 cm above the red circle. Figure 6.4 **A** and 6.4**B** shows the UAV is detecting the red circle, and Figure 6.4**C** shows the UAV hovering above the target and spraying it for 3 seconds. The test provides on-board detection and action results with 96% sensitivity and 99% selectivity.

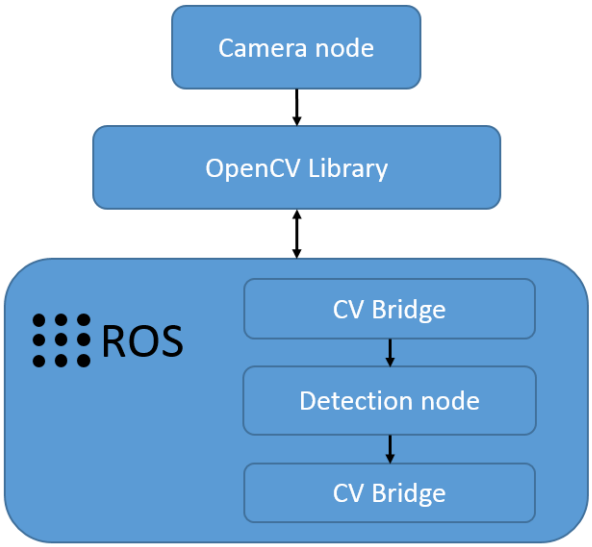


Figure 6.3: Open-CV Bridge to pass the images in ROS.

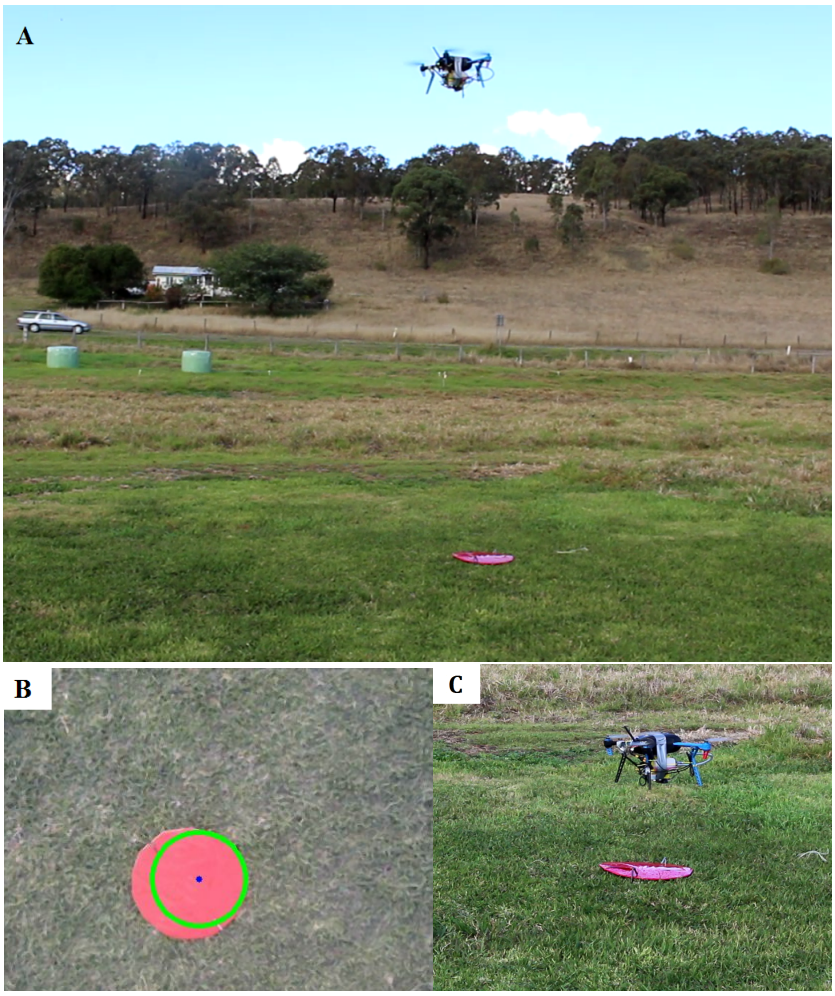


Figure 6.4: **A**: Colour target is detected. **B**: Detecting target. **C:** Hovering above the target at 45 cm.

6.2.3 Weed Detection and Spraying

The weed mapping and off-board decision making system discussed in Chapter 4 mainly focuses on the **Observation** and **Orientation** aspects of the OODA loop. The **Decision** and **Action** aspects of the OODA loop for weed detection are applied in this section. The control system for on-board decision making was presented in Figure 4.15 and the weed detection method described in Chapter 3. Figure 6.5 shows the results for the mission; Figure 6.5**A** shows the UAV above the weed when the weed is detected; Figure 6.5**B** shows the weed detection and Figure 6.5**C** shows the UAV hovering above the weed at 45 cm and spraying on the target. The test was conducted four times and its based on the sensitivity and selectivity of the algorithm. The sensitivity is the ability of the algorithm to identify and detect the true positive target while the selectivity is the capability of the algorithm to filter out the false negatives for detection targets. Furthermore, the system utilises false positive rates and false negative rates to achieve accurate results. False positives are when a classification indicates presence, however the weeds are absent, and false negatives are the point at which a classification demonstrates the weeds are absent, but are truly present. The system is capable of detecting the invasive weed to 33% sensitivity and 67% selectivity. The low sensitivity and selectivity for weed detection is mainly due to the limitations in the algorithm which are related to several reasons such as different soil types, topography and amonut of background vegetation and the sun angle and the season in which the test is performed and thus the growth stage of the weed. Furthermore, the error in the sensitivity and selectivity will also occur based on the camera settings including the ISO and the exposure. The existing method can be modified to ensure the weed can be detected accurately at different times of the day and at different stages of growth by using other methods such as machine learning based on extensive data collection to train the algorithm for more precise weed detection. The link below shows the flight tests.

https://www.youtube.com/watch?v=P8YH9cllrcE

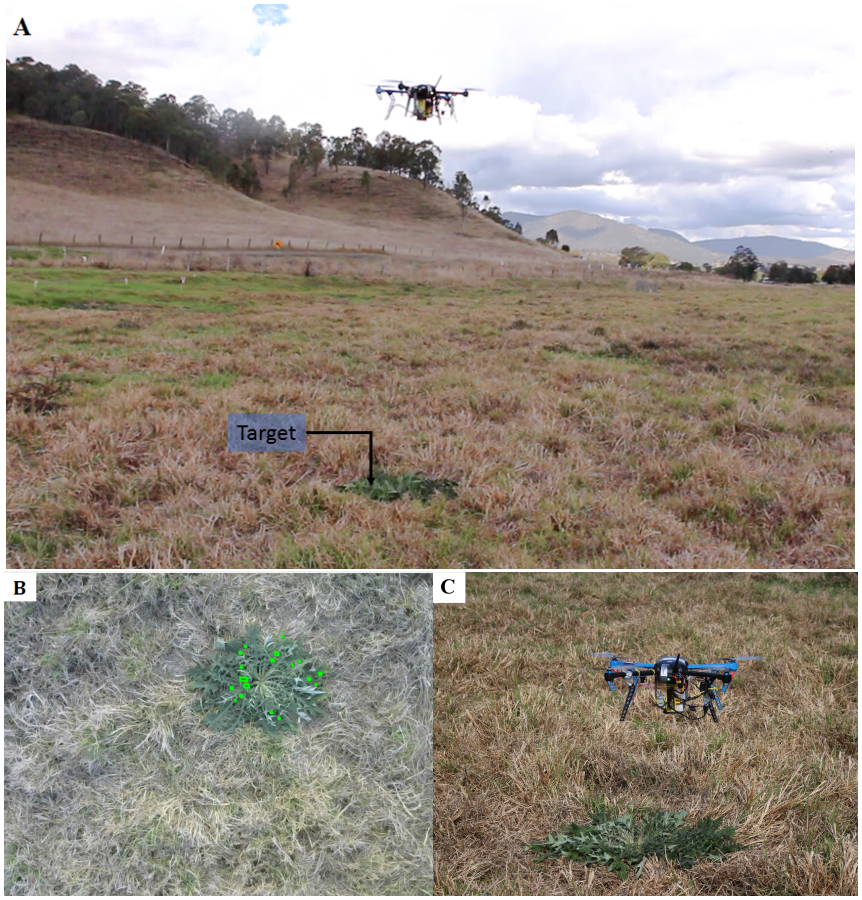
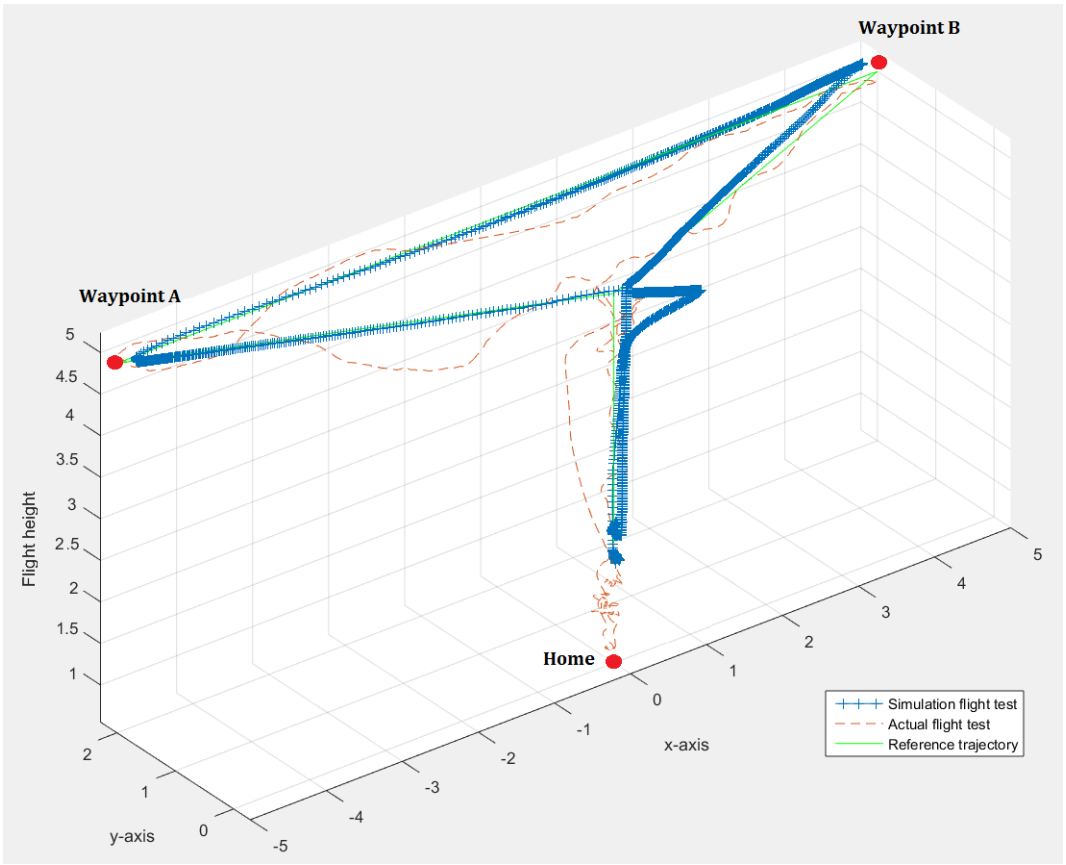


Figure 6.5: **A**: weed target is detected. **B:** Detecting target. **C**: Hovering and spraying on the target.

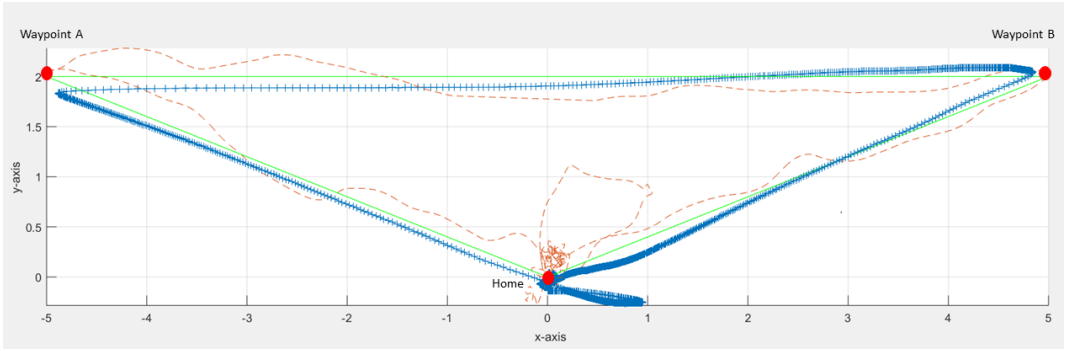
6.3 Simulation and Actual Flight Test Results and Analysis

Numerical experiments were simulated in the proposed model to verify the performance of the vision based navigation for on-board decision making. After recording the flight test data (as baglogs files) from the simulation test and the actual flight test, the MATLAB program is used to draw the flight path in order to compare them and analyse the data from the simulation and the actual flight test.

Figure 6.6 shows the results for the navigation system (no weed detection or action). The results shows the ability of the UAV to track the reference trajectory in relatively high winds (15 km/hour = 8 knots). The quad-copter flight path starts from the initial position (0,0) home and progresses to waypoint A (5,2) then to waypoint B (-5,2) in the (Jmavsim) simulation flight test.



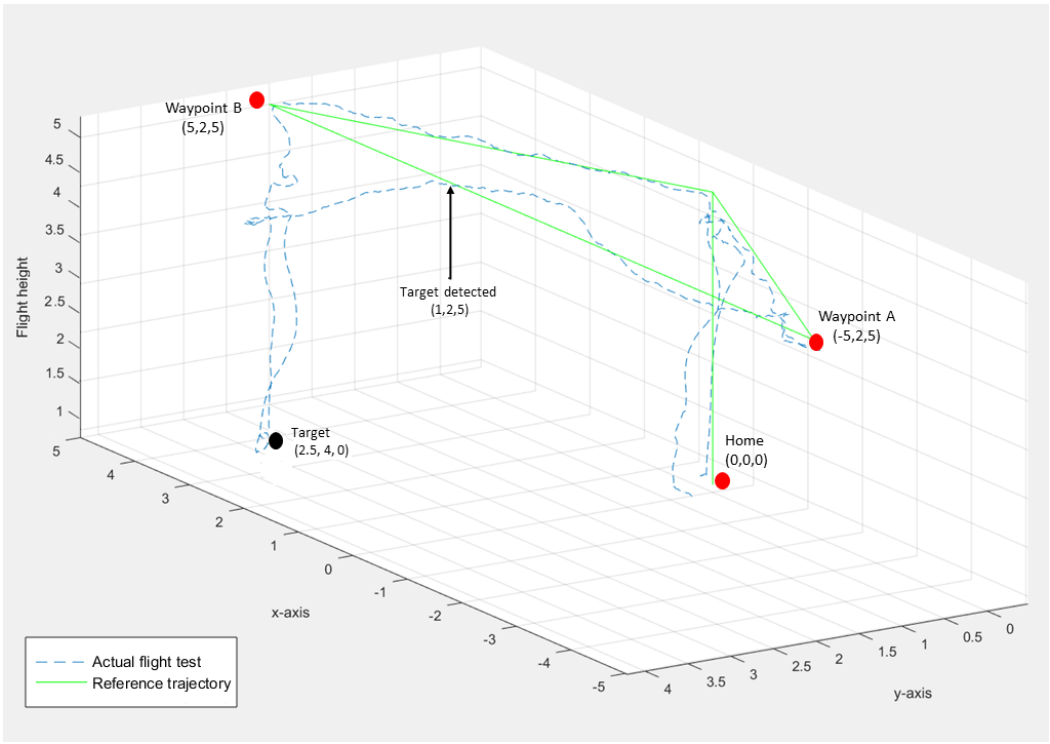
(**A**)



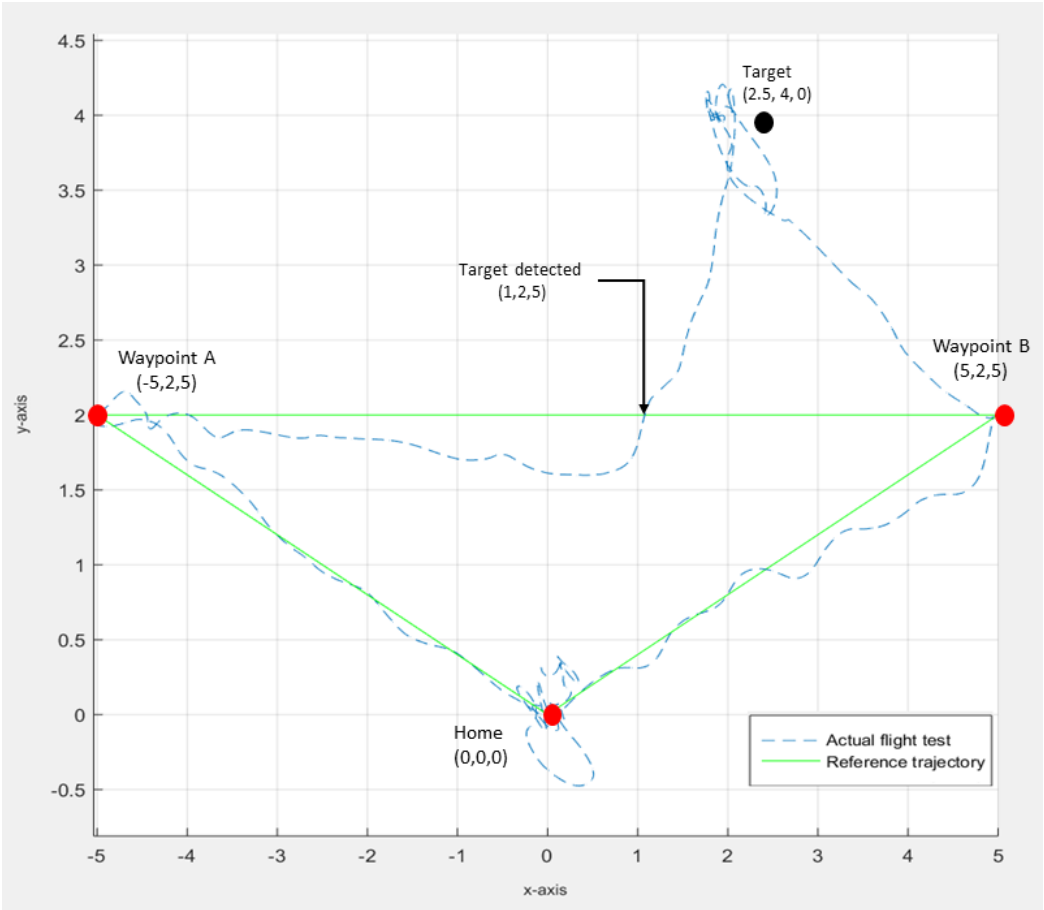
(**B**)

Figure 6.6: Simulation and actual flight test navigation verification, **A**: Simulation and actual flight trajectory, **B**: Top view for simulation and actual flight trajectory.

Figure 6.7 shows vision based navigation results for onboard decision making. Once the target is detected the control transforms image feature (pixels) to the target location in metres and directs the UAV to fly to that new location. The flight test results presented in (Figure 6.7) confirm the previous description about the on-board system for decision making using the concept of OODA loop. The MATLAB code for drawing the flight trajectory can be found in Appendix B.11.



(**A**)



(**B**)

Figure 6.7: **A**: Actual flight trajectory with on-board decision making, **B**: Top view for actual flight trajectory.

6.4 Root Mean Square Error (RMSE)

Root Mean Square Error (RMSE) is used to measure the differences between the sample values and the population values predicted by a model or an estimator of the actual values. The RMSE represents the sample standard of the differences between the predicted values and observed values. The following equations were used to calculate the RMSE for the actual and the simulation flight test. The calculation for RMSE was applied for the x, y and z axes.

$$\text{RMS\_x}=\sqrt{\frac{1}{N}*(x-x^*)} \quad (1)$$ where x is the variable value, x^* is the reference point and N is the number of the points.

$$\text{RMS\_y}=\sqrt{\frac{1}{N}*(y-y^*)} \quad (2)$$ where y is the variable value, y^* is the reference point and N is the number of the points.

$$\text{RMS\_z}=\sqrt{\frac{1}{N}*(z-z^*)} \quad (3)$$ where z is the variable value, z^* is the reference point and N is the number of the points.

Table 6.1: RMSE for simulation flight and actual flight in metres.

RMS	Simulation Flight Test (m)	Actual Flight Test (m)
RMS_x	0.0818	0.4481
RMS_y	0.0796	0.1571
RMS_z	0.0646	0.3404
RMS_Euclidean distance	0.13115	0.5842

Table 6.1 shows the RMSE for the simulation and the actual test flight. The RMSE for the actual test flight is, however, different to the the RMSE for the simulation and that is due to the weather for the outdoor flight test where the wind speed was 15 km/hour (8 knots). It is also much harder to control the light conditions in real world. The RMSE for the actual flight test overall is 58 cm which is small enough to perform the on-board decision making and action.

6.5 Summary

This chapter described the application of the system architecture for on-board vision based navigation and decision making and action. The feedback from the image features enabled the quad-rotor 3D IRIS to move to a desired position based on the OODA loop concept for feedback

control. The vision based navigation system and on-board decision making were illustrated in three types of tests: ArUco Marker detection, colour detection and weed detection. Results demonstrated that the system is capable of detecting ArUco Markers to 99% sensitivity and 100% selectivity at 5 m above the ground level. The system is also capable of detecting a red target to 96% sensitivity and 99% selectivity at the same height during a test flight at 5 metres. The real time on-board detection and action algorithm for invasive weed needs to be improved to achieve better sensitivity and selectivity. The system is capable of detecting the invasive weed to 33% sensitivity and 67% selectivity. This low sensitivity and selectivity for weed detection is mainly due to the limitations in the algorithm in relation to the sun angle and the season in which the test is performed and thus the growth stage of the weed.

This chapter also discussed the RMSE values for the flight simulation and the actual flight. The RMSE for the actual flight test is, however, significantly different to that of the simulation. The main reason for this is that the wind speed during the flight test was 15 km/hour (8 knots). The RMSE for actual flight test is nonetheless only 58 cm which is not problematic for on-board decision making and action.

The next chapter will present the conclusion of the research and further future development as well as the limitations of this research and some considerations for future work.

CHAPTER 7

Conclusions

7.1 Research Summary

The full commercial potential of using UAVs for remote sensing in agricultural applications is yet to be realised. In precision agriculture, for example, UAVs can be developed to assist in several functions including weed detection, early detection of disease outbreaks and on-board decision making. Autonomous on-board detection and action will target the specific location of the weed or pest for pesticide application instead of applying it on the whole field.

The aim of this research was to develop and flight test off-board and on-board systems for precision agriculture using a multi-rotor UAV. The on-board system includes: a UAV, a microcomputer with advanced processing interfaced with a RGB camera and a GPS with an OODA loop framework approach for on-board decision making using computer vision. The system can be used in the field of precision agriculture in order to access the plant health and take necessary action such as spraying pesticide. Firstly, a literature review was conducted on the application of UAVs in remote sensing and precision agriculture based on sensor type and operational efficiency. Secondly, a system architecture design was presented for both off-board and on-board vision based navigation for on-board decision making using computer-vision based context. Thirdly, a weed detection and mapping using the off-board system was described and tested. An UAV with a digital camera was used to capture images of a grass field to generate a map of invasive spear thistle weed in the field. The task was complex due to difficulty in differentiating highly correlated invasive spear thistle weeds and zoysia grass in their spectral properties and general appearance at two stages of growth and the additional difficulties due to seasonal variability and sun angle. The algorithm developed for invasive spear thistle weed detection was found to be effective in identifying weeds and mapping the GPS location for each weed. Lastly, a method for target detection and action using the vision based navigation and on-board decision making was presented based on a set of feature points

on the object to be detected. The feedback of the image features enables the UAV to move to the desired position based on the OODA loop concept.

This work is an active field of interest for on-board decision making with the framework of the OODA loop. A number of additional inputs were also made in the investigation, development and implementation of each of the system components.

7.2 Addressing Research Question

This section elaborates how the investigation addresses the research questions and objectives.

Research Question 1: What are the current limitations of image analysis and classification methods and vision based navigation with electro-optical sensors for on-board decision making?

To answer this research question, several activities and flight tests were undertaken. To achieve on-board decision making using vision based navigation with the concept based on the OODA loop framework, a system architecture was developed (hardware and software) and tested in a real environment at Christmas Creek, Beaudesert in Queensland Australia (see Chapter 3, Chapter 4). The task was divided into two stages.

Data collection and off-board analysis and mapping were conducted in the first stage. The weed detection algorithm for the purpose of generating invasive spear thistle weed maps demonstrated that the off-board system is capable of target detection to 95% sensitivity and 98% selectivity at 5 m above the ground, 90% sensitivity and 94.5% selectivity at 7 m above the ground and 80% sensitivity and 85% selectivity at 15 m above the ground with precise GPS mapping. This is, however, highly dependent upon season variability and stages of plant growth and camera resolution (see Chapter 5).

The task in the second stage involved on-board flight tests for vision based navigation and testing it for three different types of targets (see Chapter 6). The results showed that the on-board system using the ROS operation system is capable of object detection and of closing the OODA loop. Results demonstrated that the on-board system is capable of detecting ArUco Markers to 99% sensitivity and 100% selectivity at 5 m above the ground and of detecting red colour targets to 96% sensitivity and 99% selectivity at the same height. But the real time on-board detection and action algorithm for invasive weed needs to be improved to achieve

better sensitivity and selectivity. The system is capable of detecting the invasive weed to 33% sensitivity and 67% selectivity. This low sensitivity and selectivity for weed detection is mainly due to the limitations in the algorithm, in particular in relation to the sun angle and the seasonal variabilities in the weed. The evaluation of features for leaf classification presents a big challenge and achieving 100% accuracy using computer vision is extremely difficult in a real environment.

Research Question 2: What are the challenges in the practical applications of the off-board and on-board systems in the context of precision agriculture and plant biosecurity where vegetation characteristics, such as texture, colour and shape pose significant challenges for existing image classification algorithms?

To assess performance, the off-board and on-board systems were applied in the context of precision agriculture in two stages. The first stage consisted of developing and testing the off-board system by collecting the data and running the algorithm for detecting and mapping the invasive weed (see section 3.3 and Chapter 3). The main challenges were the difficulty in differentiating highly correlated invasive spear thistle weeds and zoysia grass in their spectral properties and general appearance at two stages of growth and the difficulties due to seasonal variability and sun angle.

The second stage was to develop and flight test the invasive weed detection method on-board the UAV using vision based navigation (see section 4.3.5, Chapter 4). The main challenges here were due to the software speed and processing complexity including applying the weed algorithm on-board with near real time image processing and also the selection and development of the hardware components of the on-board system given the limited payload weight capabilty of the UAV.

7.3 Considerations and Future Work

There are a few items that should be considered when applying the on-board system with the concept of vision based navigation for on-board decision making in precision agriculture applications.

- The communication between the ground station and the on-board system uses WiFi and this limits the UAV to remain within 400 m from GCS.

- The ultrasonic sensor was programmed to hold the UAV at a height of 45 cm while it performs an action (e.g. spraying on the target). To have more control for the UAV, the ultrasonic sensor should be connected directly to the autopilot; however, the functionality of the ultrasonic sensor (HC-SR04) is limited (see A.2). Therefore, the existing ultrasonic sensor would be replaced with another sensor for more functionality in order to control the actual height.

- Another issue for large RMSE error and the difficulties in tracking are due to on-board sensor error also contribute to the sensitivity and selectivity. Future work could focus on evaluating GPS and on-board uncertainties or installing a more precise RTK GPS system.

- The weed detection method is based on specific characteristics of the weed, which can be affected by several factors such as soil types, topography and amount of background vegetation and the sun angle and seasonal variability. Furthermore, the error in the algorithm will also occur based on the camera settings including the ISO and the exposure. The existing method can be modified to ensure the weed can be detected accurately at different times of the day and at different stages of growth. Suggestions to improve the detection method include:

1. Using micro multispectral or micro hyperspectral camera spectral bands of near infrared ranges to detect weed.

2. Using machine learning methods based on extensive data collection to train the algorithm for more precise weed detection.

APPENDIX A

Hardware specifications

A.1 The table below shows the Raspberry Pi 2B microcomputer specifications.

Table A.1: Raspberry Pi 2B specifications.

Raspberry Pi 2 Specifications
SoC Broadcom BCM2836 (CPU, GPU, DSP, SDRAM, and single USB port)
CPU 900 MHz quad-core ARM Cortex A7 (ARMv7 instruction set)
GPU Broadcom VideoCore IV @ 250 MHz
OpenGL ES 2.0 (24 GFLOPS)
1080p30 MPEG-2 and VC-1 decoder (with license)
1080p30 h.264/MPEG-4 AVC high-profile decoder and encoder
Memory 1 GB (shared with GPU)
USB ports 4
Video input 15-pin MIPI camera interface (CSI) connector
Video outputs HDMI, composite video (PAL and NTSC) via 3.5 mm jack
Audio inputs I^2S
Audio outputs Analog via 3.5 mm jack; digital via HDMI and I^2S
Storage MicroSD
Network 10/100 Mbit/s Ethernet
Peripherals 17 GPIO plus specific functions, and HAT ID bus
Weight of 45 grams

A.2 The table below shows the HC-SR04 Ultrasonic Sensor specifications.

Table A.2: HC-SR04 Ultrasonic specifications.

HC-SR04 Ultrasonic Specifications
Working Voltage: DC 5 V
Working Current: 15 mA
Working Frequency: 40 Hz
Max Range: 4 m
Min Range: 2 cm
Measuring Angle: 15 degree
Trigger Input Signal: 10 µS TTL pulse
Echo Output Signal Input TTL lever signal and the range in proportion
Dimension 45 * 20 * 15 mm

A.3 The table below shows the HD Webcam Logitech C270 specifications.

Table A.3: HD Webcam Logitech C270 specifications.

HD Webcam Logitech C270 Specifications
Connection Type: Corded USB
USB Type: High Speed USB 2.0
USB VID_PID: VID_046D&PID_081A
Microphone: Built-in, Noise Supression
Lens and Sensor Type: Plastic
Focus Type: Fixed
Field of View (FOV): 60°
Focal Length: 4.0 mm
Optical Resolution (True): 1280 x 960 1.2 MP
Image Capture (4:3 SD): 320x240, 640x480 1.2 MP, 3.0 MP
Image Capture (16:9 W): 360p, 480p, 720p
Video Capture (4:3 SD): 320x240, 640x480, 800x600
Video Capture (16:9 W): 360p, 480p, 720p,
Frame Rate (max): 30fps @ 640x480
Right Light: Right Light 2
Indicator Lights (LED): Activity/Power
Privacy Shade: No
Clip Size (max): 0 to infinity
Cable Length 5 Feet or 1.5 Metres

A.4 The table below shows the ODROID-U3$^+$ specifications.

Table A.4: ODROID-U3$^+$ specifications.

ODROID-U3$^+$ specifications
5V 2A Power
1.7GHz Quad-Core processor and 2GByte RAM
10/100Mbps Ethernet with RJ-45 LAN Jack
3 x High speed USB2.0 Host ports
Audio codec with headphone jack on board
GPIO/UART/I2C ports
XUbuntu 13.10 or Android 4.x Operating System
Size : 83 x 48 mm, Weight : 48g including heat sink
Package includes the main board and the heat sink

APPENDIX B

Software algorithms

The following are the algorithms and the codes that used in this research.

- Matlab code for invasive weed detection and mapping.
- Python code for the on-board decision making using Raspberry Pi 2B.
- The ROS nodes for the system is attached as follow:

1. Lunch file for the system using Odroid U3.
2. Marker detection node.
3. Colour detection node.
4. weed detection node.
5. Transfer pixels to local position node.
6. Ultrasonic node.
7. Navigation node.

- Arduino code for controlling the Ultrasonic (HC-SR04) and the spraying pump.
- Matlab Code for drawing flight trajectory.

B.1 Matlab code for invasive weed detection and mapping.

Algorithm B.1 Matlab code for invasive weed detection and mapping.

MATLAB code for weed detection and Mapping

```
function varargout = Test_GUI_v1(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Test_GUI_v1_OpeningFcn, ...
                   'gui_OutputFcn',  @Test_GUI_v1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Test_GUI_v1 is made visible.
function Test_GUI_v1_OpeningFcn(hObject, eventdata, handles, varargin)
%

% Choose default command line output for Test_GUI_v1
handles.output = hObject;
%
guidata(hObject, handles);

% UIWAIT makes Test_GUI_v1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Test_GUI_v1_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Lat;
global Lon;
global FOV; % Horizontal field of view via https://www.raspberrypi.↙
org/documentation/hardware/camera.md
FOV = 53.5;
global Width;
global Height;
[FileName,PathName] = uigetfile('*.jpg','Select Image');
file = strcat(PathName,FileName);
```

```
image=imread(file);
redBand = image(:,:, 1);
greenBand =image(:,:, 2);
blueBand = image(:,:, 3);

redthreshold = 68;
greenThreshold = 72;
blueThreshold = 74;
redMask = (redBand > redthreshold);
greenMask = (greenBand < greenThreshold);
blueMask = (blueBand < blueThreshold);
redObjectsMask = uint16(redMask & greenMask & blueMask);

%  imshow(redObjectsMask,[])
Ifill=imfill(redObjectsMask,'holes');
[ Ilabel num]=bwlabel(Ifill);
Iprops=regionprops(Ilabel);

 Ibox=[Iprops.BoundingBox];
 Ibox=reshape(Ibox,[4 num]);
 axes(handles.axes1)

finalImage = image;
imageHandle = imshow(finalImage);
hold on
 for cnt=1:num
    rectangle('position',Ibox(:,cnt),'edgecolor','b','linewidth',1);
 end

hold off
[n m p] = size(image);
density =(100*num/(n*m))*100;
set(findobj('Tag','edit4'),'String',[ num2str(density,'%.20f')])
F = getframe(handles.axes1);
finalImage = frame2im(F);

imageHandle = imshow(finalImage);
imagedetails= imfinfo(file);
i = exist('imagedetails.GPSInfo');
if (i ~= 0)
    Lat_Deg = imagedetails.GPSInfo.GPSLatitude;
    Lon_Deg = imagedetails.GPSInfo.GPSLongitude;
    Lat = Lat_Deg(1) + (Lat_Deg(2)/60) + (Lat_Deg(3)/3600);
    Lon = Lon_Deg(1) + (Lon_Deg(2)/60) + (Lon_Deg(3)/3600);
else
    Lat = -28.2058929;
    Lon = 153.0036092;
end
Width = imagedetails.Width;
Height = imagedetails.Height;
set(imageHandle,'ButtonDownFcn',@ImageClickCallback);
global PointCount;
PointCount = 0;
global La;
```

```
global Lo;
La = zeros(4,1);
Lo = zeros(4,1);
global Xpts;
global Ypts;
Xpts=zeros(17,1);
Ypts=zeros(17,1);


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)

close all

function ImageClickCallback ( objectHandle , eventData)
axesHandle  = get(objectHandle,'Parent');
coordinates = get(axesHandle,'CurrentPoint');
coordinates = coordinates(1,1:2);
global FOV;
global Width;
```

```
global Height;
UASHeight = 5;
 %set(findobj('Tag','edit1'),'String',[ num2str(coordinates(1))]) % updating X↙
coordinate in the static text box
 %set(findobj('Tag','edit2'),'String',[ num2str(coordinates(2))]) % updating Y↙
coordinate in the static text box
%  valx = coordinates(1)*((1.3*10^-3)/4)*5;%/3.7796;
%  valy = coordinates(2)*((1.3*10^-3)/4)*5;%/3.7796;
 pixLength = (tand(FOV/2)* UASHeight)/(Width/2);
 valx = coordinates(1)* pixLength;
 valy = coordinates(2)* pixLength;
 double Lat;double Lon ;
 global Lat;
 global Lon;

 %Lat = -28.204531;
 %Lon = 153.003782;
 [x,y,utmzone] = deg2utm(Lat,Lon);
 if (coordinates(1) < 1296 && coordinates(2) < 972)
     xx = x - valx;
     yy = y - valy;
 elseif (coordinates(1) < 1296)
     xx = x - valx;
     yy = y + valy;
 elseif (coordinates(2) < 972)
     xx = x + valx;
     yy = y - valy;
 else
     xx = x + valx;
     yy = y + valy;
 end
 format long
 xx;
 format long
 yy;
 global PointCount;
 PointCount = PointCount + 1;
 if PointCount > 17
     PointCount = 0;
 end
 [Latf,Lonf] = utm2deg(xx,yy,utmzone);
 format long
 Latf;
 format long
 Lonf;
 set(findobj('Tag','edit1'),'String',[ num2str(Latf,'%.20f')]) % updating X coordinate↙
in the static text box
 set(findobj('Tag','edit2'),'String',[ num2str(Lonf,'%.20f')])
 index = mod(PointCount,10);
 if (index== 0)
     index=10;
     PointCount = 0;
 end

 global La;
```

```
 global Lo;
 global Xpts;
 global Ypts;
 Xpts(index) = coordinates(1);
 Ypts(index) = coordinates(2);
 La(index) = Latf;
 Lo(index) = Lonf;
 Values = [La,Lo];
 set(findobj('Tag','uitable1'), 'Data',Values);
 hold on
 scatter(Xpts,Ypts,'filled','red')
 hold off
```

```
% --- Executes on button press in btnLoadFL.
function btnLoadFL_Callback(hObject, eventdata, handles)
% hObject    handle to btnLoadFL (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[FileName,PathName] = uigetfile('*.mat','Select Flight Log');
file = strcat(PathName,FileName);
test = load(file);
lat_test=test.GPS(:,7);
lon_test=test.GPS(:,8);
DateTime =test.GPS(:,3);
for i=1:length(DateTime)
    value = MStoLT(DateTime(i));
    Time(i,1:3)=value(:,4:6);
end
```

```
function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
function edit4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

B.2 Python code for the on-board decision making using Raspberry Pi 2B.

Algorithm B.2 Python Code for On-board decision making Using Raspberry Pi 2B.

```
import time
from droneapi.lib import VehicleMode, Location
from pymavlink import mavutil
import cv2
import re
import numpy as np
import math
import argparse
import RPi.GPIO as GPIO
from time import sleep

api = local_connect()
vehicle = api.get_vehicles()[0]

def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """

    print "Basic pre-arm checks"
    # Don't let the user try to fly autopilot is booting
    if vehicle.mode.name == "INITIALISING":
        print "Waiting for vehicle to initialise"
        time.sleep(1)
    while vehicle.gps_0.fix_type < 2:
        print "Waiting for GPS...:", vehicle.gps_0.fix_type
        time.sleep(1)

    print "Arming motors"
    # Copter should arm in GUIDED mode
    vehicle.mode    = VehicleMode("GUIDED")
    vehicle.armed   = True
    vehicle.flush()

    while not vehicle.armed and not api.exit:
        print " Waiting for arming..."
        time.sleep(1)

    print "Taking off!"
    vehicle.commands.takeoff(aTargetAltitude) # Take off to target altitude
    vehicle.flush()

    # Wait until the vehicle reaches a safe height before processing the goto (otherwise the
    command
    #  after Vehicle.commands.takeoff will execute immediately).
    while not api.exit:
        print " Altitude: ", vehicle.location.alt
        if vehicle.location.alt>=aTargetAltitude*0.95: #Just below target, in case of
        undershoot.
            print "Reached target altitude"
            break;
        time.sleep(1)

def set_speed(speed):
    #Send MAV_CMD_DO_CHANGE_SPEED to change the current speed when travelling to a point.
    # create the MAV_CMD_DO_CHANGE_SPEED command
    msg = vehicle.message_factory.command_long_encode(
        0, 0,    # target system, target component
        mavutil.mavlink.MAV_CMD_DO_CHANGE_SPEED, #command
        0, #confirmation
        0, #param 1
        speed, # speed
        0, 0, 0, 0, 0 #param 3 - 7
        )

    # send command to vehicle
    vehicle.send_mavlink(msg)
    vehicle.flush()
```

```
def motor(sec):
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(40,GPIO.OUT)
    GPIO.output(40,True)
    print("The motor is ON")
    time.sleep(sec)
    GPIO.output(40,False)
    print("The motor is OFF")


def Ult(sec):
    GPIO.setmode(GPIO.BCM)

    TRIG = 23
    ECHO = 24

    print "Distance Measurement In Progress"

    GPIO.setup(TRIG,GPIO.OUT)
    GPIO.setup(ECHO,GPIO.IN)

    GPIO.output(TRIG, False)
    print "Waiting For Sensor To Settle"
    time.sleep(sec)

    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    while GPIO.input(ECHO)==0:
      pulse_start = time.time()

    while GPIO.input(ECHO)==1:
      pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start

    distance = pulse_duration * 17150

    distance = round(distance, 2)
    print '\n'
    print "Distance:",distance,"cm"
    return distance


def heading(IRIS_Lat,IRIS_Lon,Target_Lat,Target_Lon,xLo1,yLo1):


    IRIS_Lat=abs(IRIS_Lat);
    Target_Lat=abs(Target_Lat);

    Delta_Lat=abs(Target_Lat)-abs(IRIS_Lat);
    Delta_Lon=abs(Target_Lon)-abs(IRIS_Lon);
    Lat=Delta_Lat/132.936
    Lon=Delta_Lon/132.936
    if IRIS_Lat > Target_Lat and IRIS_Lon < Target_Lon:
        print('1 coordination')
        target_lat=-(IRIS_Lat-Lat*xLo1)
        print target_lat
        target_lon=IRIS_Lon-Lon*yLo1
        print target_lon
        return target_lat, target_lon

    elif IRIS_Lat > Target_Lat and IRIS_Lon > Target_Lon:
        print('2 coordination')
        target_lat=-(IRIS_Lat-Lat*xLo1)
        target_lon=IRIS_Lon-Lon*yLo1
        return target_lat, target_lon
```

```
    elif IRIS_Lat < Target_Lat and IRIS_Lon > Target_Lon:
        print('3 coordination')
        target_lat=-(IRIS_Lat+Lat*xLo1)
        target_lon=IRIS_Lon-Lon*yLo1
        return target_lat, target_lon

    elif IRIS_Lat < Target_Lat and IRIS_Lon < Target_Lon:
        print('4 coordination')
        target_lat=-(IRIS_Lat+Lat*xLo1)
        target_lon=IRIS_Lon+Lon*yLo1
        return target_lat, target_lon

    elif IRIS_Lat > Target_Lat and IRIS_Lon == Target_Lon:
        print('N')
        target_lat=-(IRIS_Lat-Lat*xLo1)
        target_lon=IRIS_Lon
        return target_lat, target_lon

    elif IRIS_Lat < Target_Lat and IRIS_Lon == Target_Lon:
        print('S')
        target_lat=-(IRIS_Lat+Lat*xLo1)
        target_lon=IRIS_Lon
        return target_lat, target_lon

    elif IRIS_Lat == Target_Lat and IRIS_Lon < Target_Lon:
        print('E')
        target_lat=-IRIS_Lat
        target_lon=IRIS_Lon+Lon*yLo1
        return target_lat, target_lon

    elif IRIS_Lat == Target_Lat and IRIS_Lon > Target_Lon:
        print('W')
        target_lat=-IRIS_Lat
        target_lon=IRIS_Lon-Lon*yLo1
        return target_lat, target_lon
    else:
        print('It has not been move')
        target_lat=-IRIS_Lat
        target_lon=IRIS_Lon
        return target_lat, target_lon


a='Location:lat=-27.4564486,lon=153.0122028,alt=0.77,is_relative=True'
print " Location: %s" % vehicle.location
lat=re.search('lat=(.*?),lon',a).group(1)
lon=re.search('lon=(.*?),alt',a).group(1)

Past_lat=float(lat)
Past_lon=float(lon)

print "UAVs Past %s" %a

print "initial position %s" % vehicle.location
time.sleep(1)

camera = cv2.VideoCapture(0)



def precheck():
    # find contours in the image
    ret, frame = camera.read()
    # determine which pixels fall within the weed( in this case) boundaries
    # and then blur the binary image

    b,g,r = cv2.split(frame)

    g_mask = np.copy(g)
```

```
    g_mask[g > 70] = 255
    g_mask[g <= 70] = 0

    r_mask = np.copy(r)
    r_mask[r < 68] = 255
    r_mask[r >= 68] = 0

    b_mask = np.copy(b)
    b_mask[b < 72] = 255
    b_mask[b >= 72] = 0

    mask = g_mask & r_mask & b_mask

    mask = cv2.medianBlur(mask,7)

    kernel = np.ones((4,4),np.uint8)

    mask = cv2.erode(mask,kernel)
    mask = cv2.dilate(mask, kernel)

    (cnts, _) = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# check to see if any contours were found
    if len(cnts) > 0:
          check=1
          print(' an object has been detected ')
          return check
    else:
        print(' no object has been detected ')
        check=0
        return check

# capture frames from the camera
def detect():
    print "Object detection loop start"
    while True:
        print 'Object detection loop operative'
        ## Up-date Current gps location.

        Past_lat=vehicle.location.lat
        Past_lon=vehicle.location.lon

##      print "UAVs Past %s" %vehicle.location
        time.sleep(0.025)


        ## Object detection start from here.
        # grab the current frame
        ret, frame = camera.read()
        # check to see if we have reached the end of the
        # video
        if not ret:
            break
        b,g,r = cv2.split(frame)
        g_mask = np.copy(g)
        g_mask[g > 70] = 255
        g_mask[g <= 70] = 0
        r_mask = np.copy(r)
        r_mask[r < 68] = 255
        r_mask[r >= 68] = 0
        b_mask = np.copy(b)
        b_mask[b < 72] = 255
        b_mask[b >= 72] = 0
        mask = g_mask & r_mask & b_mask
        mask = cv2.medianBlur(mask,7)
        kernel = np.ones((4,4),np.uint8)
        mask = cv2.erode(mask,kernel)
        mask = cv2.dilate(mask, kernel)
        # find contours in the image
```

```
(cnts, _) = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)


# check to see if any contours were found
if len(cnts) > 0:
    # sort the contours and find the largest one -- we
    # will assume this contour correspondes to the area
    # of my phone

    cnt = sorted(cnts, key = cv2.contourArea, reverse = True)[0]

    # compute the (rotated) bounding box around then
    # contour and then draw it
    rect = np.int32(cv2.cv.BoxPoints(cv2.minAreaRect(cnt)))
    xLo=(rect[0][1]+rect[2][1])/2
    yLo=(rect[2][0]+rect[0][0])/2

    # cv2.imshow("Frame", frame)
    # cv2.imshow("Binary", mask)


    # Calculate a target location and angle
    if yLo < 240:
            yLo1=240-yLo
    elif yLo > 240:
            yLo1=-(240-yLo)
    elif yLo == 240:
            yLo1=0

    if xLo < 320:
            xLo1=-(320-xLo)
    elif xLo > 320:
            xLo1=640-xLo
    elif xLo == 320:
            xLo1=0
    angle=math.atan2(yLo1,xLo1)*(180/math.pi)

    if angle > 90:
            angle1=90-angle
    elif angle < 90:
            angle_t=180-angle
            angle1=-(90-angle_t)

    location=[ xLo1, yLo1 ]
    # Save the image

    cv2.imwrite("Weed_Detect"".jpg",frame)
    cv2.imwrite("weed_Binary"".jpg",mask)

    # Current GPS update
    print 'Current GPS update syncronization'
    a='Location:lat=-27.4564318,lon=153.0121937,alt=30,is_relative=True'
    print " location: %s" % a
    lat=re.search('lat=(.*?),lon',a).group(1)
    lon=re.search('lon=(.*?),alt',a).group(1)

    UAV_lat=float(lat)
    UAV_lon=float(lon)

    print "UAVs Current %s" %a

    ## Find the target GPS location
    [target_lat,target_lon]=heading(Past_lat,Past_lon,UAV_lat,UAV_lon,xLo1,yLo1)


    print "pixels location"
    print(xLo,yLo)
    print "x-y coordinator=","[",xLo1, yLo1,"]"
```

```
            #print(xLo1,yLo1)
            #print " angle in degree "
            #print(angle1)

            return target_lat,target_lon, xLo1,yLo1,angle1

            break
            # print 'Current GPS update syncronization'

            UAV_lat=vehicle.location.lat       #( check it later for flight test)#
            UAV_lon=vehicle.location.lon

            # print "UAVs Current %s" %vehicle.location

            ## Find the target GPS location
            [target_lat,target_lon]=heading(Past_lat,Past_lon,UAV_lat,UAV_lon,xLo1,yLo1)
            print 'Target Location found'
            target_location=[target_lat,target_lon]
            #print "pixels location"
            #print(xLo,yLo)
            print "x-y coordinator"
            print(xLo1,yLo1)
            # #print " angle in degree "
            # #print(angle1)

            return target_lat,target_lon,xLo1,yLo1,angle1

            break

def weed(loop,alt,point):
    for x in xrange (loop):

        time.sleep(0.5)
        print x

        check=precheck()
        print check
        if check == 1:
            [target_lat,target_lon,xLo1,yLo1,angle1]=detect()
            print (" the target location has been found ")
            print target_lat,target_lon
            set_speed(1)
            #find object and fly to target's location
            target=Location(target_lat,target_lon, alt, is_relative=True)
            vehicle.commands.goto(target)
            vehicle.flush()

            time.sleep(10)

            point=point
            print " return to the flight path "
            vehicle.commands.goto(point)
            vehicle.flush()
            time.sleep(5)
            # break
        elif x==loop-1:
            print " no object has been found between point A and B "
            break



# ##
# arm_and_takeoff(5) # Altitude in meter.
# ##
time.sleep(2)
#################################################################
print (" set speed to 1m/s ")
set_speed(1)
```

```
print ("Going to first point A")
point1 = Location(-27.2334846, 152.9778489,5 , is_relative=True)
vehicle.commands.goto(point1)
vehicle.flush()
#################
weed(60,2,point1) #loop, Altitude, location
distance = 0.0
distance = Ult(1)
if distance <= 50.0:
  motor(3)
  GPIO.cleanup()
  else:
 print '\n'
 print "Compare the altitude of the Ultrasonic with MAVproxy altitude"
 print '\n'
time.sleep(2)
#####################################################################
print ("Set speed to 1m/s")
set_speed(1)
print "Going to point B"
point2 = Location(-27.2334339,152.9778474, 1, is_relative=True)
vehicle.commands.goto(point2)
vehicle.flush()
###################
weed(60,2,point1) #loop, Altitude, location
distance = 0.0
distance = Ult(1)
GPIO.cleanup()
if distance <= 50.0:

    motor(3)
    GPIO.cleanup()
else:
 print '\n'
 print "Compare the altitude of the Ultrasonic with MAVproxy altitude"
 print '\n'
time.sleep(5)
#####################################################################
print ("Set speed to 5m/s")
set_speed(1)
print "Going to Home"
home = Location(-27.2334339,152.9778474, 5, is_relative=True)
vehicle.commands.goto(home)
vehicle.flush()

print("mission completed LAND mode...")
vehicle.mode = VehicleMode("LAND")
vehicle.flush()

vehicle.flush()
camera.release()
print " Finish"
```

B.3 The ROS nodes for the system is attached as follow:

B.3.1 Lunch file for the system using Odroid U3.

Algorithm B.3 Robotic Operating System (ROS) System Lunch File.

launch file for the onboard system

```
<launch>
    <!-- USB CAMERA -->
    <node name="camera_rgb" pkg="usb_cam" type="usb_cam_node"
output="screen" >
        <param name="video_device" value="/dev/video0" />
        <param name="image_width" value="640" />
        <param name="image_height" value="480" />
        <param name="pixel_format" value="yuyv" />
        <param name="framerate" value="10" />
        <param name="camera_frame_id" value="camera_rgb" />
        <param name="camera_name" value="camera_rgb" />
        <param name="io_method" value="mmap"/>
    </node>

    <!-- MAVROS -->
    <!-- vim: set ft=xml noet : -->
    <!-- example launch script for PX4 based FCU's -->

    <arg name="fcu_url" default="/dev/ttyUSB0:921600" />
    <arg name="gcs_url" default="" />
    <arg name="tgt_system" default="1" />
    <arg name="tgt_component" default="1" />
    <arg name="log_output" default="screen" />

    <include file="$(find mavros)/launch/node.launch">
        <arg name="pluginlists_yaml" value="$(find mavros)/launch/
px4_pluginlists.yaml" />
        <arg name="config_yaml" value="$(find mavros)/launch/
px4_config.yaml" />

        <arg name="fcu_url" value="$(arg fcu_url)" />
        <arg name="gcs_url" value="$(arg gcs_url)" />
        <arg name="tgt_system" value="$(arg tgt_system)" />
        <arg name="tgt_component" value="$(arg tgt_component)" />
        <arg name="log_output" value="$(arg log_output)" />
    </include>

    <!-- SPRAY NAV -->
    <node name="spray_nav" pkg="spray_nav" type="spray_nav_node"
output="screen" >
        <param name="height_takeoff" value="3.0" type="double" />
        <param name="height_search" value="3.0" type="double" />
        <param name="height_spray" value="0.6" type="double" />

        <param name="waypoint_radius" value="0.2" type="double" />
        <param name="vs_radius" value="0.1" type="double" />
```

```
        <param name="fov_degrees" value="60" type="double" />
        <param name="camera_width" value="640" type="double" />
        <param name="camera_height" value="480" type="double" />

        <param name="position_input" value="/mavros/local_position/
pose" type="str" />
        <param name="vs_input" value="/target/pose" type="str" />
    </node>
</launch>
```

B.3.2 Marker detection node.

Algorithm B.4 Robotic Operating System (ROS) Marker detection node.

```
<?xml version="1.0" encoding="utf-8"?>

<launch>
    <node pkg="ar_sys" type="single_board" name="ar_single_board"
output="screen">
        <remap from="/camera_info" to="/camera_rgb/camera_info" />
        <remap from="/image" to="/camera_rgb/image_raw" />

        <param name="image_is_rectified" type="bool" value="true"/>
        <param name="marker_size" type="double" value="0.13"/>
        <param name="board_config" type="string" value=
      "$(find ar_sys)/data/single/board.yml"/>
        <param name="board_frame" type="string" value="/board1" />
        <param name="draw_markers" type="bool" value="false" />
        <param name="draw_markers_cube" type="bool" value="true" />
        <param name="draw_markers_axis" type="bool" value="false" />
    </node>
</launch>
```

B.3.3 Colour detection node.

Algorithm B.5 Robotic Operating System (ROS) Colour detection node.

```
//Includes all the headers necessary to use the most common public pieces of the ROS system.
#include <ros/ros.h>
//Include geometry_msgs for publishing the estimated target pose
#include <geometry_msgs/PoseStamped.h>
//Use image_transport for publishing and subscribing to images in ROS
#include <image_transport/image_transport.h>
//Include some useful constants for image encoding.
#include <sensor_msgs/image_encodings.h>
//Use cv_bridge to convert between ROS and OpenCV Image formats
#include <cv_bridge/cv_bridge.h>
//Include headers for OpenCV Image processing
#include <opencv2/imgproc/imgproc.hpp>
//Include headers for OpenCV
#include <cv.h>
#include <math.h>
#include <std_msgs/Float64.h>

//Store all constants for image encodings in the enc namespace to be used later.
namespace enc = sensor_msgs::image_encodings;


//Global Variables
image_transport::Publisher debug_pub;
ros::Publisher pose_out;
double height_us = 0;

void us_cb(const std_msgs::Float64 msg) {
    height_us = msg.data;
}

void colorDetectionCallback(const sensor_msgs::ImageConstPtr& original_image) {
    //Convert from the ROS image message to a CvImage suitable for working with OpenCV for
    processing
    cv_bridge::CvImagePtr cv_ptr;

    try {
        //Always copy, returning a mutable CvImage
        //OpenCV expects color images to use BGR channel order.
        cv_ptr = cv_bridge::toCvCopy(original_image, enc::BGR8);
    }
    catch (cv_bridge::Exception& e) {
        //if there is an error during conversion, display it
        ROS_ERROR("tutorialROSOpenCV::main.cpp::cv_bridge exception: %s", e.what());
        return;
    }

    //Noise filter
    //NOTE: Probably don't need this as a Gaussian blur is applied later on
    cv::medianBlur( cv_ptr->image,  cv_ptr->image, 3);

    //Convert original image to Gray
    //NOTE: Not sure if this needs to actually be done...
    //IplImage image = cv_ptr->image.operator IplImage(); //convert Mat to IplImage

    //Start looking for circle
    //Convert input image to hsv
    cv::Mat img_hsv;
    cv::cvtColor(cv_ptr->image,img_hsv,CV_BGR2HSV);

    // Threshold the HSV image, keep only the red pixels
    cv::Mat lower_red_hue_range;
    cv::Mat upper_red_hue_range;
    cv::inRange(img_hsv, cv::Scalar(0, 100, 100), cv::Scalar(5, 255, 255),
    lower_red_hue_range);
    cv::inRange(img_hsv, cv::Scalar(175, 100, 100), cv::Scalar(179, 255, 255),
    upper_red_hue_range);

    // Combine the above two images
    cv::Mat red_hue_image;
    cv::addWeighted(lower_red_hue_range, 1.0, upper_red_hue_range, 1.0, 0.0, red_hue_image);
    cv::GaussianBlur(red_hue_image, red_hue_image, cv::Size(9, 9), 2, 2); // Reduce the
```

```
    noise to avoid false circle detection

    // Use the Hough transform to detect red circles in the combined threshold image
    std::vector<cv::Vec3f> circles;
    std::vector<cv::Point> found_centers;

    //Decide how far away the circles should be from each other
    int min_dist = red_hue_image.rows/8;

    if(min_dist < 8)
        min_dist = 8;

    //Scan the red-filtered image for circles
    cv::HoughCircles(red_hue_image, circles, CV_HOUGH_GRADIENT, 1, min_dist, 100, 20, 30,
    150);

    //If there was only 1 circle found in the image, publish that data
    //NOTE: This is quite poor, all circles should be checked against a desired diameter or
    something
    if(circles.size() == 1) {
        geometry_msgs::PoseStamped msg;

        //double px_ratio = 0.00902;
        //double px_ratio = 0.0054;
        double px_ratio = height_us*tan(60/2)/320;

        double cam_x = (cvRound(circles[0][0]) - 320)*px_ratio;
        double cam_y = (cvRound(circles[0][1]) - 240)*px_ratio;

        ROS_INFO("Target location in the camera frame is: [%0.2f, %0.2f", cam_x, cam_y);

        msg.header.stamp = ros::Time::now();
        msg.header.frame_id = "/camera";

        msg.pose.position.x = cam_x;
        msg.pose.position.y = cam_y;

        pose_out.publish(msg);
    }

    //If there are subscribers, publish debug image
    if(debug_pub.getNumSubscribers() > 0) {
        for( int i = 0; i < circles.size(); i++ ) {
            int radius = cvRound(circles[i][2]);

            CvPoint red_center = {cvRound(circles[i][0]), cvRound(circles[i][1])};

            // circle center
            circle( cv_ptr->image, red_center, 3, cv::Scalar(255,0,0), -1, 8, 0 );
            // circle outline
            circle( cv_ptr->image, red_center, radius, cv::Scalar(0,255,0), 3, 8, 0 );
        }

        debug_pub.publish(cv_ptr->toImageMsg());
    }
}

int main(int argc, char **argv)
{
    //Initialize the original node
    ros::init(argc, argv, "red_detection");

    //Create the handles for both the node and image transport
    ros::NodeHandle nh(ros::this_node::getName());
    image_transport::ImageTransport it(nh);

    //Initialize the default camera message
    std::string camera_topic = "/camera/image_raw";

    if (nh.getParam("image_input", camera_topic))
       ROS_INFO("Connecting to camera: %s", camera_topic.c_str());
```

```
    //Subscribers
    image_transport::Subscriber sub = it.subscribe(camera_topic, 1, colorDetectionCallback);

    ros::Subscriber us_sub = nh.subscribe<std_msgs::Float64>
        ("/us_distance", 10, us_cb);

    //Publishers
    debug_pub = it.advertise("image_processed", 1);
    pose_out = nh.advertise<geometry_msgs::PoseStamped>("target_location", 100);

    //Lock thread and listen to messages
    ros::spin();
}
```

B.3.4 weed detection node.

Algorithm B.6 Robotic Operating System (ROS) Weed detection node.

```
#!/usr/bin/env python
import rospy
import sys
import cv2
import numpy
import numpy as np
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image
from geometry_msgs.msg import PoseStamped

class image_converter():
        def __init__(self):

                self.pose_pub =
                rospy.Publisher("/weed_detection/target_location",PoseStamped,queue_size=10)
                self.debug_image_pub = rospy.Publisher("debug_image",Image,queue_size=1)
                self.bridge = CvBridge()
                self.image_sub = rospy.Subscriber("/camera_rgb/image_raw",Image,self.callback)
                self.debug_pub = 1;

        def callback(self,data):
                print "Analysing new image..."
                try:
                    cv_image = self.bridge.imgmsg_to_cv2(data, "rgb8")
                except CvBridgeError as e:
                        print(e)

                r,g,b=cv2.split(cv_image)

                g_mask = numpy.copy(g)
                g_mask[g > 75] = 255
                g_mask[g <= 75] = 0

                r_mask = numpy.copy(r)
                r_mask[r < 68] = 255
                r_mask[r >= 68] = 0

                b_mask = numpy.copy(b)
                b_mask[b < 62] = 255
                b_mask[b >= 62] = 0

                mask = g_mask & r_mask & b_mask

                mask = cv2.medianBlur(mask,7)

                kernel = numpy.ones((4,4),numpy.uint8)

                mask = cv2.erode(mask,kernel)
                Col = cv2.dilate(mask, kernel)

                Col= cv2.GaussianBlur(Col, (3,3),0)

                # find contours
                (cnts, _) = cv2.findContours(Col.copy() , cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)

                if self.debug_pub:
                        cv2.drawContours(cv_image, cnts, -1, (0,255,0), 3)

                #check to see if any contours were found
                if len(cnts) > 0:
                        #short
                        cnt = sorted(cnts, key =cv2.contourArea, reverse = True)[0]

                        rect = np.int32(cv2.cv.BoxPoints(cv2.minAreaRect(cnt)))

                        xLo=(rect[0][1]+rect[2][1])/2
                        yLo=(rect[2][0]+rect[0][0])/2
```

```
                        print "Found target in image!"

                        cam_x = (xLo - 320)*0.00902
                        cam_y = (yLo - 240)*0.00902

                        pose_out = PoseStamped()
                        pose_out.header.frame_id = "/camera"
                        pose_out.header.stamp = rospy.get_rostime()
                        pose_out.pose.position.x = cam_x
                        pose_out.pose.position.y = cam_y

                        self.pose_pub.publish(pose_out)

                        if self.debug_pub:
                                try:

                                        self.debug_image_pub.publish(self.bridge.cv2_to_imgmsg
                                        (cv_image, "rgb8"))
                                except CvBridgeError as e:
                                        print(e)

def main(args):
        ic = image_converter()
        rospy.init_node('image_converter', anonymous=True)
        try:
                rospy.spin()
        except KeyboardInterrupt:
                print("Shutting down")

if __name__ == '__main__':
        main(sys.argv)
```

B.3.5 Transfer pixels to local position node.

Algorithm B.7 Robotic Operating System (ROS) Transfer pixels to local position node.

```python
#!/usr/bin/env python
# license removed for brevity
import rospy
from geometry_msgs.msg import PoseStamped
import numpy as np

pub = rospy.Publisher('/target/pose', PoseStamped, queue_size=10)


def callback(data):
    #rospy.loginfo(data)
    pose_out = data

    pose_out.header.frame_id = "/uavbody"

    R = np.matrix([[0, 1, 0], [1, 0, 0], [0, 0, -1]])   # camera to body frame rotation

    p = np.matrix([[data.pose.position.x],[data.pose.position.y],[data.pose.position.z]])

    p_dash = R*p

    pose_out.pose.position.x = p_dash.item(0)
    pose_out.pose.position.y = p_dash.item(1)
    pose_out.pose.position.z = p_dash.item(2)

    pose_out.pose.orientation.x = 0
    pose_out.pose.orientation.y = 0
    pose_out.pose.orientation.z = 0
    pose_out.pose.orientation.w = 1


    rospy.loginfo(pose_out)
    pub.publish(pose_out)

def listener():
    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('FrameTransform', anonymous=True)

    #input_str = "/ar_single_board/pose"
    #input_str = "/color_detection/target_location"
    input_str = "/red_detection/target_location"
    #input_str = "/weed_detection/target_location"

    print("Listening to: " + input_str)
    print("Outputting on: /target/pose")

    rospy.Subscriber(input_str, PoseStamped, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

B.3.6 Ultrasonic node.

Algorithm B.8 Robotic Operating System (ROS) Ultrasonic node.

```
#! /usr/bin/env python
import serial
import rospy
import sys
from std_msgs.msg import Float64

ser=serial.Serial('/dev/ttyACM0',115200)

ser.flushInput()

def ultrasonic():
    pub = rospy.Publisher('us_distance',Float64,queue_size=10)
    rospy.init_node('ultrasonic', anonymous=True)

    while not rospy.is_shutdown():
        readdata=ser.readline()

        if readdata[0][:1] != '-':
            height = float(readdata)/100

            rospy.loginfo(height)

            msg = Float64
            msg = height
            pub.publish(msg)

if __name__=='__main__':
    try:
        ultrasonic()
    except rospy.ROSInterruptException:
        pass
```

B.3.7 Navigation node.

Algorithm B.9 Robotic Operating System (ROS) Navigation node.

```
/**
* @file offb_node.cpp
* @brief offboard example node, written with mavros version 0.14.2, px4 flight
* stack and tested in Gazebo SITL
*/
#include <string.h>
#include <ros/ros.h>
#include <math.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/Point.h>
#include <std_msgs/Float64.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>


//================================//
// Global Parameters              //
//================================//

enum nav_modes {
    NAV_MODE_TAKEOFF,
    NAV_MODE_WAYPOINT_A,
    NAV_MODE_WAYPOINT_B,
    NAV_MODE_FOUND_PLANT,
    NAV_MODE_SPRAY_DESC,
    NAV_MODE_SPRAY_WAIT,
    NAV_MODE_SPRAY_ASC,
    NAV_MODE_RTL,
};


//================================//
// Global Variables               //
//================================//
mavros_msgs::State current_state;
geometry_msgs::Point current_pos;
geometry_msgs::Point current_goal;
unsigned int nav_mode = NAV_MODE_TAKEOFF;

geometry_msgs::Point waypoint_home;
geometry_msgs::Point waypoint_a;
geometry_msgs::Point waypoint_b;
geometry_msgs::Point waypoint_plant;

bool ignore_ip = true;
bool land_mode = false;
double height_search = 2.0;
double height_takeoff = 2.0;
double height_spray = 0.5;
double height_us = 0;
ros::Time start_spray_time;

double waypoint_radius = 0.2;
double vs_radius = 0.2;

double fov = 0;
double camera_height = 480;
double camera_width = 640;
//estimated_pos.z = current_pos.z - real_height + 0.5;

//================================//
// Callback Functions             //
//================================//

void state_cb(const mavros_msgs::State::ConstPtr& msg){
    current_state = *msg;
}

void us_cb(const std_msgs::Float64 msg){
    height_us = msg.data;
```

```
}
/*
void ip_cb(const geometry_msgs::Point msg){
    if(!ignore_ip){
        //msg.x ---> pixel X location
        //estimate pixel location in meters
        //rotate point --- for now, assume you don't have to
        //set new goal based on estimate

        //Just to do an example
        geometry_msgs::Point estimated_pos; //estimated position of the plant in the world
        frame

        //double x2 = current_pos.z * tan(fov_height/2);
        double dpx = current_pos.z * tan(fov/2);

        estimated_pos.x = (msg.x*dpx)/(camera_width/2);
        estimated_pos.y = (msg.y*dpx)/(camera_width/2);
        estimated_pos.z = 0;

        //estimated_pos.z = current_pos.z;

        waypoint_plant.x = current_pos.x + estimated_pos.x;
        waypoint_plant.y = current_pos.y + estimated_pos.y;
        waypoint_plant.z = current_pos.z + estimated_pos.z;

        nav_mode = NAV_MODE_FOUND_PLANT;
        ROS_INFO("Calculated [du, dv] as: [%0.2f, %0.2f]", estimated_pos.x, estimated_pos.y);
        ROS_INFO("Estimated plant location at: [%0.2f, %0.2f]", waypoint_plant.x,
        waypoint_plant.y);
        ROS_INFO_ONCE("Guiding center of the plant");
    }
}
*/

void ip_cb(const geometry_msgs::PoseStamped msg){
    if(!ignore_ip){
        geometry_msgs::Point estimated_pos; //estimated position of the plant in the world
        frame

        estimated_pos.x = msg.pose.position.x;
        estimated_pos.y = msg.pose.position.y;
        estimated_pos.z = 0;

        waypoint_plant.x = current_pos.x + estimated_pos.x;
        waypoint_plant.y = current_pos.y + estimated_pos.y;
        waypoint_plant.z = current_pos.z + estimated_pos.z;

        nav_mode = NAV_MODE_FOUND_PLANT;
        ROS_INFO("Calculated [du, dv] as: [%0.2f, %0.2f]", estimated_pos.x, estimated_pos.y);
        ROS_INFO("Estimated plant location at: [%0.2f, %0.2f]", waypoint_plant.x,
        waypoint_plant.y);
        ROS_INFO_ONCE("Guiding center of the plant");
    }
}

void local_pos_cb(const geometry_msgs::PoseStamped msg){
    current_pos = msg.pose.position;

    double dHeight = 0;

    switch(nav_mode) {
        case NAV_MODE_TAKEOFF:
                ROS_INFO_ONCE("Guiding the UAV to takeoff");

                current_goal = waypoint_home;

                if((fabs(current_goal.x - current_pos.x) < waypoint_radius) &&
                   (fabs(current_goal.y - current_pos.y) < waypoint_radius) &&
                   (fabs(current_goal.z - current_pos.z) < waypoint_radius)) {
                        nav_mode = NAV_MODE_WAYPOINT_A;
```

```
                ROS_INFO("Guiding the UAV to waypoint A");
        }

        break;
case NAV_MODE_WAYPOINT_A:
        current_goal = waypoint_a;

        if((fabs(current_goal.x - current_pos.x) < waypoint_radius) &&
           (fabs(current_goal.y - current_pos.y) < waypoint_radius) &&
           (fabs(current_goal.z - current_pos.z) < waypoint_radius)) {
                nav_mode = NAV_MODE_WAYPOINT_B;
                ignore_ip = false;
                ROS_INFO("Guiding the UAV to waypoint B");
        }

        break;
case NAV_MODE_WAYPOINT_B:
        current_goal = waypoint_b;

        if((fabs(current_goal.x - current_pos.x) < waypoint_radius) &&
           (fabs(current_goal.y - current_pos.y) < waypoint_radius) &&
           (fabs(current_goal.z - current_pos.z) < waypoint_radius)) {
                ignore_ip = true;
                nav_mode = NAV_MODE_RTL;
                ROS_INFO("Guiding the UAV to home");
        }

        break;
case NAV_MODE_FOUND_PLANT:
        current_goal = waypoint_plant;

        if((fabs(current_goal.x - current_pos.x) < vs_radius) &&
           (fabs(current_goal.y - current_pos.y) < vs_radius) &&
           (fabs(current_goal.z - current_pos.z) < waypoint_radius)) {
                nav_mode = NAV_MODE_SPRAY_DESC;
                ignore_ip = true;
                ROS_INFO("Identified the plant...");
        }

        break;
case NAV_MODE_SPRAY_DESC:
        current_goal = waypoint_plant;

        dHeight = height_us - height_spray;
        current_goal.z = current_pos.z - dHeight;

        if((fabs(current_goal.x - current_pos.x) < vs_radius) &&
           (fabs(current_goal.y - current_pos.y) < vs_radius) &&
           (fabs(current_goal.z - current_pos.z) < vs_radius)) {
                nav_mode = NAV_MODE_SPRAY_WAIT;
                start_spray_time = ros::Time::now();
                ROS_INFO("Reached spray height, starting spray timer...");
        }

        break;
case NAV_MODE_SPRAY_WAIT:
        current_goal = waypoint_plant;

        dHeight = height_us - height_spray;
        current_goal.z = current_pos.z - dHeight;

        if((ros::Time::now() - start_spray_time).toSec() > 5.0) {
                nav_mode = NAV_MODE_SPRAY_ASC;
                ROS_INFO("Finished waiting for spray, returning to waypoint
                height...");
        }

        break;
case NAV_MODE_SPRAY_ASC:
        current_goal = waypoint_plant;
```

```
                if((fabs(current_goal.x - current_pos.x) < waypoint_radius) &&
                   (fabs(current_goal.y - current_pos.y) < waypoint_radius) &&
                   (fabs(current_goal.z - current_pos.z) < waypoint_radius)) {
                        nav_mode = NAV_MODE_WAYPOINT_B;
                        ROS_INFO("Reached waypoint height, continuing to waypoint B.");
                }

                break;
        case NAV_MODE_RTL:
                current_goal = waypoint_home;

                if((fabs(current_goal.x - current_pos.x) < waypoint_radius) &&
                   (fabs(current_goal.y - current_pos.y) < waypoint_radius) &&
                   (fabs(current_goal.z - current_pos.z) < waypoint_radius)) {
                        //land
                        ROS_INFO_ONCE("Reached Home");

                        land_mode = true;
                }

                break;
        default:
            ROS_ERROR("Navigation mode unknown");
            nav_mode = NAV_MODE_RTL;
            ROS_INFO("Guiding the UAV to home");
    }
}

//=================================//
// Main Function                   //
//=================================//
int main(int argc, char **argv)
{
    //=================================//
    // Initialize node                 //
    //=================================//
    ros::init(argc, argv, "spray_nav");
    ros::NodeHandle nh(ros::this_node::getName());

    double fov_deg = 60;

    //Parameters
    if(!nh.getParam("height_takeoff", height_takeoff)){
        ROS_WARN("No parameter set for \"height_takeoff\"");
    }
    ROS_INFO("Setting takeoff height to: %0.2f", height_takeoff);

    if(!nh.getParam("height_search", height_search)){
        ROS_WARN("No parameter set for \"height_search\"");
    }
    ROS_INFO("Setting search height to: %0.2f", height_search);

    if(!nh.getParam("height_spray", height_spray)){
        ROS_WARN("No parameter set for \"height_spray\"");
    }
    ROS_INFO("Setting spray height to: %0.2f", height_spray);

    if(!nh.getParam("waypoint_radius", waypoint_radius)){
        ROS_WARN("No parameter set for \"waypoint_radius\"");
    }
    ROS_INFO("Setting waypoint radius to: %0.2f", waypoint_radius);

    if(!nh.getParam("vs_radius", vs_radius)){
        ROS_WARN("No parameter set for \"vs_radius\"");
    }
    ROS_INFO("Setting visual servoing radius to: %0.2f", vs_radius);

    if(!nh.getParam("fov_degrees", fov_deg)){
        ROS_WARN("No parameter set for \"fov_degrees\"");
    }
    ROS_INFO("Setting camera width FoV to: %0.2f", fov_deg);
```

```
fov = fov_deg*M_PI/180.0;

if(!nh.getParam("camera_width", camera_width)){
    ROS_WARN("No parameter set for \"camera_width\"");
}
ROS_INFO("Setting camera width: %0.2f", camera_width);

if(!nh.getParam("camera_height", camera_height)){
    ROS_WARN("No parameter set for \"camera_height\"");
}
ROS_INFO("Setting camera height: %0.2f", camera_height);

std::string position_input = "/pose";
if(!nh.getParam("position_input", position_input)){
    ROS_WARN("No parameter set for \"position_input\"");
}
ROS_INFO("Setting position input to: %s", position_input.c_str());

std::string vs_input = "/target_location";
if(!nh.getParam("vs_input", vs_input)){
    ROS_WARN("No parameter set for \"vs_input\"");
}
ROS_INFO("Setting visual servoing input to: %s", vs_input.c_str());

fov = fov_deg*M_PI/180.0;

//Subscribers
ros::Subscriber ip_sub = nh.subscribe<geometry_msgs::PoseStamped>
    (vs_input, 10, ip_cb);
ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
    ("/mavros/state", 10, state_cb);
ros::Subscriber local_pos_sub = nh.subscribe<geometry_msgs::PoseStamped>
    (position_input, 10, local_pos_cb);
ros::Subscriber us_sub = nh.subscribe<std_msgs::Float64>
    ("/us_distance", 10, us_cb);

//Publishers
ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>
    ("/mavros/setpoint_position/local", 10);

//Services
ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>
    ("/mavros/cmd/arming");
ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>
    ("/mavros/set_mode");

//the setpoint publishing rate MUST be faster than 2Hz
ros::Rate rate(20.0);

// wait for FCU connection
while(ros::ok() && current_state.connected){
    ros::spinOnce();
    rate.sleep();
}

//================================//
// Initialize setpoint sequence   //
//================================//
//Set the takeoff goal
waypoint_home.x = 0;
waypoint_home.y = 0;
waypoint_home.z = height_takeoff;

waypoint_a.x = -5;
waypoint_a.y = 2;
waypoint_a.z = height_takeoff;

waypoint_b.x = 5;
waypoint_b.y = 2;
waypoint_b.z = height_search;
```

```
current_goal = waypoint_home;

//Prepare the pose message
geometry_msgs::PoseStamped pose;
pose.header.frame_id = "/fcu";
pose.pose.position = current_goal;

//send a few setpoints before starting
ROS_INFO("Initializing pose stream...");
for(int i = 100; ros::ok() && i > 0; --i){
    pose.header.stamp = ros::Time::now();
    local_pos_pub.publish(pose);
    ros::spinOnce();
    rate.sleep();
}
ROS_INFO("Done!");


//================================//
// Set Parameters                 //
//================================//
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";
mavros_msgs::SetMode land_set_mode;
land_set_mode.request.custom_mode = "AUTO.LAND";


mavros_msgs::CommandBool arm_cmd;
arm_cmd.request.value = true;

ros::Time last_request = ros::Time::now();


//================================//
// Main Loop                      //
//================================//
while(ros::ok()){

    //================================//
    // Update Callbacks               //
    //================================//
    ros::spinOnce();

    //================================//
    // Set Mode                       //
    //================================//

    if( !land_mode) {
        if( current_state.mode != "OFFBOARD" &&
            current_state.mode == "AUTO.LOITER" &&
            (ros::Time::now() - last_request > ros::Duration(5.0))){

            ROS_INFO("Current mode is not \"OFFBOARD\" [%s]", current_state.mode.c_str());
            if( set_mode_client.call(offb_set_mode) &&
                offb_set_mode.response.success){
                ROS_INFO("Offboard enabled");
            }

            last_request = ros::Time::now();
        } else {
            if( !current_state.armed &&
                (ros::Time::now() - last_request > ros::Duration(5.0))){

                if( arming_client.call(arm_cmd) &&
                    arm_cmd.response.success){
                    ROS_INFO("Vehicle armed");
                }

                last_request = ros::Time::now();
            }
```

```
            }
        } else {
            if( set_mode_client.call(land_set_mode) && land_set_mode.response.success) {
                ROS_INFO("Landing...");
                break;
            }
        }

        //==================================//
        // Send setpoints                   //
        //==================================//
        pose.header.stamp = ros::Time::now();
        pose.pose.position = current_goal;

        local_pos_pub.publish(pose);


        //==================================//
        // Sleep                            //
        //==================================//
        rate.sleep();
    }

    return 0;
}
```

B.4 Arduino code for controlling the Ultrasonic (HC-SR04) and the spraying pump.

Algorithm B.10 Arduino code for controlling the Ultrasonic (HC-SR04) and the spraying pump.

```
/*
 HC-SR04 Ping distance sensor:
 VCC to arduino 5v
 GND to arduino GND
 Echo to Arduino pin 7
 Trig to Arduino pin 8
 */
#define echoPin 7 // Echo Pin
#define trigPin 8 // Trigger Pin
#define sprayPin 3 // Spray Trigger Pin
#define LEDPin 13 // Onboard LED

int maximumRange = 200; // Maximum range needed
int minimumRange = 0; // Minimum range needed
long duration, distance; // Duration used to calculate distance
bool sprayTrig = 0;
long sprayTime = 0;

void setup() {
 Serial.begin (115200);
 pinMode(trigPin, OUTPUT);
 pinMode(sprayPin, OUTPUT);
 digitalWrite(sprayPin, HIGH);
 pinMode(echoPin, INPUT);
 pinMode(LEDPin, OUTPUT); // Use LED indicator (if required)
}

void loop() {
/* The following trigPin/echoPin cycle is used to determine the
 distance of the nearest object by bouncing soundwaves off of it. */
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);

 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);

 digitalWrite(trigPin, LOW);
 duration = pulseIn(echoPin, HIGH);

 //Calculate the distance (in cm) based on the speed of sound.
 distance = duration/58.2;

 if (distance >= maximumRange || distance <= minimumRange){
 /* Send a negative number to computer and Turn LED ON
 to indicate "out of range" */
 Serial.println("-1");
 }
 else {
 /* Send the distance to the computer using Serial protocol, and
 turn LED OFF to indicate successful reading. */
 Serial.println(distance);
 }
 if((distance>40)&&(distance<60)&&!sprayTrig)
 {
    sprayTrig = 1;
    sprayTime = millis();
    digitalWrite(sprayPin, LOW);
    digitalWrite(LEDPin, HIGH);
 }

 if(sprayTrig&&((sprayTime+2000)<millis()))
 {
    sprayTrig = 0;
    sprayTime = millis();
    digitalWrite(sprayPin, HIGH);
    digitalWrite(LEDPin, LOW);
 }

 //Delay 50ms before next reading.
 delay(50);
}
```

B.5 Matlab Code for drawing flight trajectory.

Algorithm B.11 Matlab Code for drawing flight trajectory.

```
%% Pre-Script

close all;

clear;
clc;


%% Get 2 lots of data and plot

disp('Select the file containing the pointer position data');
pather
pather

%%
% generated from the file name
x = str2double(data.simm3m.field_pose_position_x);
y = str2double(data.simm3m.field_pose_position_y);
z = str2double(data.simm3m.field_pose_position_z);



plot3(x,y,z, '+-');
axis('equal');
hold on;

%%

% generated from the file name
x2 = str2double(data.flight1.field_pose_position_x);
y2 = str2double(data.flight1.field_pose_position_y);
z2 = str2double(data.flight1.field_pose_position_z);

plot3(x2,y2,z2, '--');

 %%
grid on
 waypoints = [0, 0, 0; ...
              0, 0, 3; ...
              -5, 2, 3; ...
              5, 2, 3; ...
              0, 0, 3; ...
              0, 0, 0];

line(waypoints(:,1), waypoints(:,2), waypoints(:,3), 'color', 'g');

hold off;
```

```
%%
% File Read

[file,path,~] = uigetfile('*.csv','Read Recorded Data');
filePath = [path, file];

if file == 0
    error('Please select a file to analyse.');
end

fid = fopen(filePath,'r'); %Open specified file
C = textscan(fid, repmat('%s',1,12), 'delimiter',',', 'CollectOutput',true); %scan↙
read data and format
C = C{1}; %Knock the cell table down a level
C{1,1} = strrep(C{1,1},'%',''); %Take out the comment if it exists
fclose(fid);


%% Phrase Data

vname = matlab.lang.makeValidName(file);
vname = strrep(vname,'_csv',''); %Take out the comment if it exists

for i = 1:size(C,2)
    tname = matlab.lang.makeValidName(C{1,i});
    data.(vname).(tname) = cell(size(C,1)-1,1);
    for j = 2:size(C,1)
        data.(vname).(tname){j-1} = C{j,i};
    end
end

%% Clean Up
clearvars vname tname i j C fid file path filePath ans
```

References

[1] H. Choi, M. Geeves, B. Alsalam, and F. Gonzalez, "Open source computer-vision based guidance system for UAVs on-board decision making," *IEEE Aerospace Conferece, Big Sky, Montana*, 2016.

[2] S. L. Ward, J. Hensler, B. H. Y. Alsalam, and L. F. Gonzalez, "Autonomous UAVs wildlife monitoring and tracking using thermal imaging and computer vision," *IEEE Aerospace conferece, Big Sky, Montana*, 2016.

[3] Y. Han, "An autonomous Unmanned Aerial Vehicle-based imagery system development and remote sensing images classification for agricultural applications," *Graduate Theses and Dissertations*, p. 513, 2009.

[4] H. Xiang and L. Tian, "Method for automatic georeferencing aerial remote sensing (RS) images from an unmanned aerial vehicle (UAV) platform," *Biosystems Engineering*, vol. 108, no. 2, pp. 104–113, 2011.

[5] E. R. Hunt Jr, M. Cavigelli, C. S. Daughtry, J. E. Mcmurtrey III, and C. L. Walthall, "Evaluation of digital photography from model aircraft for remote sensing of crop biomass and nitrogen status," *Precision Agriculture*, vol. 6, no. 4, pp. 359–378, 2005.

[6] M. Rieke, T. Foerster, J. Geipel, and T. Prinz, "High-precision positioning and real-time data processing of uav systems," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, pp. 1–C22, 2011.

[7] C. C. D. Lelong, P. Burger, G. Jubelin, B. Roux, S. Labbe, and F. Baret, "Assessment of Unmanned Aerial Vehicles imagery for quantitative monitoring of wheat crop in small plots," *Sensors*, vol. 8, no. 5, pp. 3557–3585, 2008.

[8] V. Gonzalez-Dugo, P. Zarco-Tejada, E. Nicolas, P. A. Nortes, J. J. Alarcon, D. S. Intrigliolo, and E. Fereres, "Using high resolution UAV-thermal imagery to assess the variability in the water status of five fruit tree species within a commercial orchard," *Precision Agriculture*, vol. 14, no. 6, pp. 660–678, 2013.

[9] L. Felderhof, D. Gillieson, P. Zadro, and A. Van Boven, "Linking UAV(Unmanned Aerial Vehicle) technology with precision agriculture," 2008.

[10] S. Von Bueren and I. Yule, "Multispectral aerial imaging of pasture quality and biomass using unmanned aerial vehicles (UAV)," *Accurate and Efficient Use of Nutrients on Farms, Occasional Report*, no. 26, 2013.

[11] N. Hallermann and G. Morgenthal, "Visual inspection strategies for large bridges using unmanned aerial vehicles (UAV)," in *Proc. 7th International Conference on Bridge Maintenance, Safety and Management, IABMAS 2014, July 7, 2014-July 11*, 2014.

[12] F. Chaumette and S. Hutchinson, "Visual servo control image basic approaches," *IEEE Robotics Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[13] P. I. Corke, *Visual Control of Robots: high-performance visual servoing*. Baldock, UK: Research Studies Press, 1996.

[14] H. H. A. Kadouf and Y. M. Mustafah, "Colour-based object detection and tracking for autonomous quadrotor UAV," in *Proc. IOP Conference Series: Materials Science and Engineering*, vol. 53. IOP Publishing, 2013, p. 012086.

[15] C. Knoth, B. Klein, T. Prinz, and T. Kleinebecker, "Unmanned aerial vehicles as innovative remote sensing platforms for high-resolution infrared imagery to support restoration monitoring in cut-over bogs," *Applied Vegetation Science*, vol. 16, no. 3, pp. 509–517, 2013.

[16] E. Hunt, W. D. Hively, C. S. Daughtry, G. W. McCarty, S. J. Fujikawa, T. Ng, M. Tranchitella, D. S. Linden, and D. W. Yoel, "Remote sensing of crop leaf area index using unmanned airborne vehicles," in *Proceedings of the Pecora 17 Symposium, Denver, CO*, 2008.

[17] D. Gomez-Candon, F. Lopez Granados, J. Caballero Novella, M. Gomez Fmez Casero, M. Jurado Exposito, and L. Garca Torres, "Geo-referencing remote images for precision agriculture using artificial terrestrial targets," *Precision Agriculture*, vol. 12, no. 6, pp. 876–891, 2011.

[18] J. Berni, P. Zarco-Tejada, L. Suarez, V. Gonzalez-Dugo, and E. Fereres, "Remote sensing of vegetation from uav platforms using lightweight multispectral and thermal imaging sensors," *Int. Arch. Photogramm. Remote Sens. Spatial Inform. Sci*, vol. 38, p. 6, 2009.

[19] C. Hung, Z. Xu, and S. Sukkarieh, "Feature learning based approach for weed classification using high resolution aerial images from a digital camera mounted on a uav," *Remote Sensing*, vol. 6, no. 12, pp. 12 037–12 054, 2014.

[20] E. Honkavaara, H. Saari, J. Kaivosoja, I. Polonen, T. Hakala, P. Litkey, J. Makynen, and L. Pesonen, "Processing and assessment of spectrometric, stereoscopic imagery collected using a lightweight uav spectral camera for precision agriculture," *Remote Sensing*, vol. 5, no. 10, pp. 5006–5039, 2013.

[21] S. Candiago, F. Remondino, M. De Giglio, M. Dubbini, and M. Gattelli, "Evaluating multispectral images and vegetation indices for precision farming applications from UAV images," *Remote Sensing*, vol. 7, no. 4, pp. 4026–4047, 2015.

[22] A. S. Laliberte, A. Rango, K. M. Havstad, J. F. Paris, R. F. Beck, R. McNeely, and A. L. Gonzalez, "Object-oriented image analysis for mapping shrub encroachment from 1937 to 2003 in southern new mexico," *Remote Sensing of Environment*, vol. 93, no. 1, pp. 198–210, 2004.

[23] S. Herwitz, L. Johnson, S. Dunagan, R. Higgins, D. Sullivan, J. Zheng, B. Lobitz, J. Leung, B. Gallmeyer, and M. Aoyagi, "Imaging from an unmanned aerial vehicle: agricultural surveillance and decision support," *Computers and Electronics in Agriculture*, vol. 44, no. 1, pp. 49–61, 2004.

[24] S. R. Herwitz, L. F. Johnson, J. Arvesen, R. Higgins, J. Leung, and S. Dunagan, "Precision agriculture as a commercial application for solar-powered Unmanned Aerial Vehicles," in *AIAA 1st Technical Conference and Workshop on Unmanned Aerospace Vehicles*, 2002, Conference Proceedings.

[25] M. Kontitsis, K. P. Valavanis, and N. Tsourveloudis, "A UAV-vision system for airborne surveillance," in *Proc. Robotics and Automation, 2004. ICRA'04. 2004 IEEE International Conference on*, vol. 1, 2004, pp. 77–83.

[26] C. H. Koger, D. R. Shaw, C. E. Watson, and K. N. Reddy, "Detecting late-season weed infestations in soybean (glycine max) 1," *Weed Technology*, vol. 17, no. 4, pp. 696–704, 2003.

[27] J. M. Pena Barragan, F. Lopez Granados, M. Jurado Exposito, and L. Garcia Torres, "Mapping ridolfia segetum patches in sunflower crop using remote sensing," *Weed Research*, vol. 47, no. 2, pp. 164–172, 2007.

[28] A. I. de Castro, M. Jurado Exposito, J. M. Pena Barragan, and F. Lopez Granados, "Airborne multi spectral imagery for mapping cruciferous weeds in cereal and legume crops," *Precision Agriculture*, vol. 13, no. 3, pp. 302–321, 2012.

[29] A. S. Laliberte, E. L. Fredrickson, and A. Rango, "Combining decision trees with hierarchical object-oriented image analysis for mapping arid rangelands," *Photogrammetric engineering & Remote sensing*, vol. 73, no. 2, pp. 197–207, 2007.

[30] F. Lopez-Granados, "Weed detection for site-specific weed management: mapping and real-time approaches," *Weed Research*, vol. 51, no. 1, pp. 1–11, 2011.

[31] M. Perez-Ortiz, J. M. Pena, P. A. Gutierrez, J. Torres-Sanchez, C. Hervas-Martinez, and F. Lopez-Granados, "Selecting patterns and features for between-and within-crop-row weed mapping using UAV-imagery," *Expert Systems with Applications*, vol. 47, pp. 85–94, 2016.

[32] M. Kelly, "Weed mapping in early-season maize fields using object-based analysis of unmanned aerial vehicle (UAV) images," *PLOS One*, vol. 8, no. 10, 2013.

[33] J. Torres-Sanchez, F. Lopez-Granados, and J. Pena, "An automatic object-based method for optimal thresholding in uav images: Application for vegetation detection in herbaceous crops," *Computers and Electronics in Agriculture*, vol. 114, pp. 43–52, 2015.

[34] A. S. Laliberte and A. Rango, "Texture and scale in object-based analysis of subdecimeter resolution unmanned aerial vehicle (UAV) imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 3, pp. 761–770, 2009.

[35] J. M. Pena, J. Torres Sanchez, A. I. de Castro, M. Kelly, and F. Lopez Granados, "Weed mapping in early-season maize fields using object-based analysis of unmanned aerial vehicle (UAV) images," *PLoS One*, vol. 8, no. 10, p. e77151, 2013.

[36] C. Zhang and J. M. Kovacs, "The application of small unmanned aerial systems for precision agriculture: a review," *Precision Agriculture*, vol. 13, no. 6, pp. 693–712, 2012.

[37] E. R. Hunt, W. D. Hively, S. J. Fujikawa, D. S. Linden, C. S. T. Daughtry, and G. W. McCarty, "Acquisition of nir-green-blue digital photographs from unmanned aircraft for crop monitoring," *Remote Sensing*, vol. 2, no. 1, pp. 290–305, 2010.

[38] A. S. Laliberte, J. E. Herrick, A. Rango, and C. Winters, "Acquisition, orthorectification, and object-based classification of unmanned aerial vehicle (UAV) imagery for rangeland monitoring," *Photogrammetric Engineering & Remote Sensing*, vol. 76, no. 6, pp. 661–672, 2010.

[39] J. A. Thomasson, Y. Shi, J. Olsenholler, J. Valasek, S. C. Murray, and M. P. Bishop, "Comprehensive UAV-agricultural remote-sensing research at texas a and m university," in *SPIE Commercial+ Scientific Sensing and Imaging*. International Society for Optics and Photonics, 2016, pp. 986 602–986 602.

[40] A. S. Laliberte, M. A. Goforth, C. M. Steele, and A. Rango, "Multispectral remote sensing from unmanned aircraft: Image processing workflows and applications for rangeland environments," *Remote Sensing*, vol. 3, no. 11, pp. 2529–2551, 2011.

[41] E. Salami, C. Barrado, and E. Pastor, "Uav flight experiments applied to the remote sensing of vegetated areas," *Remote Sensing*, vol. 6, no. 11, pp. 11 051–11 081, 2014.

[42] R. Dunford, K. Michel, M. Gagnage, H. Piegay, and M. L. Tremelo, "Potential and constraints of unmanned aerial vehicle technology for the characterization of mediterranean riparian forest," *International Journal of Remote Sensing*, vol. 30, no. 19, pp. 4915–4935, 2009.

[43] A. Mcfadyen, F. Gonzalez, D. Campbell, and D. Eagling, "Evaluating unmanned aircraft systems for deployment in plant biosecurity," Tech. Report, Canberra, Australia, Tech. Rep., 2014.

[44] E. Puig, F. Gonzalez, G. Hamilton, and G. P, "Assessment of crop insect damage using unmanned aerial systems: A machine learning approach," *Proc. 21st International Congress on Modelling and Simulation*, pp. 1420–1426, 2015.

[45] S. Nebiker, A. Annen, M. Scherrer, and D. Oesch, "A light-weight multispectral sensor for micro uav opportunities for very high resolution airborne remote sensing," *The international archives of the photogrammetry, remote sensing and spatial information sciences*, vol. 37, pp. 1193–1200, 2008.

[46] J. A. Berni, P. J. Zarco-Tejada, L. Suarez, and E. Fereres, "Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 47, no. 3, pp. 722–738, 2009.

[47] T. Guo, T. Kujirai, and T. Watanabe, "Mapping crop status from an unmanned aerial vehicle for precision agriculture applications," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 1, pp. 485–490, 2012.

[48] M. Bryson, A. Reid, F. Ramos, and S. Sukkarieh, "Airborne vision based mapping and classification of large farmland environments," *Journal of Field Robotics*, vol. 27, no. 5, pp. 632–655, 2010.

[49] Y. Fan, S. Haiqing, and W. Hong, "A vision-based algorithm for landing Unmanned Aerial Vehicles," in *Computer Science and Software Engineering, Proc. 2008 International Conference on*, vol. 1, 2008, pp. 993–996.

[50] S. A. Quintero and J. P. Hespanha, "Vision-based target tracking with a small UAV: Optimization-based control strategies," *Control Engineering Practice*, vol. 32, pp. 28–42, 2014.

[51] C. Anderson, "3DR-robotic," 2009. [Online]. Available: http://dev.ardupilot.com/

[52] A. Mcfadyen, "Visual control for automated aircraft collision avoidance systems," 2015.

[53] Y.-c. Liu and Q.-h. Dai, "Vision aided unmanned aerial vehicle autonomy: an overview," in *Image and Signal Processing (CISP), 2010 3rd International Congress on*, vol. 1, 2010, pp. 417–421.

[54] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme, "Vision-based autonomous landing of an unmanned aerial vehicle," in *Robotics and automation, 2002. Proceedings. ICRA'02. IEEE international conference on*, vol. 3, 2002, pp. 2799–2804.

[55] V. M. Raja, "Vision based landing for unmanned aerial vehicle," in *Prod. Aerospace Conference, 2011 IEEE.* IEEE, 2011, pp. 1–8.

[56] C. Fu, A. Carrio, M. Olivares-Mendez, and P. Campoy, "Online learning-based robust visual tracking for autonomous landing of Unmanned Aerial Vehicles," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, 2014, pp. 649–655.

[57] R. Mahony and T. Hamel, "Image-based visual servo control of aerial robotic systems using linear image features," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 227–239, 2005.

[58] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart, "Onboard imu and monocular vision based control for mavs in unknown in-and outdoor environments," in *Robotics and automation (ICRA), 2011 IEEE international conference on*, 2011, pp. 3056–3063.

[59] N. Veerasamy, "High-level mapping of cyberterrorism to the ooda loop," in *Proceedings of the 5th International Conference on Information Warfare and Security*, 2010, pp. 352–360.

[60] D. Maccuish, "Orientation: key to the ooda loop the culture factor," *Journal of Defense Resources Management (JoDRM)*, no. 02, pp. 67–74, 2012.

[61] J. B. Pullen, "The committee to abolish hell: Strategic culture, OODA-loops, and decision-making by the us national security council during the bosnian war," M.A. thesis, Univ. North Carolina, Chapel Hill, NC„ 2014.

[62] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 30, no. 3, pp. 286–297, 2000.

[63] H. Peng, M. Huo, Z. Liu, and Y. He, "Challenges and technologies for networked multiple UAVs cooperative control," in *Electrical and Control Engineering (ICECE), 2011 International Conference on*, 2011, pp. 3860–3863.

[64] 3DRobotics, "3dr-drone and UAV-technology," 2015. [Online]. Available: http://3drobotics.com/

[65] pixhawk.org, "Pixhawk autopilot," 2015-04-19 2015. [Online]. Available: https://pixhawk.org/modules/pixhawk

[66] raspberrypi.org, "Raspberry pi 2 - model B," April,10,2015 2015. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b

[67] A. D. Team, "Mission planner overview," 2016. [Online]. Available: http://ardupilot.org/planner/docs/mission-planner-overview.html

[68] Modmypi, "HC-SR04 ultrasonic range sensor on the Raspberry Pi," 2016. [Online]. Available: http://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi

[69] Logitech, "HD-webcam logitech C270." [Online]. Available: http://support.logitech.com/en_us/article/17556

[70] QGroundControl, "Ground contral station for small air, land, water autonomous unmanned system," 2016. [Online]. Available: http://qgroundcontrol.org/mavlink/mavproxy_startpage

[71] PUTTY, "Download putty - a free ssh and telnet client for windows," 2016. [Online]. Available: http://www.putty.org/

[72] A. Dev, "Communicating with raspberry pi via mavlink," 2016. [Online]. Available: http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html

[73] Odroid. [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275

[74] Arduino, "Arduino / genuino uno," 2016. [Online]. Available: https://www.arduino.cc/en/guide/introduction

[75] Von, "Ros.org," 2016. [Online]. Available: http://wiki.ros.org/mavros

[76] M. Tonnis, *Darstellung virtueller Objekte.* Springer, 2010, pp. 7–41.

[77] O. Araar and N. Aouf, "Visual servoing of a quadrotor uav for the tracking of linear structured infrastructures," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 3310–3315.

[78] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel, and L. Eck, "Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle," *IEEE Trans. on Robotics*, vol. 25, no. 3, pp. 743–749, 2009.

[79] ROS.com, "rviz," 2016. [Online]. Available: http://wiki.ros.org/rviz

[80] C. A. Weeds, "National weeds strategy," 2016. [Online]. Available: http://www.weeds.org.au/cgi-bin/weedident.cgi?tpl=plant.tpl&state=&s=&ibra=all&card=H71

[81] HerbiGuide, "Spear thistle (cirsium vulgare)," 2015. [Online]. Available: http://www.herbiguide.com.au/Descriptions/hg_Spear_Thistle.htm

[82] M. Zurn, A. McFadyen, S. Notter, A. Heckmann, K. Morton, and L. F. Gonzalez, "Mpc controlled multirotor with suspended slung load: System architecture and visual load detection," 2016.

[83] opencv.org, "Opencv," 2015. [Online]. Available: http://opencv.org/

[84] G. Bradski *et al.*, "The opencv library," *Doctor Dobbs Journal*, vol. 25, no. 11, pp. 120–126, 2000.

[85] P. Mihelich, "CVbridge tutorials using cvbridge to convert between rosimages and open cv images." [Online]. Available: http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages