VISION BASED AUTOMATED LANDING OF QUADROTORS

by

Mustafa Mete

B.S., Electrical and Electronics Engineering, Bogazici University, 2016

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electroncis Engineering
Boğaziçi University
2018

ACKNOWLEDGEMENTS

I firstly would like to express my gratitude to my supervisor, Professor H. Işıl Bozma for all of her support, supervision, and assistance during this thesis. She has provided me a great environment with regard to facilities and network and encouraged me throughout all this period. Also, she has taught me how to overcome numerous obstacles I have been facing throughout my research.

Secondly, I would like to thank Professor Yağmur Denizhan and Assistant Professor Haluk Bayram for their participation in my thesis committee.

This work has been supported by BAP project 13800. Also, thanks to TUBITAK project EEEAG 115E380.

I also would like to convey my thanks to Kadir Türksoy, Berkan Höke, Doğan Patar, and Serhat İşcan who are the members of Intelligent Systems Laboratory (ISL) for their support and assistance. Additionally, I want to thank Alperen Gökçe, Berat Aktar, Mücahit Ekinci, İbrahim Özcan, Orhan Eren Akgün, and Burak Yıldızöz for their contributions to my work and helps during experiments. I would like to express my special thanks to Hüseyin Demirci for his extensive assistance during this work.

Lastly, I want to take this opportunity to express my heartiest gratitude to my parents Rafet and Yeşim Mete, my sister Şuheda Mete, my brothers Oğuzhan and Muhammed Yusuf Mete, and all of my friends for their unflagging support, encouragement, and trust throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

b	Thrust coefficient
d	The drift coefficient
e	Total position error
e_x, e_y, e_z	Position errors in x, y, and z directions
$F_i, i = 1, 2, 3, 4$	The thrust forces of the propellers
g	Gravity
h_l,h_h	Low and high altitude limits
I	Diagonal inertia matrix
I_x, I_y, and I_z	Diagonal elements of the inertia matrix
$K_{di}, i = 1, ..., 7$	Derivative Controller Coefficients
$K_{pi}, i = 1, ..., 7$	Proportional Controller Coefficients
L	Quadrotor arm length
l	Side length of an ArUco marker
m	Mass of quadrotor
M	Torque applied to the quadrotor body
M_G	Gyroscopic torques of the rotors
r	Quadrotor orientation
$R(\Omega)$	Rotation matrix
$u_i, i = 1, 2, 3, 4$	Invertable artificial system inputs
$\hat{x}_b, \hat{y}_b, \hat{z}_b$	Body fixed frames
$\hat{x}_i, \hat{y}_i, \hat{z}_i$	Inertial frame
$\dot{x_d}, \dot{y_d}, \dot{z_d}$	The desired speed values
Δ	Field of view angle (rad)
θ	Pitch angle (rad)
θ_d	The desired pitch angle (rad)
ϕ	Roll angle
ϕ_d	The desired roll angle (rad)

ψ	Yaw angle (rad)
ψ_d	The desired yaw angle (rad)
Ω	Quadrotor orientation (rad)
$\omega_i, i = 1, 2, 3, 4$	Motor speeds (rad/sec)

LIST OF ACRONYMS/ABBREVIATIONS

API	Application Programming Interface
AR	Augmented Reality
DGPS	Differential Global Positioning System
DoF	Degree of Freedom
ESC	Electronic Speed Controller
FSR	Force Sensitive Resistor
FoV	Field of View
FPS	force per second
GCS	Ground Control Stations
GPS	Global Positioning System
GUI	Graphical User Interface
IBVS	Image-Based Visual Servoing
IMU	Inertial Measurement Unit
LRF	Lazer Range Finder
PBVS	Position-Based Visual Servoing
PD	Proportional and Derivative
PID	Proportional Integral Derivative
RAM	Random Access Memory
RC	Radio Control
RGB-D	Red/Green/Blue-Depth
ROS	Robot Operating System
RPi3	Raspberry Pi 3
SDF	Simulation Descripton Format
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
URDF	Unified Robot Description Format
QVGA	Quad Ultra Extended Graphics Array

ABSTRACT

VISION BASED AUTOMATED LANDING OF QUADROTORS

This thesis focuses on the design and development of quadrotors that are capable of autonomous landing on a stationary platform. This problem is motivated by the fact that quadrotors have to land frequently to charge their batteries, as maximum flight time is rather short with the current state-of-art. In this work, we particularly consider three different quadrotor systems including the quadrotor constructed in our Intelligent Systems Laboratory, the ISL quadrotor. The ISL quadrotor is endowed with a camera attached to a gimbal stabilizing the view for visual sensing, a laser range sensor for measuring the altitude, PIXHAWK flight controller and a Raspberry Pi 3 (RPi3) as an onboard computer for autonomous operation. The software is designed and developed in Robot Operating System (ROS) and includes control, visual processing and serial communication with the flight controller. The code is designed to be multi-threaded as to have the quadrotor be capable of doing visual processing concurrently with flight control. The dynamic modeling of the quadrotor is based on one of the commonly used models. For the landing task, position, altitude, and velocity controllers based on proportional and derivative (PD) control are developed. Position information is provided by visual feedback through detecting the marker on the landing platform. The developed approaches are first tested on the Gazebo simulation environment. We also conduct similar landing experiments on the ISL quadrotor for different wind, light, and initial altitude conditions and observe that the quadrotor is able to land autonomously within approximately 80 cm error range on a stationary platform that is marked by a 50cm $\times$ 50cm ArUco marker. The designed quadrotor is also capable of tracking a platform that is moving with a slow linear velocity.

ÖZET

DÖRT PERVANELİ UÇAN ROBOTLARIN GÖRÜNTÜ TEMELLİ OTONOM İNİŞİ

Bu tez, sabit bir platforma otonom iniş yapabilme kabiliyetine sahip dört pervaneli uçan robotların (4-PUR) tasarım ve geliştirilmesine odaklanmaktadır. Son teknoloji ile mevcut maksimum uçuş süresi oldukça kısa olan 4-PUR'ların bataryalarını şarj etmek için sık sık iniş yapma zorunlulukları bulunmaktadır. Bu sebeple bu problem popülerliğini korumaktadır. Bu çalışmada, özellikle Akıllı Sistemler Laboratuvarımızda geliştirilen ISL 4-PUR'umuzun da dahil olduğu, üç farklı 4-PUR üzerinde durduk. ISL 4-PUR hareket halindeyken görüş alanı sorununu çözmek için bakış açısını sabitleyen gimbal'a bağlı bir kamera, daha hassas yükseklikleri ölçmek için bir lazer mesafe sensörü, PIXHAWK uçuş denetimcisi ve otonom operasyon için bir Raspberry Pi 3 (RPi3) yerleşik bilgisayarı bünyesinde barındırmaktadır. Yazılım Robot İşletim Sisteminde (ROS) tasarlanmış ve geliştirilmiş olup, denetim, görüntü işleme ve seri haberleşmeyi içerir. Kod 4-PUR'un uçuş denetimiyle görüntü işlemeyi eşzamanlı olarak yapabilmesi için çoklu-dizin kullanılarak tasarlanmıştır. 4-PUR'un dinamik modellemesi, yaygın olarak bilinen modellerden birine dayanmaktadır. İniş görevi için oransal ve türevsel (PD) denetimi yaklaşımını kullanan hız, konum ve yükseklik denetimcileri geliştirilmiştir. Görsel geri besleme, iniş platformundaki ArUco işaretinin tespitine dayalı göreceli konum bilgisi sağlar. Geliştirilen yaklaşımlar ilk olarak Gazebo simülasyon ortamında test edildi. Ayrıca, farklı rüzgar, ışık ve başlangıç irtifa koşulları için ISL 4-PUR'u üzerinde benzer iniş deneyleri başarıyla yapılmıştır ve 4-PUR'un sabit 50cm $\times$ 50cm ArUco işaretli bir platforma yaklaşık 80 cm hata aralığında otonom olarak iniş yapabildiği gözlenmiştir. Tasarlanan 4-PUR'un ayrıca yavaş lineer hız ile hareket eden bir platformu takip ettiği gözlenmiştir.

1. INTRODUCTION

Quadrotors are among the most popular types of unmanned aerial vehicles (UAVs). They are able to vertically take-off and land, do aggressive maneuvers and have a small size and relatively simple structure. Their potential applications can be summarized as shooting videos and photos, building inspection, search-and-rescue, surveillance, military operations, product delivery, routine security patrol, and even daily transportation.

In applications, one of the major problems encountered is due to their short flight times. One approach to overcome this problem is to have a coordination with a ground vehicle. Such a coordination enables several advantages as well. Ground vehicles typically have more payload capacity, battery life, and powerful computation units. Thus, they can carry the quadrotors to their operation sites and act as a mobile launching pad for them. During the task, quadrotors can land on the ground vehicle and recharge and continue with their missions. Thus, their operating range is expanded. This requires autonomous landing of quadrotors onto the ground vehicles.

While precise landing in indoors has been worked well [1], outdoor applications still have challenges. The position and velocity feedback of quadrotors in outdoor environments have low accuracy and update rate due to the wind, noise, and available sensors. A typical GPS sensor has approximately 3m accuracy at 10Hz which is not sufficient enough for precise autonomous landing. Moreover, other disturbances such as wind make autonomous tracking, hovering, and landing harder.

1.1. Related Literature

In general, there are three main considerations that are associated with the landing problem: landing platform, sensing, and control. Landing platforms can be divided into two sub-categories depending on the mobility and identification. In the former, stationary platforms, planar moving platforms (2 Degree of Freedom) [2], and ship

like platforms(4-6 DoF) [3, 4] are three main categories. In the latter, markers vary from being known [5] to unknown [3], or being obscure [3, 6], or no marker [7]. Prior knowledge of markers on the platform for landing with different shapes has also been studied [5, 8, 9].

Sensing is critical as it is used for estimating the relative position of the quadrotor with respect to the landing platform. Some commonly used sensors include:

- Global Positioning System(GPS): It provides global position feedback. The accuracy of GPS is in meters meaning that it is not so precise. It also consumes too much power.
- Differential GPS (The DGPS): More accurate than the GPS, but still not so accurate. It consumes too much power. [10]
- Inertial Measurement Unit (IMU): An IMU usually includes 3 axes accelerometer, 3 axes gyroscope, compass or magnetometer, barometric/temperature sensors. It utilizes integrators for giving position and velocity feedback. Hence, it leads too much error [11].
- Laser Range Finders (LRFs): They provide sensitive distance measurements utilizing laser beams.
- Red/Green/Blue-Depth (RGB-D) sensors: They combine an RGB color image with depth information. They are heavy and consumes high power.
- Vision (monocular, stereo camera): Monocular cameras are the most popular and inexpensive tools. Their popularity has several contributing factors: lightweight, low cost and providing rich knowledge about the environments [12]. Stereo cameras are also used, but their performance is reduced when the distance between quadcopter and landing pad is low. [1, 13].
- VICON Motion Capture Systems: It provides position estimation of the quadrotor whose sensitivity is around mm [14]. It has a high cost and typically applicable for indoor applications.

Due to its lightweight, lower cost, passiveness, compatibility with indoor and outdoor applications, the capability of operating in GPS-denied areas, and the ability to provide extra information about the surrounding environment, a monocular camera is preferred for the sensing.

Controllers are responsible for achieving the desired 3D position and velocity. The complexity of the quadrotor dynamics has led to the design and development of various different control methods including the proportional, integral, and derivative (PID) controller [15], feedback linearization [16], back-stepping control, and sliding mode control [17]. The success of the controller depends not only on the controller but also the sensor which provides state feedback.

The attitude control of the quadrotor is generally achieved by an onboard controller, namely a flight controller or an autopilot. Position control is generally done in two alternative ways: by a remote control unit or by an onboard controller. A remote controller has typically much more computational power compared to an onboard controller, but the communication between it and the quadrotor which is obtained telemetry, radio, or Wifi is slower than an onboard controller which utilizes serial connection. What is more, the onboard controller enables the autonomous behavior of the quadrotor.

Autonomous landing with an onboard monocular camera has been also studied [1]. Conventional control techniques have been used with visual feedback to obtain a safe and sensitive landing controller [18, 19], which is known as Visual Servoing in the literature. The proposed approaches are generally categorized into two: Position-Based Visual Servoing (PBVS) and Image-Based Visual Servoing (IBVS) [20].

In PBVS, the relative position is obtained and the error is calculated from the image obtained. Then, the controllers begin to work. On the other hand, in IBVS, the error is obtained directly from the image properties and is transmitted to the controller. Although PBVS is easier than IBVS in terms of calculations, it is more sensitive to calibration errors.

For the onboard vision landing techniques, robust and faster machine vision algorithms are necessary due to the limited power of onboard computers. The accuracy of state estimation for both the quadcopter and the landing platform needs to be improved. There is a trade-off between accuracy and speed.

1.2. General Approach and Contribution

This thesis focuses on the autonomous landing of quadrotors that are endowed with simple onboard processors. The goal is to have the quadrotor land by itself on a stationary platform as indicated by a known marker using visual feedback. The contributions of the thesis can be summarized as follows:

- First, three different quadrotor systems are considered. Two are commercially available while the last one was designed and assembled in ISL. As such, considerable expertise has been accrued regarding the development of quadrotor systems.
- A visual-feedback based control approach has been designed and implemented on the real quadrotor that has a low-power processor. In this approach, the conventional PD controller is integrated with visual processing as to enable the quadrotor to compute its relative planar position with respect to the landing platform.
- A thorough evaluation of landing performance has been done based on the experiments with a real quadrotor in outdoors with varying wind, light and initial altitude conditions.
- Two landing platforms have been designed and built.

1.3. Organization of Thesis

The general outline of the thesis is as follows: Chapter 2 presents three different quadrotor systems - namely AR.Drone, Erle-Copter, and ISL Quadrotor. The ISL Quadrotor setup, the hardware, and software solutions are described in detail. In Chapter 3, the landing control of the quadrotor is developed. First, the dynamic model of the quadrotor is explained. Following, the control structure of the quadrotor

is presented. Finally, a simple PD controller utilizing visual feedback is designed for position and velocity control. Visual processing is explained in Chapter 4. Simulation and experiment setups and results are given in Chapter 5. The thesis concludes with a brief summary along with an evaluation including future directions. The appendices present supplementary materials. The software that needs to be installed is listed in Appendix A. A user's guide to the developed software is provided in Appendix B. The landing platforms that have been designed and built are explained in Appendix C. Work that has been done with the force sensors is discussed in Appendix D.

2. QUADROTOR SYSTEMS

In this thesis, three different quadrotors have been considered - namely AR.Drone 2, Erle-Copter, and ISL Quadrotor. The first two of them are commercial quadrotors, while the third one is developed from scratch. AR.Drone 2 is not suitable for outdoor applications since it is highly affected by wind due to its lightweight. However, it is used for testing the developed algorithms due to its high stability in indoor. Erle-Copter has provided a better understanding of the quadrotors which has enabled us to develop our own quadrotor, ISL Quadrotor. In the rest of this chapter, each system is presented in detail.

Let it be noted that two landing platforms have also been designed and built. The first is a platform that is placed on top of an outdoors robot. The second is a stationary platform that can be moved vertically. Both have been designed as to eventually test landing on such platforms. They are explained in Appendix C. However, they have not been used in the experimental evaluation due to time constraints.

2.1. AR.Drone 2

AR.Drone 2 is a cheap commercial remote controlled quadrotor produced by Parrot company [21]. It is an upgraded version of the first AR.Drone series. AR stands for the Augmented Reality. It has two onboard monocular cameras: front and bottom cameras. The front camera has 720p resolution with 93 degrees lens, recording up to 30fps, while the bottom camera has QVGA sensor with 64 degrees lens, recording up to 60fps. It uses its bottom camera to stabilize itself in XY direction by optical flow. It has ultrasound and air pressure sensors allowing for more stable flight and hovering. Moreover, there are 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer which provides orientation, velocity, and acceleration data. The Wi-Fi hardware with 802.11n standard enables to create a Wi-Fi hotspot. AR.Drone needs 12V-8A power. Its original battery is a 3 cell LiPo 1500 mAh with mini-Tamiya connector. The existing batteries of AR.Drone only provide 5-10 minutes-flight time on average. With proper

connectors and sufficient power, quadrotors can fly with cables, which provides very long flight times. However, as the quadrotor ascends, the carried wire weight increases, which may disrupt the quadrotor's balance or cause the maximum payload limit to be exceeded. The total weight of AR.Drone is around 600 gr with its battery and hull. Due to its lightweight, it can be drifted by the wind. Hence, it is not suitable for outdoor applications.

Figure 2.1. AR.Drone 2

The product is originally designed to be controlled by Android and IOS devices. An API is provided which enables communication with the quadrotor over Wi-Fi. Thus, the quadrotor states and onboard sensor measurements can be transmitted to the control device via Wi-Fi communication [22, 23]. Moreover, several ROS packages have been developed for enabling communication with AR.Drone such as ardrone_autonomy and tum_ardrone packages. They enable the reading and transmission of state information, video stream, and the control of AR.Drone with a keyboard.

2.2. Erle-Copter

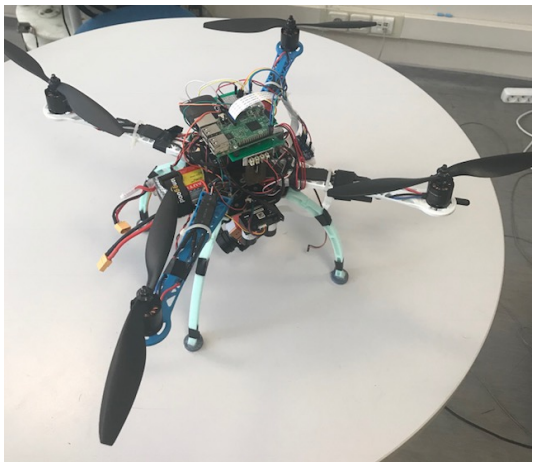
Erle-Copter is a commercial assembled quadrotor with an onboard processor. It is relatively cheap compared to its rivals. All of its components and their features are presented in Table 2.1. Its flight controller, Erle Brain 3, utilizes ArduPilot software. Its onboard processor is Raspberry Pi 3 (RPi3) with a Raspbian operating system image with necessary pre-built ROS packages. Erle-Copter has Gazebo simulation model and ready-to-use ROS/Gazebo simulation launch file compatible with ROS Indigo. The

available MAVROS package is a ROS communication driver for MAVLink which is a communication protocol for UAVs. In addition, MAVROS provides MAVLink bridge for the ground control station. Using this protocol, specific commands can be sent to the quadrotor utilizing ROS.

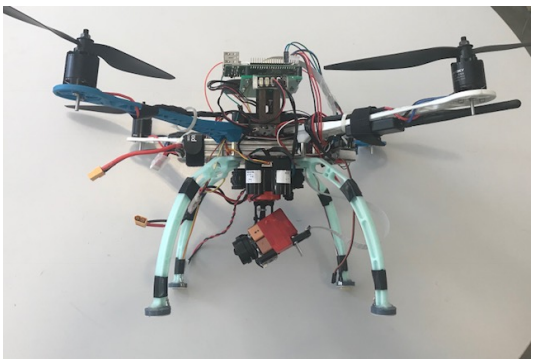
Table 2.1. Main components of Erle-Copter

Component	Quantity	Model	Function
Quadrotor	1	Erle-Copter drone kit	-
Flight controller	1	Erle Brain 3	Controls the quadrotor attitude
RC Receiver and Transmitter	1	Turnigy 9X and FrSky D4R-II	Enables remote control
Telemetry	1	-	Provides communication
Global Positioning System (GPS)	1	uBlox Neo-8M	Provides global position feedback
LiPo Battery	1	Turnigy 3S 5200mAh 10C	Supplies power
Onboard Computer	1	RPi3	Image processing and high level control
Onboard Camera	1	Rpi Camera v2	Records video, front

Figure 2.2. Erle-Copter



(a) Top View



(b) Side View

Figure 2.3. The ISL Quadrotor

2.3. The ISL Quadrotor

ISL Quadrotor is a quadrotor that has been self-designed and implemented as shown in Figure 2.3. The specifications of the components utilized in the ISL Quadrotor is summarized in Table 2.2. The ISL quadrotor consists of a quadrotor frame with 250 mm frame size, 4 x motors, 4 x Electronic Speed Controllers (ESCs) to drive the motors, 4 x 1045 propellers (1045 stands for 10 inches in length and 4.5 inches in pitch), a PIXHAWK flight controller, power distribution board to obtain different voltage values (11.1V and 5V), RC receiver and transmitter, telemetry, a laser range finder, Global Positioning System (GPS), 3C LiPo batteries, landing gears, gimbal, a RPi3 onboard computer, and a RPi3 camera v2 onboard camera. In addition to these, force sensitive resistors are added to legs for measuring the success rate of the landing. All electrical and mechanical connections of the hardware are illustrated in Figure 2.4. The whole system is around 2.5 kilograms.

2.3.1. Flight Controller: PIXHAWK

Flight controllers can be described as the primary context of a quadrotor. They have sensors for detecting its states and generate the required Pulse Width Modulation (PWM) outputs for motors depending on the desired flight mission. As such, they are responsible for the low-level control of a quadrotor.

Table 2.2. Main components of ISL Quadrotor

Component	Quantity	Model	Function
Quadrotor Frame	1	S500	Skeleton of the quadrotor
Propeller	4	1045, Carbonfiber	Creates thrust
Motor	4	SunnySky X2216 1250KV Outrunner Brushless	Drive propellers
ESC	4	HolyBro Tekko32 35A	Drives motors
Flight controller	1	PIXHAWK	Controls the quadrotor
Power Distribution Board	1	Matek Systems PDB-XT60	Provides 5V and 11.1V
RC Receiver and Transmitter	1	FrSky Taranis Q X7 and X8R	Enables remote control
Telemetry	1	3DR Radio Telemetry	Provides communication
Laser Range Finder	1	LIDAR-Lite v3	Provides sensitive distance
Global Positioning System (GPS)	1	3DR uBlox GPS with compass	Provides global position feedback
LiPo Battery	1	ZOP Power 3S 4200mAh 40C	Supplies power
Onboard Computer	1	Rpi3	Image processing and high level control
Onboard Camera	1	Rpi3 Camera v2	Records video, bottom
Gimbal	1	3 Axis Gimbal With Controller	Stabilizes the onboard camera
Force Sensitive Resistor	1	Interlink FSR 402	Measures landing success rate

In our design, we used Pixhawk version 1 as the flight controller. It is a high-quality hardware compatible with open-source autopilots such as ArduPilot and PX4. Its hardware includes 32bit, 168 MHz, STM32F427 Cortex-M4F with FPU CPU. In

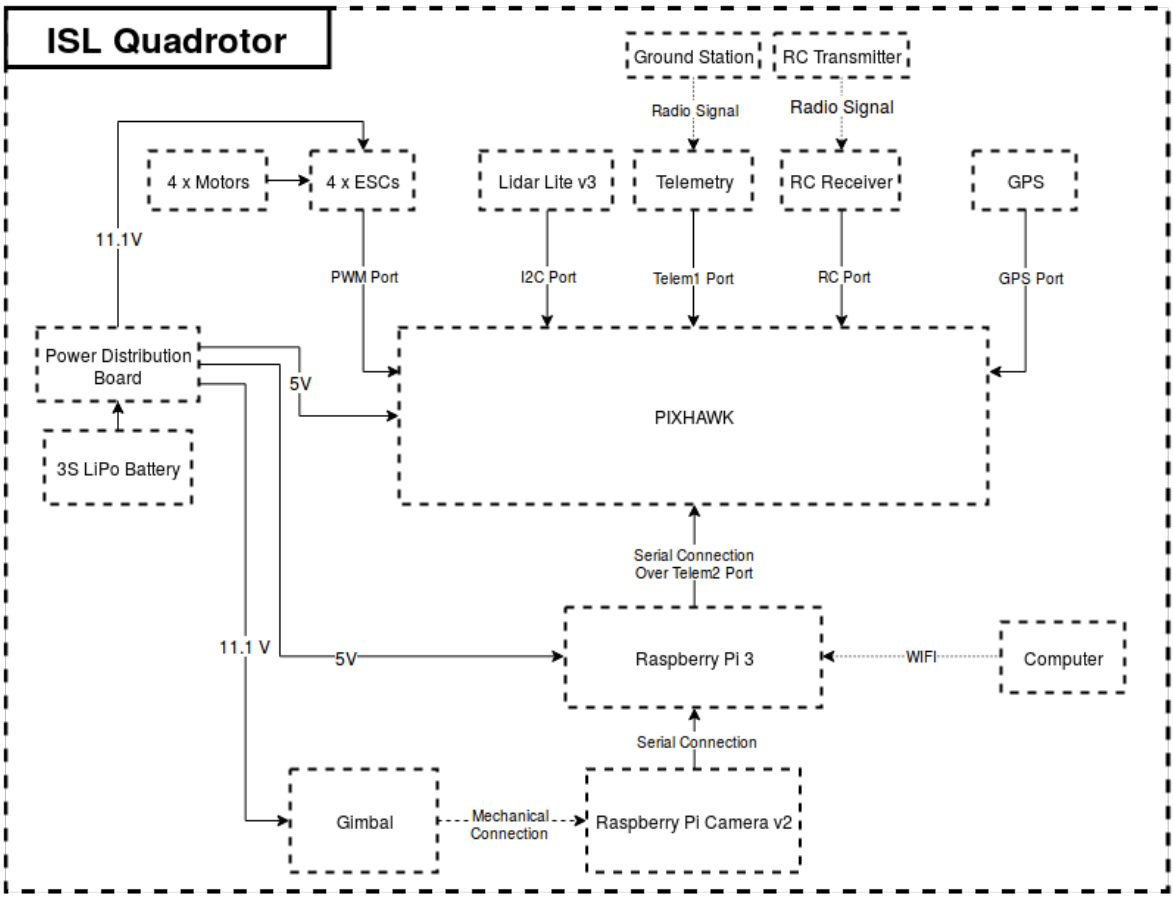


Figure 2.4. ISL Quadrotor hardware architecture

addition, it has an IMU including a gyroscope, an accelerometer, a magnetometer, and a barometer. When it comes to connections, it has 1x I2C, 1x CAN (2x optional), 1x ADC, 4x UART, 1x Console, 8x PWM with manual override, 6x PWM/GPIO input, S.BUS/PPM/Spektrum input, and S.BUS output connections. It can be powered from three different channels: USB, power module, and servo rail [24]. Many hardware peripherals can be attached as well. Since it is widely used by the quadrotor community, it is well-tested and stable. Moreover, there are many supporting communities and forums to handle the encountered problems.

2.3.2. Onboard Processor: Raspberry Pi 3

Raspberry Pi 3 (Rpi3) is a tiny computer with 85.60mm x 56.5mm size and 45g weight. RPi3 has Quad-core 64-bit ARM Cortex A53 clocked at 1.2 GHz CPU and 400MHz VideoCore IV multimedia GPU, 1 GB memory, and built-in Wi-Fi [25]. It

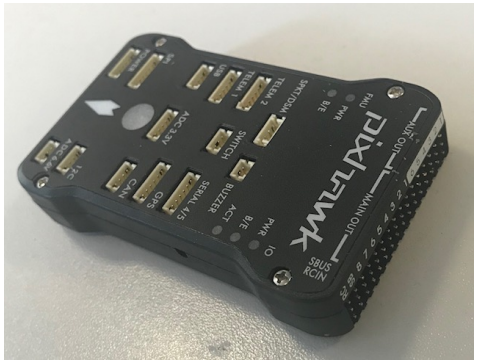


Figure 2.5. PIXHAWK

can be classified as a low power computer. Due to its light weight, small size, active support community, and cheap price, Rpi3 is used as an onboard processor.

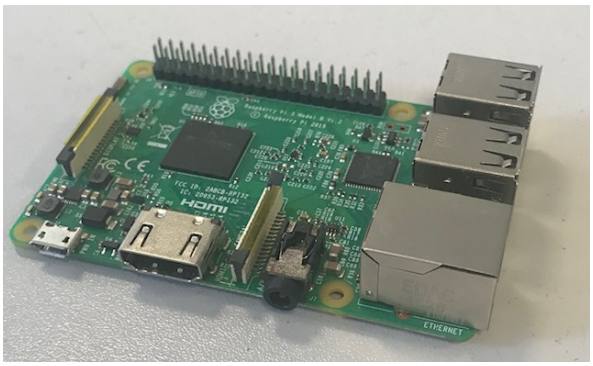


Figure 2.6. Raspberry Pi 3

2.3.3. Global Positioning System (GPS)

GPSs are utilized for localization of the quadrotor in the world frame using satellite signals. However, their resolution has meter sensitivity which is quite low for sensitive tasks requiring more precise measurement. On the other hand, most of the flight modes require GPS data in PX4. These are supported by other sensors such as optical flow sensors and rangefinders. 3DR uBlox GPS with compass is preferred in the quadrotor.

GPS works in the outdoor environment with an open area but does not in indoor. The first GPS signal is received after 20 minutes for the first time. The necessary time to receive GPS signal decreases to approximately 2 min after the first trial. The necessary time can vary from GPS to GPS.

2.3.4. Onboard Camera

Raspberry Pi camera v2 is utilized as an onboard camera due to its small size, lightweight, and low price. Also, it can be serially connected to RPi3 which decreases delay. It has 3280 x 2464 pixels sensor resolution, 62.2 degrees horizontal field of view, and 48.8 degrees vertical FOV. Moreover, it supports 1080p/30fps, 720p/60fps, and 640x480p/90fps video modes [26].

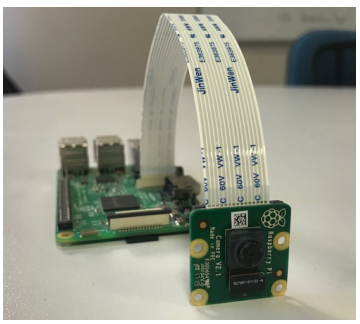


Figure 2.7. Raspberry Pi Camera v2

Let it be noted that we also tried to use GoPro Hero 4 action camera as the onboard camera in the quadrotor. GoPro Hero 4 has lightweight (83 gram), small size (41mm x 59mm x 21mm), wide FoV (79 to 150 degrees), its own battery (up to 2 hours shooting), and different shooting modes such as wide angle and low resolution. However, it was found not to be proper onboard camera choice for RPi3. This is attributed to two reasons. First, as it supports HDMI video output, an HDMI to USB converter needs to be used in order to receive the video stream from the RPi3 onboard computer. This is not practical since the available converters in the market such as Magewell USB Capture HDMI are heavy (around 160 gram) and expensive. Second, a delay occurs during the conversion.

2.3.5. Gimbal

Gimbal is a device that is used for motion and noise cancellation for video recording. Thus, it keeps a camera fixed and stable in the desired axis. It consists of a motor or motors depending on the number of axes that require stability. In our design, we need the yaw orientation of the quadrotor and that of the camera to be the same, but

pitch and roll angles of the camera should be independent of the roll and pitch angle of the quadrotor and always be zero in the ground inertial frame. Therefore, we need two-axis gimbal to fix roll and pitch angles. Since we bought a three-axis gimbal, the motors associated with yaw movement was disabled. Our gimbal needs a 300-gram load to work well. It calibrates itself when powering. After around 15 seconds, it bleeps. The gimbal should not be moved during the calibration. If you cannot see a motion in the motors during the calibration, then something is wrong. Possible reasons are:

(i) The gimbal is moved during the calibration
(ii) The load is not heavy enough or too much heavy
(iii) The motors cannot draw enough current.

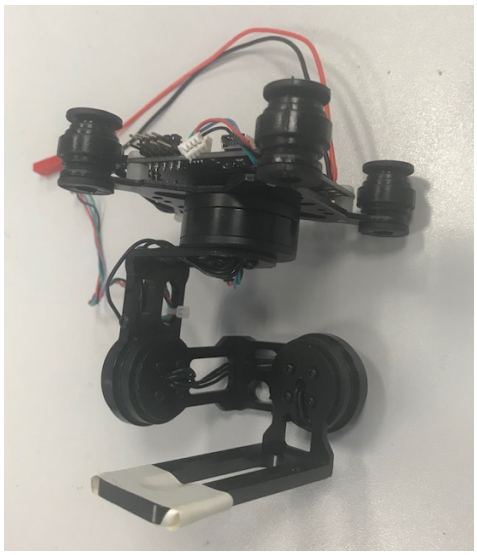


Figure 2.8. Gimbal

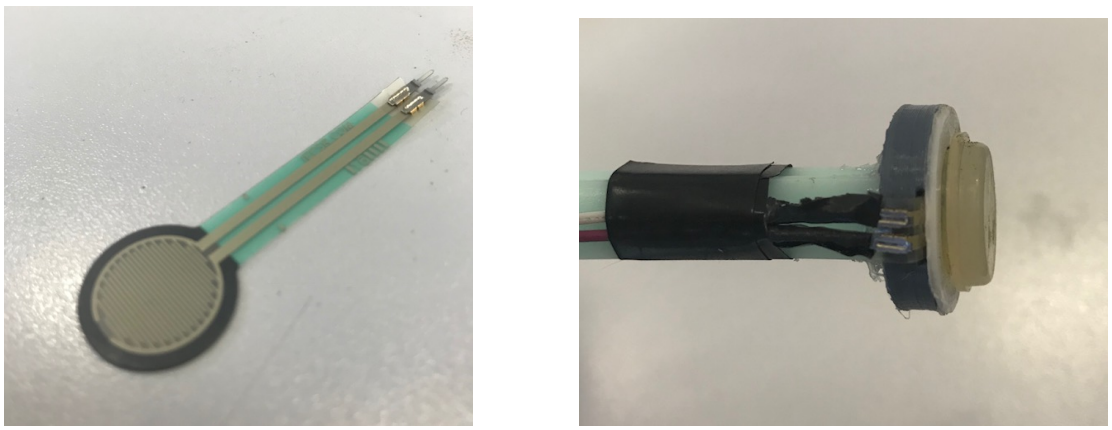
2.3.6. Laser Range Finder

Laser rangefinders provide sensitive altitude feedback for quadrotor applications. In our project, LIDAR-Lite v3 is utilized due to its lightweight, low power consumption, long ranges that is 5 cm to 40 meters, 1 cm resolution, and high accuracy (+/- 25 mm). Moreover, communication over I2C and PWM channels are available, which makes them compatible with PIXHAWK.

Figure 2.9. LIDAR-Lite v3

2.3.7. Force Sensors

The landing gear of the quadrotor is endowed with force sensing. This is achieved using force sensitive resistors. This is a novel approach that can enable the evaluation of landing performance in the quadrotor. When the quadrotor hits the ground with higher velocity, it will create a higher output signal at the force sensors. Apart from that, it can be used in the calibration process. Further details are given in Appendix D.



(a) Force sensor

(b) Force sensor attached to the bottom of landing gear.

Figure 2.10. Force sensing

Force sensors are attached at the bottom of the quadrotor gears as shown in Figure 2.10(b). Interlink FSR 402 sensors are used as force sensors in our design. A 3D printed case whose 3D drawings are shown in D.1 fixes the sensor and the silicon rubber attached on to the effective area of the sensor distributes the applied force to the effective area equally. All of them are connected to an Arduino attached to the quadrotor from analog input ports.

2.3.8. PIXHAWK and Raspberry Pi 3 Serial Connection

The serial connection required for fast data transmission and receiving is made between PIXHAWK's TELEM2 port and RPi3' GPIOs. TELEM2 port's 1-6 pins correspond to +5V, Tx (out), Rx (in), CTS (in), RTS (out), and GND respectively. The officially recommended wiring of the serial connection between PIXHAWK and RPi3 is shown in Figure 2.11.

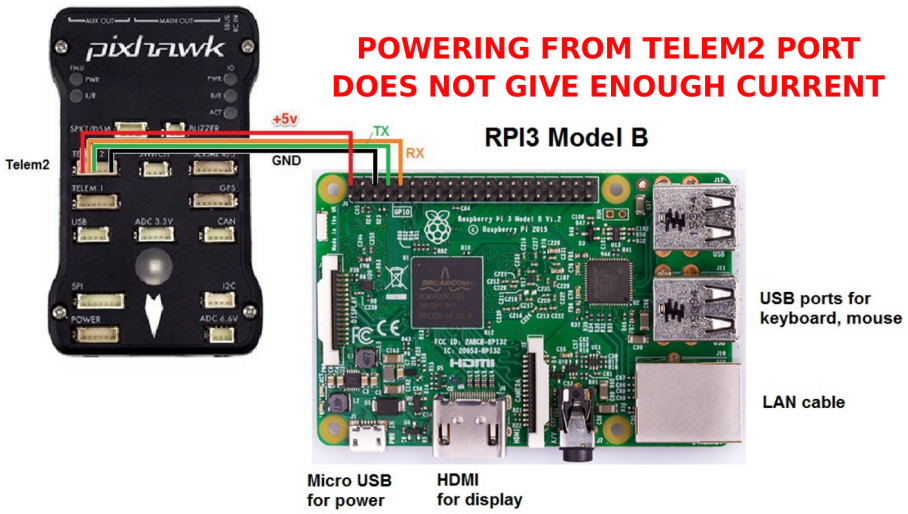


Figure 2.11. PIXHAWK and RPi3 serial connection [27]

However, powering RPi3, which can draw up to 2A from PIXHAWK's telem2 port does not give enough current. Also, powering it from PIXHAWK'S GPIOs is not recommended. Therefore, PIXHAWK and RPi 3 should be powered separately. For that reason, a power distribution board which takes 11.1V as input and gives 5V-2A(max) as output which is the required for powering RPi3.

2.3.9. Quadrotor Software

Various software is used in the different parts of the quadrotor. A list of all the software that needs to be installed is provided in Appendix A. A schematic of the main software components is given in Figure 2.12. The user's guide is provided in Appendix B.

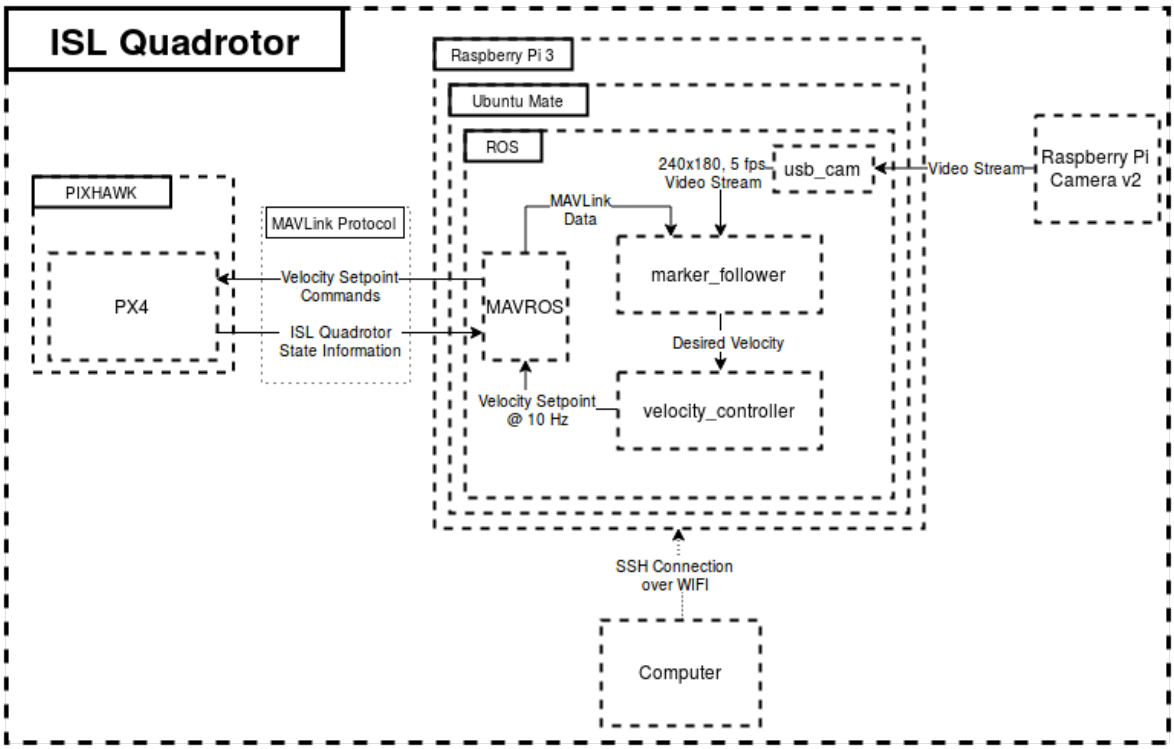


Figure 2.12. ISL Quadrotor software architecture

2.3.9.1. RPi3 Operating System. There are several operating systems supporting RPi3 hardware such as Raspbian and Ubuntu Mate, but we have used Ubuntu Mate in our project because there are no binaries available for you to install ROS directly in Raspbian. Instead, you have to compile all the ROS packages and other dependencies from their sources. Raspbian images with pre-installed ROS and some other packages can be a solution to that problem, but if you need another package, you have to compile it from its source as well.

For the setup, Ubuntu MATE operating system should be written to a micro sd card. Then, RPi3 can be connected to a monitor and used as a computer. In order to remote access to RPi3 from another computer which is required while developing an autonomous quadrotor, Secure Shell (SSH) is utilized. To that, RPi3 and a remote computer should be connected to the same network first. One of the methods is to create a Wi-Fi hotspot on RPi3 and connect it from a remote computer. Then, ssh should be installed and its configuration should be done. You have to enable SSH from raspi-config that is the RPi3 configuration tool. Also, receiving image data over the ssh should be enabled. Otherwise, "QXcbConnection: Could not connect to display ssh"

error is encountered. For receiving image data flow, /etc/ssh/sshd_config parameter "X11Forwarding" should be set to "yes", also connect to ssh by using command "ssh (username)@(ipaddress) -X".

Another important point about RPi3 is that due to its limited memory, required memory for an operation such as building a ROS package exceeds the existing memory. Thus, it gives virtual memory exhausted error. The solution to this problem is to use swap memory. Swap memory is used when the amount of RAM is not sufficient. If a process needs more memory, specified empty places in sd card are utilized as the swap memory which is used as RAM. To create and enable swap memory, the commands in Ubuntu MATE need to be applied:

(i) sudo fallocate -l 2G /swapfile (Specifies the amount of memory for swap memory, 2 GB in this case)
(ii) sudo chmod 600 /swapfile (Changes permissions)
(iii) sudo mkswap /swapfile (Marks the specified memory as swap memory)
(iv) sudo swapon /swapfile (Enables the swap memory)
(v) sudo swapon –show (Verifies that the swap memory is available)

2.3.9.2. PIXHAWK Software. ArduPilot and PX4 are open-source autopilots compatible with PIXHAWK for controlling aerial vehicles. ArduPilot supports several ground vehicles and marine vehicles as well. They enable communication between a quadrotor and a transmitter over radio signals. They have fully manual and semi-autonomous flight modes such as manual, stabilize, altitude hold, position hold, loiter, return-to-haunch, acro, land, and offboard. These modes are assigned to RC Transmitter channels, which enables to change the flight mode manually. In order to test these modes, several simulation environments such as jMAVSim, AirSim, and Gazebo simulation environments can be used. Although both autopilots are very similar, they have minor differences. For instance, PX4 has auto take-off and land modes which requires GPS data. On the other hand, ArduPilot has autoland mode which can land without GPS, but does not has auto takeoff mode. Another difference is related to

the calibration process of the quadrotor. While ArduPilot, for instance, needs ESC calibration, PX4 does not.

One of the most crucial modes for our project is OFFBOARD mode. This flight mode disables RC control except to change modes and enables the control of the quadrotor by position, velocity or attitude setpoints provided by another companion computer over MAVLink protocol which is a very lightweight messaging protocol for communicating with drones. Since high-level image processing to detect ArUco marker, tracking algorithms, and other high-level controllers should run on companion computers, RPi3 in our case, the desired velocity commands should be sent from the computer connected via serial cable or Wi-Fi. Therefore, OFFBOARD mode which enables autonomy is required.

In order to switch to OFFBOARD mode, the quadrotor must be armed and already be receiving a stream of setpoints whose rate should be higher than 2Hz. Otherwise, you cannot switch to this mode. OFFBOARD mode goes back to the previous mode if setpoints are sent below 2 Hz or completely lost. If it also loses RC signals, it switches to return home and land mode. These failsafe modes can be arranged in a desired way as well.

The most suitable ground control stations (GCS) for PIXHAWK that enable human control of the quadrotor are QGroundControl and Mission Planner. Both of them can be described as a GUI for communication between PIXHAWK and a remote computer over MAVLink protocol. QGroundControl is the most suitable GCS for PX4 software, while Mission Planner is more compatible with ArduPilot software. In our project, we have used QGroundControl as GCS and PX4 software for PIXHAWK.

The initial setup of PIXHAWK is also done by using QGroundControl. The necessary steps are listed below:

(i) Power the PIXHAWK from USB
(ii) Select PX4 firmware to upload.

(iii) Select airframe which is similar to x sign.

(iv) Calibrate compass, gyroscope, accelerometer, and level horizon by following calibration steps on GUI.

(v) Calibrate Radio to configure the mapping of RC transmitter's roll, pitch, yaw, throttle to channels. To do that, bind RC transmitter and receiver first.

(vi) Set flight modes to RC transmitter channels.

2.3.9.3. PIXHAWK and Raspberry Pi 3 Communication. The communication between Pixhawk and RPi3 is achieved using MAVLINK protocol. This protocol has been implemented by the MAVROS software. MAVROS is a ROS communication driver for MAVLink communication protocol. After MAVROS package is installed, the communication for PX4 firmware is initiated by px4.launch file. For serial communication, fcu_url parameter in the px4.launch file should be set to "/dev/ttyS0:921600" or "/dev/ttyAMA0:921600". The first term refers to port and the second one stands for baud rate that is the rate at which information is transferred in a communication channel. The serial communication port can be /dev/ttyS0 or /dev/ttyAMA0 depending on Raspberry Pi version and settings. 921600 baud rate is the maximum possible communication rate between PIXHAWK and RPi3. Also, you have to set SYS_COMPANION parameter to 921600 baud rate on QGroundControl for communication on that rate. You can run "roslaunch mavros px4.launch fcu_url:=/dev/ttyS0:921600" command on terminal. For telemetry communication, fcu_url parameter should be set to "fcu_url:=/dev/ttyUSB0:57600" which can vary for different ports and baud rates.

If the serial communication does not work, you have to set "enable_uart" parameter to 1 at /boot/config.txt which is not default in some RPi versions or disable a login shell to access serial port from raspi-config that is the Raspberry Pi configuration tool. Other solutions for specific errors are listed below:

- "FCU: DeviceError:serial:open: Permission denied"
  You have to give ownership to the specified fcu_url ports by using chmod command from the terminal or you have to switch to the root user with "sudo -s"

command.

- "FCU: DeviceError:serial:open: Device or resource busy"

  If you connected telemetry from a USB port (/dev/ttyUSB0), other mounted USBs or launched QGroundControl may lead to this error. Dismount the other USBs or close QGroundControl.

- "serial0: receive: End of file"

  This error might stem from inconsistent baud rate. The specified baud rate at QGroundControl should match with fcu_url:=/dev/ttyAMA0:57600 or 921600 etc. You can test your connection by "mavproxy.py –master=/dev/ttyAMA0 –baudrate 57600 –aircraft MyCopter" command after setting the correct baud rate. If it still does not work and continues to give "serial0: receive: End of file" error, try to restart PIXHAWK a few times. Finally, reinstall the firmware.

- Multiple port usage error

  Do not power RPi3 from USB while using a serial connection. As such, disable the serial connection.

2.3.9.4. Simulation Software. Nowadays, there are many different simulation environments for quadrotors such as Gazebo, V-REP, AirSim, and jMavSim. In our simulations, Gazebo simulation environment is preferred since it is open source and compatible with ROS. Furthermore, it has better documentation, several built-in packages for ROS and a large community.

3. LANDING CONTROL

This chapter focuses on presenting the autonomous landing control of a quadrotor. In the first section, the dynamic model of the quadrotor depending on a well-known model is examined. Then, the position, attitude, and velocity controllers are presented.

3.1. Dynamic Model

The dynamic modelling of quadrotors is well studied in the literature [28] [29] [30]. Assuming that the quadrotor is a rigid body, Euler-Newton equations can be utilized to extract the dynamic model of the quadrotor. A rigid body satisfies principles of conservation of linear and angular momentum. When the dynamic equations of the quadrotor are extracted, different frames should be taken into consideration. The general structure of a quadrotor is shown in Figure 3.1. There are two frames: inertial $(\hat{x}_i, \hat{y}_i, \hat{z}_i)$ and body fixed $(\hat{x}_b, \hat{y}_b, \hat{z}_b)$ frames.

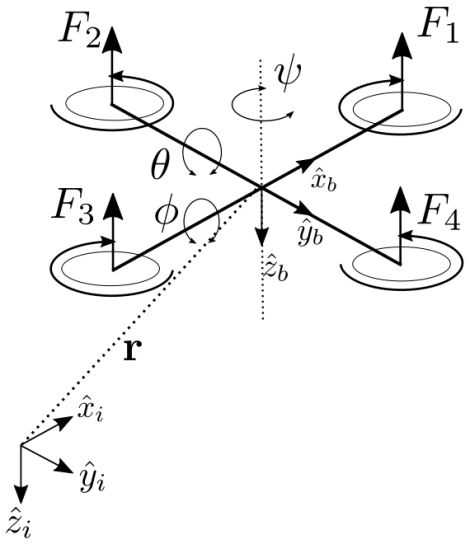


Figure 3.1. Dynamic model of a quadrotor

The quadrotor orientation and position in the inertial frame is expressed by $\Omega^T = [\phi, \theta, \psi]$ and $r^T = [x, y, z]$ respectively. The thrust of the propellers is modeled as $F_i = b \cdot \omega_i^2$, $i = 1, 2, 3, 4$ in body-fixed frame. In the equation, b represents the thrust coefficient and $\omega_i, i = 1, 2, 3, 4$ represents the motor speeds. From the conservation of

linear momentum, the following equation is obtained:

$$m\ddot{r} = mg \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - R(\Omega) b \sum_{i=1}^{4} \omega_i^2 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.1}$$

In the equation, m and g represent the mass (kg) and gravity (kg/m^2) respectively. Also, $R(\Omega)$ stands for rotation matrix from the body-fixed frame to the inertial frame from which is given below :

$$R(\Omega) = \begin{bmatrix} c_\theta c_\Psi & c_\Psi s_\theta s_\phi - s_\Psi c_\phi & c_\Psi s_\theta s_\phi + s_\Psi c_\phi \\ c_\theta s_\Psi & s_\Psi s_\theta s_\phi - c_\Psi c_\phi & s_\Psi s_\theta c_\phi + c_\Psi s_\phi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

The conservation of angular momentum yields the following equation:

$$I\ddot{\Omega} = -(\dot{\Omega} \times I\dot{\Omega}) - M_G + M \tag{3.2}$$

Here, I, M, and M_G represent diagonal inertia matrix $(kg.m^2/s)$, applied torque to the body $(N.m)$, and gyroscopic torques of the rotors $(N.m)$ respectively.

The M, M_G, and I matrices are defined as follows, where d is the drift coefficient, $L > 0$ arm length (m), I_R is rotor inertia matrix $(kg.m^2/s)$, and I_x, I_y, and I_z are the diagonal elements of the inertia matrix I.:

$$M = \begin{bmatrix} Lb(\omega_2^2 - \omega_4^2) \\ Lb(\omega_1^2 - \omega_3^2) \\ d(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2) \end{bmatrix} \tag{3.3}$$

$$M_G = I_R(\dot{\Omega} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}) \cdot (\omega_1 - \omega_2 + \omega_3 - \omega_4) \tag{3.4}$$

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{3.5}$$

For more simple representation of the system equations, invertable artificial system inputs $u_i, i = 1, 2, 3, 4$ are defined based on the motor speeds $\omega_i, i = 1, 2, 3, 4$:

$$\begin{aligned} u_1 &= b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \\ u_2 &= b(\omega_2^2 - \omega_4^2) \\ u_3 &= b(\omega_1^2 - \omega_3^2) \\ u_4 &= d(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2) \end{aligned} \tag{3.6}$$

The system state vector contains the state variables defined in the inertial frame: $X = [\dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}]$

Thus, from the conservation of momentum and angular momentum of inertia (3.1 & 3.2), quadrotor dynamics are obtained as follows:

$$\dot{X} = \begin{bmatrix} -(c_\phi s_\theta c_\psi + s_\phi s_\psi) \cdot u_1/m \\ -(c_\phi s_\theta s_\psi - s_\phi c_\psi) \cdot u_1/m \\ g - (c_\phi c_\theta) \cdot u_1/m \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ (\dot{\theta}\dot{\psi}(I_y - I_z) - I_R\dot{\theta}g(u) + Lu_2)/I_x \\ (\dot{\psi}\dot{\phi}(I_z - I_x) + I_R\dot{\phi}g(u) + Lu_3)/I_y \\ (\dot{\phi}\dot{\theta}(I_x - I_y) + u_4)/I_z \end{bmatrix} \tag{3.7}$$

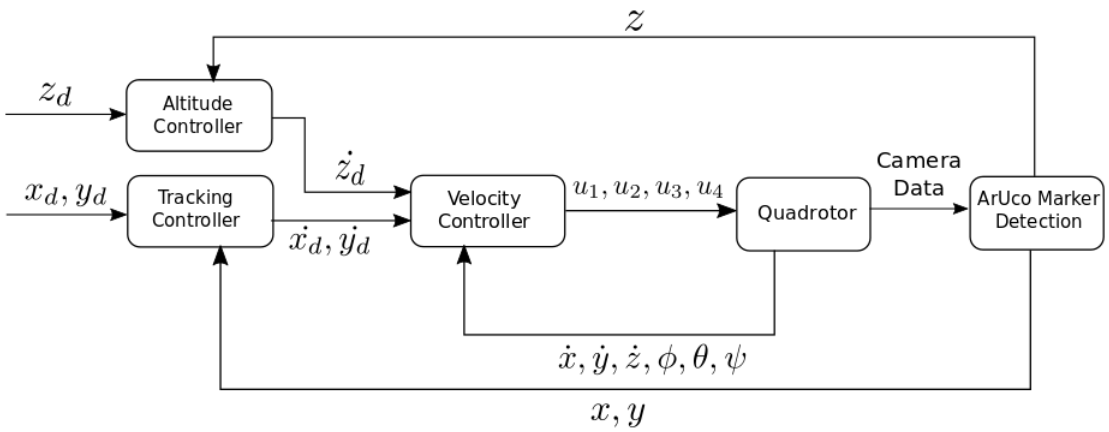


Figure 3.2. The controller of the quadrotor is composed of three coupled parts.

3.2. Control

Consider a quadrotor in flight. The landing problem can be formulated as position control problem - namely to bring the quadrotor to position x_d, y_d, z_d. The values x_d, y_d and z_d depend on the landing platform. The required control structure can be divided into three main categories as shown in Figure 3.2. These are:

- Planar position control: This controller is used to bring the quadrotor to the planar position of the landing platform.
- Altitude control: This controller is used to gradually decrease the height of the quadrotor.
- Velocity control: This controller is used to have the quadrotor stop once it reaches the landing platform.

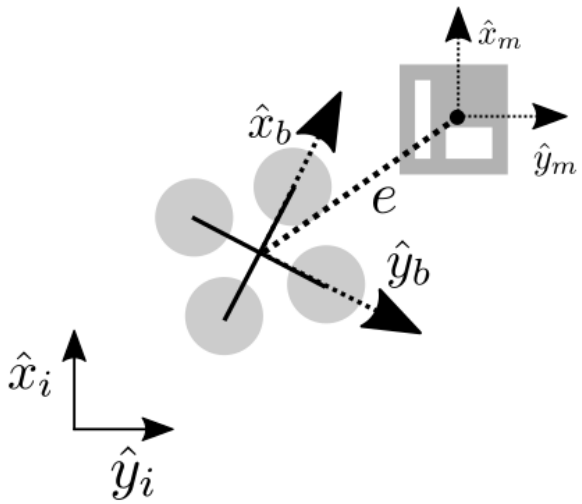


Figure 3.3. Planar position error is defined with respect to to the marker on the landing platform as obtained through visual feedback.

3.2.1. Planar Position and Altitude Control

The position control is based on tracking the relative position of the landing platform with respect to the quadrotor. Let $x_d \in$ and $y_d \in R$ denote the planar position of the landing platform in body-fixed coordinate frame. e_x and e_y denote position errors in body-fixed frame and their formulas are given as follows:

$$\begin{aligned} e_x &= x_d - x \\ e_y &= y_d - y \end{aligned}$$

A PD controller is used in order to generate the desired speeds with respect to x and y axes of the body-fixed frame according to position errors. The desired velocities are

calculated as follows:

$$\begin{aligned} \dot{x}_d &= K_{p5}e_x + K_{d5}\dot{e}_x \\ \dot{y}_d &= K_{p6}e_y + K_{d6}\dot{e}_y \end{aligned}$$

The total error is defined as $e = \sqrt{e_x^2 + e_y^2}$ as shown in Figure 3.3. In the figure $\hat{x}_m$ and $\hat{y}_m$ denotes the ArUco marker frame.

Similarly, the aim of altitude control is to bring the quadrotor to the desired height value. Let $e_z \in R$ defined the altitude error. Then,

$$e_z = z_d - z$$

A PD controller is also used for altitude control.

$$\dot{z}_d = K_{p7}e_z + K_{d7}\dot{e}_z$$

3.2.2. Quadrotor Velocity Control

The velocity control of quadrotors consists of two nested controllers: the inner attitude controller and the outer speed controller as demonstrated in Figure 3.4.

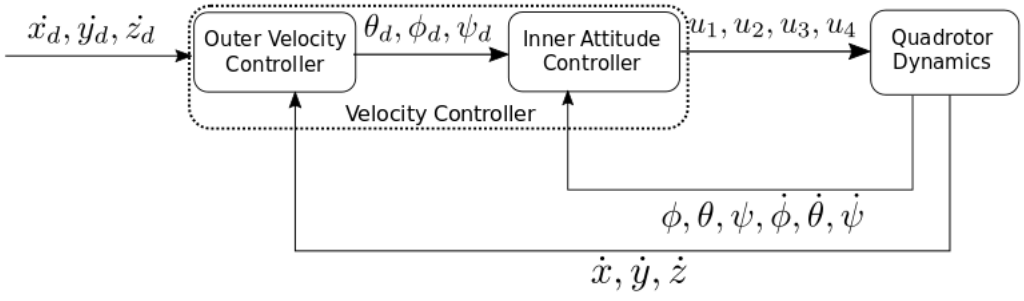


Figure 3.4. Quadrotor velocity control

Let the values of $\dot{x_d}, \dot{y_d}, \dot{z_d} \in \mathbb{R}(m/s)$ denote the desired speeds with respect to the body-fixed frame and $\phi_d, \theta_d, \psi_d(rad/s)$ denote the desired orientation values of quadrotor. Let X be the state values observed from IMU and other onboard sensors. The inner attitude controller brings the system to the desired roll, pitch, and yaw angles values determined by the outer speed controller. Thus, the quadrotor is tried to be brought to the desired speed values specified by reference input. The outer velocity controller computes desired orientation states that need to be attained for the given target velocity values and passes them as inputs to the inner attitude controller. The inner attitude controller operates at a relatively higher frequency than the outer velocity controller. Since the attitude controller takes the output of the velocity controller as input, they are named as inner and outer respectively.

As it can be understood from the quadrotor dynamics, u_2, u_3, u_4 affects the quadrotor's ϕ, θ, ψ direction. Since u_1 is proportional to the sum of the thrust forces, it affects the movements in $\hat{z}_b$ direction. Gyroscopic effects were neglected, while the simple PD controller is used for inner attitude control. The ψ orientation in the target tracking control is considered as constant since quadrotors can move in any direction regardless of their ψ orientation. With the quadrotors dynamics as presented in Section 3.1, the following inner attitude controllers is obtained where $\ddot{z_d}, \ddot{z}, \dot{\phi}_d, \dot{\theta}_d, \dot{\psi}_d, \dot{\phi}, \dot{\theta}, \dot{\psi}$ denote desired and measured acceleration in $\hat{z}_i$ axis, derivative of the desired and measured roll, pitch, and yaw angles respectively.

$$
\begin{aligned}
u_1 &= (g - K_{p1}(\dot{z_d} - \dot{z}) + K_{d1}(\ddot{z_d} - \ddot{z}))/(c\phi\, c\theta) \\
u_2 &= K_{p2}(\phi_d - \phi) + K_{d2}(\dot{\phi}_d - \dot{\phi}) \\
u_3 &= K_{p3}(\theta_d - \theta) + K_{d3}(\dot{\theta}_d - \dot{\theta}) \\
u_4 &= K_{p4}(\psi_d - \psi) + K_{d4}(\dot{\psi}_d - \dot{\psi})
\end{aligned}
$$

A final note regarding the implementation of the controllers:

- Simulations: The developed controllers are tested on the Gazebo simulation environment by utilizing tum_simulator ROS package [31].
- AR.Drone 2: The velocity and altitude controllers are already implemented in its flight controller while the position controller is implemented on a remote computer.
- ISL Quadrotor: The velocity and altitude controllers are implemented in PIXHAWK flight controller while the position controller runs on the RPi3 onboard processor.

4. VISUAL FEEDBACK

The detection and estimation of the position of a quadrotor relative to the landing platform play an important role in autonomous landing. This is achieved through visual feedback as obtained from the onboard camera. The landing platform is detected using the a priori information regarding the marker. The rest of the chapter explains the marker that is used and the detection of the marker while in flight.

4.1. ArUco Marker

The landing platform is marked by an ArUco marker. A sample ArUco marker is shown in Figure 4.1. An ArUco marker is a square marker composed by a wide black border which enables its fast detection in the image and an inner binary matrix which indicates its id. The marker size corresponds to the size of the internal matrix. For instance, a marker size of 5x5 is composed of 25 bits. Moreover, its inner binary structure allows applying error detection and correction techniques. Also, for the detection of different ids, there is a dictionary of markers which is the list of binary codifications of each marker.

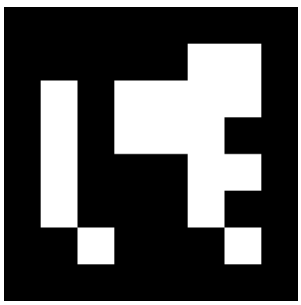


Figure 4.1. ArUco Marker with ID 23

Marker images can be generated by using ArUco library in OpenCV. For detection of the marker, ArUco library is utilized [32]. In the related approach, image segmentation is done first. Then the image contours are found and the unnecessary parts are filtered out and left outmost. In finding the sign code, which is the next step, the inner parts of the found contours are examined and 0 or 1 is assigned to the pixels

according to their color. Finally, in the sign recognition and error correction step, the identification of the sign id is made. In addition, by applying different methods, the position of the marks relative to the camera is estimated.

The aruco_ros package that is available in ROS provides marker generation with given size and id, position estimation, and high-performance tracking. There are single and double marker detection options. For single marker detection node to work, the desired markerId, prior markerSize information, image and camera info topics should be specified in single.launch file. Also, these parameters can be set from the terminal. After running "roslaunch aruco_ros single.launch" command, the detection and position estimation begin. If the marker is detected, the relative position of the marker relative to the quadrotor's camera frame is published to /aruco_single/pose and/aruco_single/position topics. The detected marker image with its coordinate frame is published to /aruco_single/result topic which can be displayed as shown in Figure 4.2.

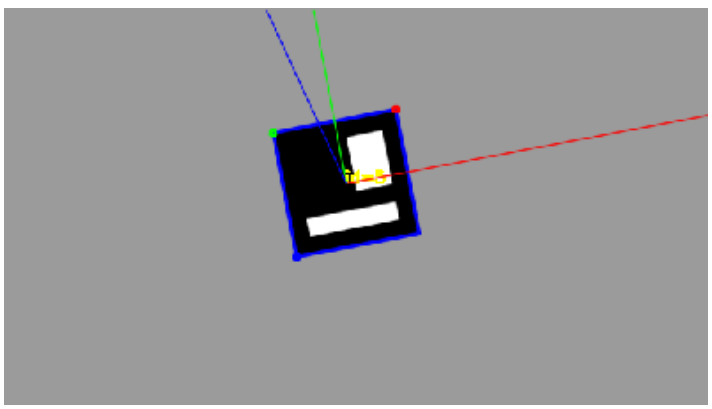
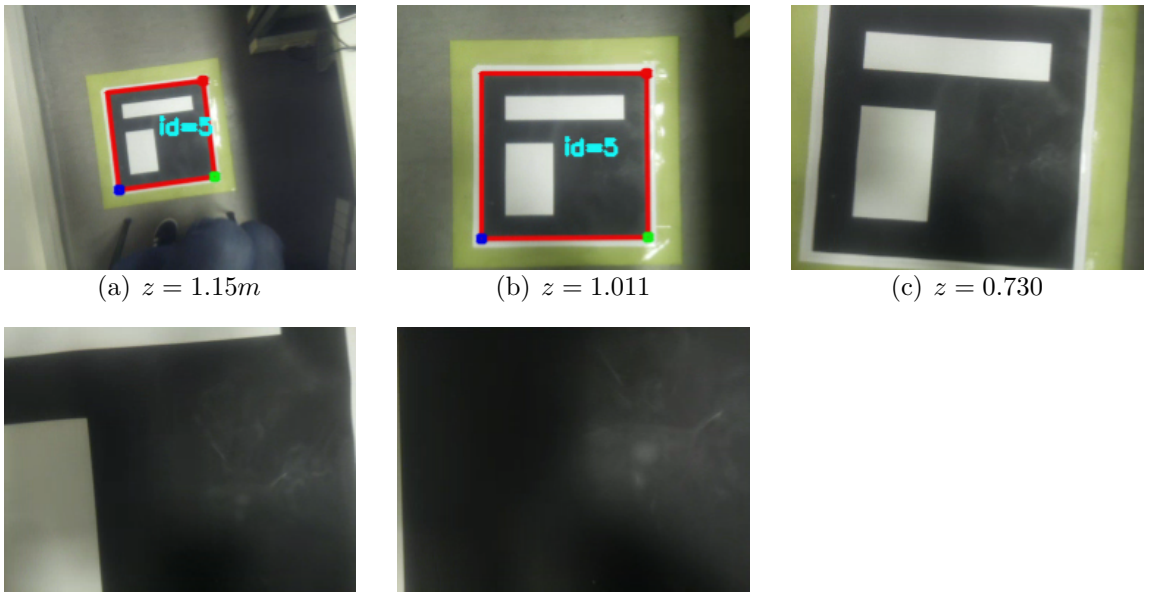


Figure 4.2. Detected ArUco marker

4.2. Marker Detection

Visual processing needs to consider the robot's altitude as ArUco marker detection is possible only between certain altitudes of the quadrotor. A sample sequence is as shown in Figure 4.3. Note that the marker in the figure gets progressively larger. Under a certain altitude, the marker extends the field of view (FoV) of the camera as seen in the figure. In parallel, past an altitude, the marker becomes too small. In both cases, the marker detection algorithm does not work. Therefore, if we can make



(a) $z = 1.15m$ (b) $z = 1.011$ (c) $z = 0.730$

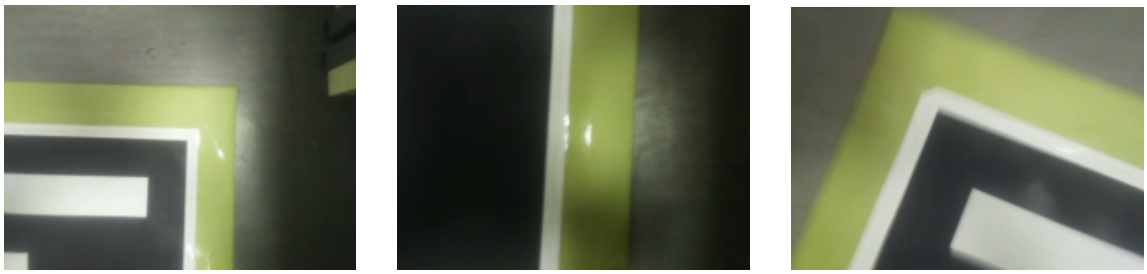
(d) $z = .496$ (e) $z = 0.379$

Figure 4.3. Landing image sequences at various altitudes z while quadrotor's planar position is aligned with that of the ArUco marker.

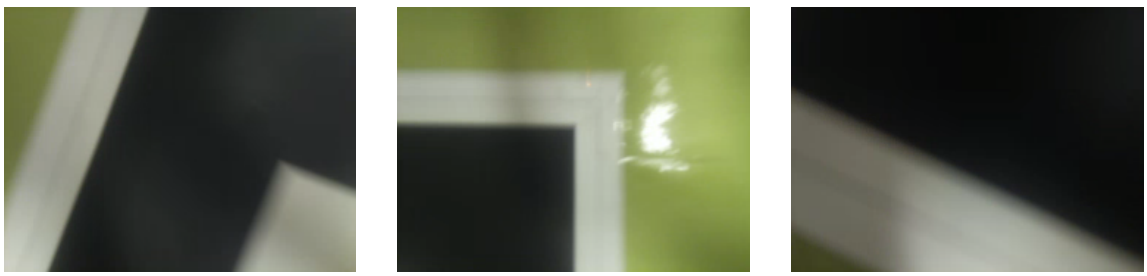
a reasonable estimation of low and high altitudes as shown in Figure 4.5, the detection algorithm works well if the quadrotor's altitude is between these limiting values. Furthermore, the ArUco marker cannot be detected when all the marker is in the FoV even if the quadrotor's altitude is between the limiting values as shown in Figure 4.4. Note that the marker in the figure is only partially viewed in the quadrotor's field of view.

The low altitude can be calculated if marker dimension, camera resolution, and camera FoV are known by a simple geometric approach. Let's assume the resolution of the camera is $N_1 \times N_2$ pixels and horizontal field of view angle is $\Delta°$. As the worst case scenario, it can be considered that the axis of ArUco is positioned to be 45°, as shown in Figure 4.6. Suppose that the edge length of the mark is l, the low altitude h_l is calculated geometrically as follow:

$$h_l = \frac{N_1 l \sqrt{2} \tan(\frac{\Delta}{2})}{N_2 \times 2} \tag{4.1}$$



(a) $z = 0.74m$ (b) $z = 0.71m$ (c) $z = 0.53m$



(d) $z = 0.38m$ (e) $z = 0.31m$ (f) $z = 0.28m$

Figure 4.4. Landing image samples at various altitudes z while the quadrotor's planar position is not aligned with that of the ArUco marker.

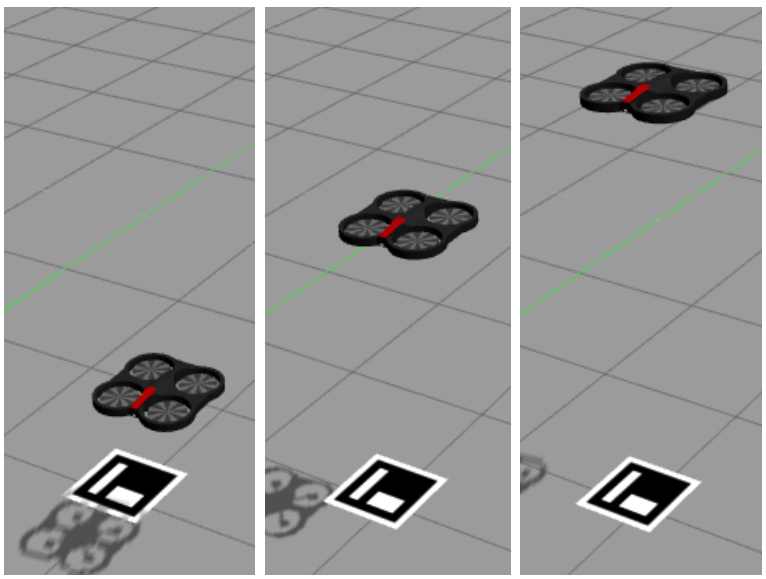


Figure 4.5. Low, average and high altitudes

The high altitude value is determined experimentally. For an 8x8 marker with 0.5-meter sides, maximum detection height is measured as approximately 7.5 meters. So, the far distance h_h is calculated geometrically for an ArUco sign with side length l as such:

$$h_h \approx l \cdot 30 \tag{4.2}$$

The low and high altitude limits can be computed based on the geometry associated with the known ArUco marker dimensions. These limits specify the altitudes at which the quadrotor can detect the landing platform.

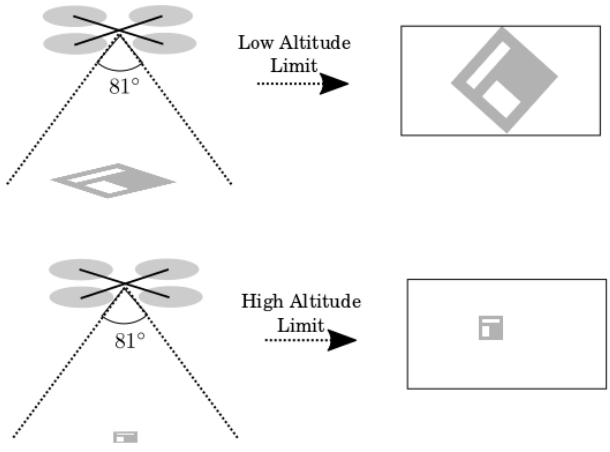


Figure 4.6. Low and high altitude geometry

5. SIMULATION AND EXPERIMENTAL RESULTS

This chapter presents the extensive set of simulations and experiments that have been conducted in order to evaluate landing performance. In the experiments, landing is done autonomously. Experimental setup as illustrated in Figure 5.1 consists of the following stages:

(i) The quadrotor takes off with the automatic start command.

(ii) Next, the low and high altitudes are calculated according to ArUco marker, and the quadrotor ascends to the far altitude accordingly.

(iii) Then, ArUco marker is searched.

(iv) If the marker is detected, the relative planar position is calculated by visual processing and computed e_x and e_y is given to the system as input.

(v) If the position error e is below a threshold τ_p in both x and y direction, the quadrotor begins to descend while trying to keep position error below the threshold.

(vi) Finally, the quadrotor lands onto the platform autonomously.

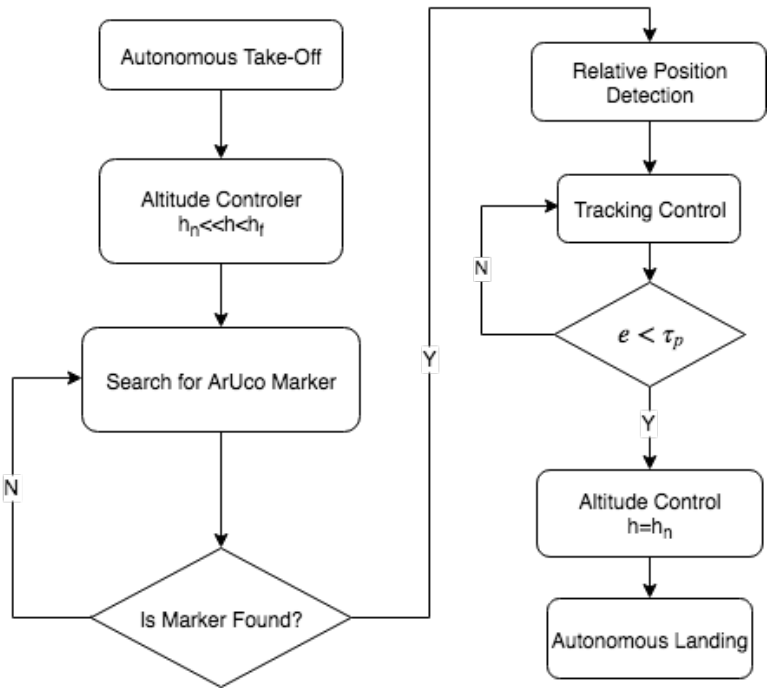


Figure 5.1. Landing experiment algorithm

It should be noted that the quadrotor cannot detect the marker when the quadrotor is below the low altitude or over the high altitude. Moreover, it cannot be detected when the marker is not in the FoV of the quadrotor obviously. In such cases, if there is no external feedback of the marker position such as GPS feedback, x and y velocity of the quadrotor is set to zero in order to keep the planar position constant while descending or hovering.

Another point to be mentioned is that the ISL Quadrotor cannot autonomously take off where GPS data is not available. It is because of the fact that PX4 does not allow autonomous take-off for safety reasons. Therefore, in the ISL Quadrotor experiments, taking off and going to the desired altitude steps shown in Figure 5.1 are done manually.

The landing method is first tested in the simulation environment. This speeds up the debugging process considerably due to four primary reasons. First, quadrotors have power constraint. Since their flight times are rather short, the quadrotor needs to land and its battery should be recharged as necessary. This slows down the testing considerably. The second consideration is due to safety reasons. During testing, the quadrotor can damage the environment and itself which takes time and money to repair the harmed parts. The third one is that the environment and quadrotor itself on the simulation can be easily modified. Lastly, the quadrotor states such as velocity and position can be reached very accurately and localization, one of the most challenging issues of quadrotors in outdoor, can be easily achieved. Once landing performance is acceptable in the simulations, then experiments with real quadrotors are conducted.

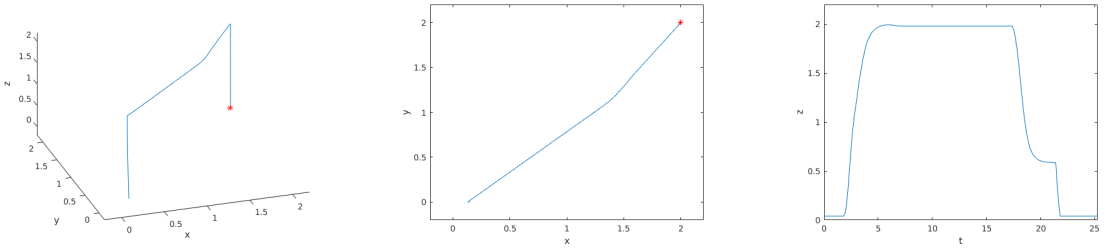
5.1. Simulation

In the simulations, a world containing a quadrotor and an ArUco marker is created for autonomous landing on a fixed marker. The downward facing camera of the quadrotor has a resolution of 640x360 pixels and a horizontal field of view angle 81 degrees. Thus, for an 8x8 Aruco marker with 0.5-meter side, the minimum detection height is theoretically 0.54 meters ($h_l = 1,074l$) while the measured height is about 0.6

Table 5.1. Simulation parameters

Variable	Value	Unit	Variable	Value	Unit
h_l	0.6	m	h_h	2	m
K_{p1}	5	-	K_{d1}	1	-
K_{p2}	10	-	K_{d2}	5	-
K_{p3}	10	-	K_{d3}	5	-
K_{p4}	2	-	K_{d4}	1	-
K_{p5}	0.8	-	K_{d5}	0.5	-
K_{p6}	0.8	-	K_{d6}	0.5	-
K_{p7}	40	-	K_{d7}	30	-
τ_p	0.05	-	l	0.5	m
g	9.8	m/s^2	L	0.2	m
I_{xx}	0.014	$kg.m^2$	I_{xy}	0	$kg.m^2$
I_{xz}	0	$kg.m^2$	I_{yy}	0.014	$kg.m^2$
I_{yz}	0	$kg.m^2$	I_{zz}	0.025	$kg.m^2$

meters. This result is consistent with the developed method. In the simulations, the quadrotor takes off from an initial planar position, ascends to 2m height to search the Aruco marker and then starts descending to 0.6 m. The quadrotor cannot detect the marker below that altitude. In such cases, x and y velocity of the quadrotor is set to zero in order to keep the planar position constant while descending. Finally, quadrotor lands. All steps are done autonomously. The parameter values used in the simulation and control of the quadrotor are presented in Table 5.1. The PD coefficients are chosen by experimental tuning as to minimize oscillations.



(a) Descent path to landing (b) Planar position path (c) Altitude path

Figure 5.2. A sample simulation. The landing position is shown by red asteriks.

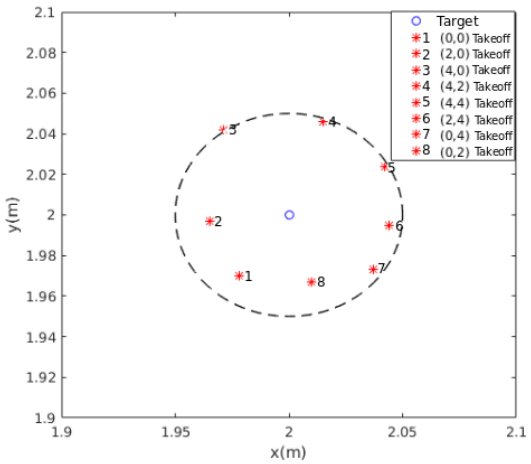


Figure 5.3. Landing performance in simulations for varying initial positions

A sample simulation is shown in Figure 5.2. As can be seen, the landing target is located at (2m, 2m). The path of the quadrotor in its descent to land is shown in Figure 5.2(a). It can be observed that the quadrotor moves toward Aruco marker based on the GPS position of the marker which is assumed to be known. After the marker is detected, more precise marker position estimation becomes possible and, as a result, the direction of the quadrotor slightly changes. This can also be observed more clearly in Figure 5.2(b) which shows how the quadrotor's planar position evolves. The evolution of altitude is shown in Figure 5.2(c).

The simulations are repeated 8 times with different take-off positions of the quadrotor. The landed positions are shown in Figure 5.3. It is observed that all landed positions are located within the 5cm diameter area centered on the target point. Thus, the proposed approach for landing with visual feedback enables successful performance in the Gazebo simulation.

5.2. Experiments with Real Quadrotors

Next, we consider experimentation with real quadrotors. We expect performance to be different as the simulation environment does not include several disturbances such as different light and wind conditions. Furthermore, while the quadrotor is capable of very precise motion, has very low communication delays and can localize itself in

the simulations, these will not be the cases in practical applications. Therefore, the developed algorithms and approaches need to be tested on a real quadrotor system. In fact, the real quadrotor experiments differ from simulations in several aspects:

- In a real flight, the speed needs to be above a certain minimum speed threshold. Otherwise, unlike simulations, the quadrotor does not move in the desired direction. Instead, it moves in random directions.
- There is also a field of view (FoV) problem while moving. Visual feedback does not work well if the quadrotor movements are unexpected, aggressive and fast - since such movements lead to loss of target object. This problem is alleviated through two means. First, the bottom camera is used in conjunction with a gimbal. Furthermore, the speed of quadrotor is lowered to be between 0.03 m/s and 0.05 m/s. This is due to the fact that in order to move in a planar manner, quadrotor as designed needs to make both roll and pitch movements in the desired direction. With a body-fixed camera, the quadrotor possibly loses its track of the marker as it disappears from its FoV. Empirically, over the magnitude of 0.05 m/s speed, AR.Drone makes over 30 degrees rotation in roll or pitch axis which results in the loss of the marker. Thus, it has a very narrow velocity range to achieve the tracking.
- Another major problem is the battery problem.
- Communication delays lead to unstability in position control. This is exacerbated with possible communication losses in the WIFI. The latter is addressed by using an onboard processor. As such, full autonomy can be attained.
- In the simulations, the GPS coordinate of the marker is assumed to be known, while it is not the case for the real quadrotor experiments. Since the GPS position of the marker is not known in the real experiments, the quadrotor cannot find the marker if the marker is not in its FoV. For all cases when the quadrotor cannot detect the marker, it tries to keep its planar position constant by giving zero velocities in x and y directions. Therefore, the quadrotor takes off over the marker.

The experimentation is done on two different real quadrotors: AR. Drone and ISL Quadrotor. As discussed, while it is possible to track the position of the quadrotor with respect to the ground inertial frame in the simulations, this is not the case for real quadrotor experiments. As such, relative planar landing position with respect to marker centroid is used to measure landing performance.

Figure 5.4. AR.Drone hovering over ArUco marker

5.2.1. AR.Drone

AR.Drone experiments are done in the indoors with varying initial altitudes (low, medium, and high). The minimum speed threshold is set to 0.03 m/s for AR.Drone. With increasing error threshold to 0.5, decreasing quadrotor maximum velocity 0.05 m/s, and decreasing marker size from 30cm x 30cm to 20cm x 20cm, the robot is able to hover over the platform as seen in Figure 5.4. The parameter values used in the experiment are given in Table 5.2. The PD coefficients are chosen by experimental tuning as to minimize oscillations.

The quadrotor takes off and ascends to the desired altitude value. Then, visual processing begins to work and the quadrotor tries to track the marker as explained. If the planar error is less than the threshold, the quadrotor begins to descend. Finally, the quadrotor is able to land within a maximum 0.2 m error as shown in Figure 5.5.

Table 5.2. AR.Drone experiment parameters

Variable	Value	Unit	Variable	Value	Unit
MIN_VELOCITY	0.03	m/s	MAX_VELOCITY	0.05	m/s
K_{p5}	0.3	-	K_{d5}	0.05	-
K_{p6}	0.3	-	K_{d6}	0.05	-
τ_p	0.5	-	l	0.2	m

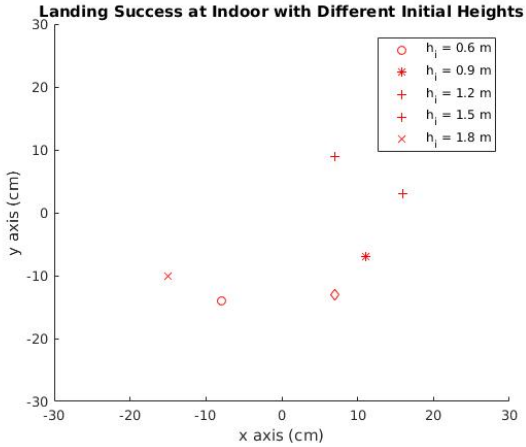


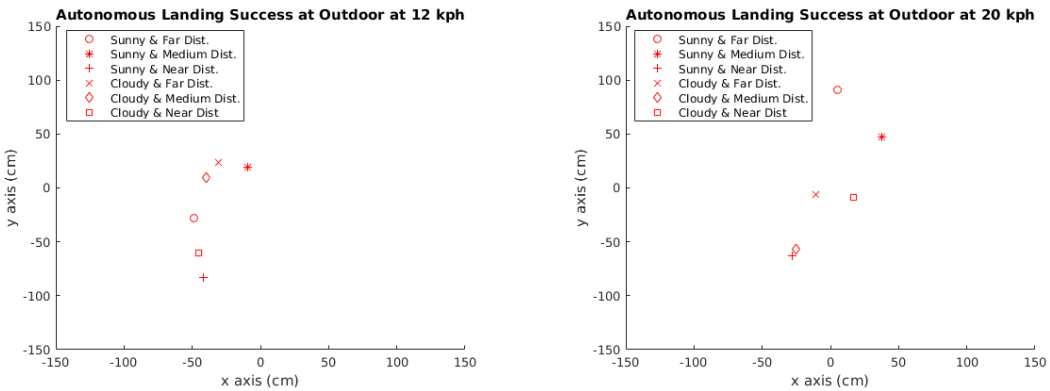
Figure 5.5. Ar.Drone landing performance for varied altitudes

5.2.2. ISL Quadrotor

ISL quadrotor experiments are done in the outdoors under varying illumination (sunny and cloudy) conditions with varying initial altitude (near, medium, and far) and wind (low, medium, and high) conditions. The stationary landing platform is indicated by an ArUco marker of dimension 50cm x 50cm. The gimbal provides highly stable image stream which solves FoV problem during the motion. That enables to move quadrotor relatively higher speeds compared to the AR.Drone without losing the marker. The robot needs 12V-70A power approximately. The battery problem still exists due to its high weight (2.5 kg). The average flight time with a 4200 mAh battery is around 5 minutes. The parameter values used in the experiment are given in Table 5.3. The PD coefficients are determined by experimental tuning as to minimize oscillations.

Table 5.3. The ISL Quadrotor experiment parameters

Variable	Value	Unit	Variable	Value	Unit
MIN_VELOCITY	0	m/s	MAX_VELOCITY	1.5	m/s
K_{p5}	1	-	K_{d5}	0.01	-
K_{p6}	1	-	K_{d6}	0.01	-
τ_p	0.3	-	l	0.5	m



(a) For 12kph wind (b) For 20kph wind

Figure 5.6. ISL Quadrator landing performance for varied illumination, initial altitude and wind speed

At low to medium winds, the quadrotor is able to hover on the marker and make a successful landing on the marker with a maximum 0.8 m error in average as seen in Figure 5.6 from different altitudes 2m, 4m, and 6m. This error results from the quadrotor not being able to localize itself with respect to the ArUco marker below the low altitude - 0.7m in our case. It uses its low sensitive sensors such as IMU and GPS to control its planar velocities while descending. Therefore, the quadrotor can easily be drifted by winds. As it can be seen from the figures, the average landing error increases as the wind speed increases. At very high wind conditions (higher than 30kph), the quadrotor loses the marker. The videos of autonomous landing experiments are available online [33]. Moreover, the quadrotor is able to track a mobile platform with a marker that is moving at different speeds as available online [34].

Interestingly, illumination does not seem to affect landing performance. While one expects visual feedback to be affected by lighting conditions, the detection of ArUco marker seems to be less prone to it. Moreover, it should be noted that the average landing error is greater than compared to the Ar.Drone as expected. This is attributed to two reasons. The first is due to outdoors experimentation as compared to indoors. Thus, there is more drift due to the wind in the environment. Second, while AR.Drone utilizes optical flow to stabilize its planar position, there is no such stabilization in the ISL quadrotor. Therefore, in cases of losing the marker below an altitude, it drifts more as compared to the AR.Drone.

6. CONCLUSIONS AND FUTURE WORK

In this thesis, the autonomous landing of a quadrotor on a fixed platform by visual feedback is presented. This work can be seen as the first steps toward UAV-UGV coordination. After studying quadrotor dynamics, a simple PD controller is developed for attitude, velocity, and position control. The developed controllers are combined with vision in such a way that the quadrotor determines the relative position of itself with respect to an ArUco marker and calculates target velocity and position values. Developed approaches have been tested in the Gazebo simulation environment.

Another contribution of the thesis is that a quadrotor is constructed from scratch. Thus, electrical, mechanical, and control know-how is accumulated, which enables us to design quadrotors with different functionalities. In our design, a gimbal is utilized in order to solve the field of view problem which has improved tracking performance. Moreover, an onboard computer with relatively low computational power, Raspberry Pi 3, is integrated with the flight controller to have a fully autonomous behavior. Running a comparatively complex vision algorithm on a low-power onboard computer that is responsible for generating desired attitudes in order to control its velocity and position can be shown as another major contribution of the thesis.

In experiments, we have shown that autonomous landing on a 50cm x 50cm marker with approximately 80 cm position error is achieved in outdoor with different initial position, light, and wind conditions. Moreover, tracking of a linearly moving marker at low speeds is achieved.

There are several extensions of the current work. The first improvement would be to use more powerful onboard computers such as NVIDIA Jetson TX2 in order to decrease processing time. Hence, image streams with higher FPS, more complex detection and estimation codes, and even online learning algorithms can be processed during flight. As a result, control, tracking, and landing performances are enhanced.

Another improvement would be to design better and faster controller, marker detection, prediction, and search algorithms. More advanced PID applications or non-linear controllers can be used instead of simple PD controllers. Also, a more intelligent marker detection and position estimation algorithm that can predict the marker position when only a part of it in the FoV can enhance the success rate of the landing. Maybe, a nested marker design can be combined with the developed algorithms as well. Moreover, for GPS-denied environments, search algorithms should be integrated with the system to find the marker when the marker is not in the FoV.

When it comes to the force sensors, more sensitive force or load sensors such as strain gauge can be used to obtain more precise measurements. This requires a different leg design. Also, a three-leg design would be better because one leg might hang in the air in four-leg design. After that, the developed force sensing leg design can be used in the calibrations to obtain a better take-off as well.

Finally, the platform should be integrated with the IMU sensor to obtain precise vertical, horizontal and oscillatory motions. Thus, landing on the oscillatory platforms with online motion learning and prediction methods can be studied. However, these methods require very stable motion and sensitive position and velocity feedback. Hence, motion capture systems such as VICON or OptiTrack would be needed.

REFERENCES

1. Mathe, K. and L. Busoniu, "Vision and Control for UAVs: A Survey of General Methods and of Inexpensive Platforms for Infrastructure Inspection", *Sensors*, Vol. 15, No. 7, pp. 14887–14916, 2015, http://www.mdpi.com/1424-8220/15/7/14887.

2. Jung, W., Y. Kim and H. Bang, "Target state estimation for vision-based landing on a moving ground target", *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 657–663, June 2016.

3. Lin, S., M. A. Garratt and A. J. Lambert, "Monocular Vision-based Real-time Target Recognition and Tracking for Autonomously Landing an UAV in a Cluttered Shipboard Environment", *Auton. Robots*, Vol. 41, No. 4, pp. 881–901, April 2017, https://doi.org/10.1007/s10514-016-9564-2.

4. Hu, B., L. Lu and S. Mishra, "Fast, safe and precise landing of a quadrotor on an oscillating platform", *2015 American Control Conference (ACC)*, pp. 3836–3841, July 2015.

5. Yu, C., J. Cai and Q. Chen, "Multi-resolution visual fiducial and assistant navigation system for unmanned aerial vehicle landing", *Aerospace Science and Technology*, Vol. 67, pp. 249 – 256, 2017, http://www.sciencedirect.com/science/article/pii/S1270963816312858.

6. Desaraju, V. R., N. Michael, M. Humenberger, R. Brockers, S. Weiss, J. Nash and L. Matthies, "Vision-based Landing Site Evaluation and Informed Optimal Trajectory Generation Toward Autonomous Rooftop Landing", *Auton. Robots*, Vol. 39, No. 3, pp. 445–463, October 2015, http://dx.doi.org/10.1007/s10514-015-9456-x.

7. Kendoul, F., "Survey of advances in guidance, navigation, and control of unmanned

rotorcraft systems", *Journal of Field Robotics*, Vol. 29, No. 2, pp. 315–378, 2012, http://dx.doi.org/10.1002/rob.20414.

8. Dotenco, S., F. Gallwitz and E. Angelopoulou, *Autonomous Approach and Landing for a Low-Cost Quadrotor Using Monocular Cameras*, pp. 209–222, Springer International Publishing, Cham, 2015, https://doi.org/10.1007/978-3-319-16178-5_14.

9. Yang, S., J. Ying, Y. Lu and Z. Li, "Precise quadrotor autonomous landing with SRUKF vision perception", *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2196–2201, May 2015.

10. Daly, J. M., Y. Ma and S. L. Waslander, "Coordinated landing of a quadrotor on a skid-steered ground vehicle in the presence of time delays", *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4961–4966, September 2011.

11. Mebarki, R., V. Lippiello and B. Siciliano, "Autonomous landing of rotary-wing aerial vehicles by image-based visual servoing in GPS-denied environments", *2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, October 2015.

12. Chaves, S. M., R. W. Wolcott and R. M. Eustice, "NEEC research: Toward GPS-denied landing of unmanned aerial vehicles on ships at sea", *Naval Engineers Journal*, Vol. 127, No. 1, pp. 23–35, 2015.

13. Dijkshoorn, N. and A. Visser, "Integrating Sensor and Motion Models to Localize an Autonomous AR.Drone", *International Journal of Micro Air Vehicles*, Vol. 3, No. 4, pp. 183–200, 2011, http://dx.doi.org/10.1260/1756-8293.3.4.183.

14. Mellinger, D., M. Shomin and V. Kumar, "Control of quadrotors for robust perching and landing", *Proceedings of the International Powered Lift Conference*, pp. 205–225, 2010.

15. Bouabdallah, S., A. Noth and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor", *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Vol. 3, pp. 2451–2456 vol.3, September 2004.

16. Voos, H., "Nonlinear control of a quadrotor Micro-UAV using feedback-linearization", *2009 IEEE International Conference on Mechatronics*, pp. 1–6, April 2009.

17. Bouabdallah, S. and R. Siegwart, "Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor", *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2247–2252, April 2005.

18. Kong, W., D. Zhou, D. Zhang and J. Zhang, "Vision-based autonomous landing system for unmanned aerial vehicle: A survey", *2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*, pp. 1–8, September 2014.

19. Gautam, A., P. B. Sujit and S. Saripalli, "A survey of autonomous landing techniques for UAVs", *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1210–1218, May 2014.

20. Fahimi, F. and K. Thakur, "An alternative closed-loop vision-based control approach for Unmanned Aircraft Systems with application to a quadrotor", *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 353–358, May 2013.

21. Official, P., *Quadcopter AR Drone 2.0 Elite Edition*, 2018, `https://www.parrot.com/global/drones/parrot-ardrone-20-elite-edition`, accessed at July 2018.

22. Krajník, T., V. Vonásek, D. Fišer and J. Faigl, "AR-drone as a platform for robotic research and education", *International conference on research and education in*

robotics, pp. 172–186, Springer, 2011.

23. Bristeau, P.-J., F. Callou, D. Vissiere, N. Petit *et al.*, "The navigation and control technology inside the ar. drone micro uav", *18th IFAC world congress*, Vol. 18, pp. 1477–1484, Milano Italy, 2011.

24. Team, P. D., *Pixhawk 1. PX4 User Guide*, 2018, `https://docs.px4.io/en/flight_controller/pixhawk.html`, accessed at July 2018.

25. Pi, R., *Raspberry Pi 3 Model B - Product Features*, 2018, `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`, accessed at July 2018.

26. Pi, R., *Raspberry Pi Camera Module - Product Features*, 2018, `https://www.raspberrypi.org/documentation/hardware/camera/README.md`, accessed at July 2018.

27. ArduPilot, *Communicating with Raspberry Pi via MAVLink*, 2018, `http://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html`, accessed at July 2018.

28. Mahony, R., V. Kumar and P. Corke, "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor", *IEEE Robotics Automation Magazine*, Vol. 19, No. 3, pp. 20–32, September 2012.

29. Erginer, B. and E. Altug, "Modeling and PD Control of a Quadrotor VTOL Vehicle", *2007 IEEE Intelligent Vehicles Symposium*, pp. 894–899, June 2007.

30. Bouabdallah, S. and R. Siegwart, "Full control of a quadrotor", *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 153–158, October 2007.

31. Bristeau, P.-J., F. Callou, D. Vissière and N. Petit, "The Navigation and Control technology inside the AR.Drone micro UAV", *IFAC Proceedings Volumes*, Vol. 44, No. 1, pp. 1477 – 1484, 2011, http://www.sciencedirect.com/science/article/pii/S1474667016438188, 18th IFAC World Congress.

32. Garrido-Jurado, S., R. Muñoz-Salinas, F. Madrid-Cuevas and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, Vol. 47, No. 6, pp. 2280 – 2292, 2014, http://www.sciencedirect.com/science/article/pii/S0031320314000235.

33. Mete, M., *Autonomous Landing on Fixed ArUco Market at Different Wind Conditions*, 2018, https://youtu.be/rcH38IwGZtg and https://youtu.be/SyO2c1GJwYE, accessed at July 2018.

34. Mete, M., *Autonomous Tracking Moving ArUco Marker*, 2018, https://youtu.be/ql5MKFnyqyU, accessed at July 2018.

35. Ubuntu, *Ubuntu 16.04.5*, 2018, http://releases.ubuntu.com/16.04/, accessed at July 2018.

36. Wiki, R., *ROS Kinetic Installation*, 2018, http://wiki.ros.org/kinetic, accessed at July 2018.

37. PX4, *PX4 Source Code*, 2018, https://github.com/PX4/Firmware, accessed at July 2018.

38. UbuntuMATE, *Download Ubuntu MATE*, 2018, https://ubuntu-mate.org/download/, accessed at July 2018.

39. Dronecode, *Download and Install QGroundControl User Guide*, 2018, https://docs.qgroundcontrol.com/en/getting_started, accessed at July 2018.

40. Ada, L., *Using an Force Sensitive Resistor (FSR)*, 2018, `https://learn.adafruit.com/force-sensitive-resistor-fsr/using-an-fsr`, accessed at July 2018.

APPENDIX A: INSTALLATION SOFTWARE

All the software that needs to be installed is presented in the sequel.

A.1. Gazebo Simulation

- Operating System: Linux Ubuntu 16.04 [35]
- ROS Kinetic [36]
- ROS packages: tum_simulator, aruco_ros, marker_follower, teleop_twist_keyboard
- Image Processing: OpenCV 3 (Comes with ROS Kinetic)

A.2. AR.Drone 2

- Operating System: Linux Ubuntu 16.04
- ROS Kinetic
- ROS Packages: ardrone_autonomy, aruco_ros, marker_follower, teleop_twist_keyboard
- Image Processing: OpenCV 3

A.3. The ISL Quadrotor

- PIXHAWK : PX4 [37] (Can be uploaded directly from QGroundControl)
- Raspberry Pi 3 Operating System: Linux Ubuntu Mate [38]
- ROS Kinetic
- ROS Packages: usb_cam, marker_follower, mavros, aruco_ros
- Image Processing: OpenCV 3
- Ground Station: QGroundControl [39]

APPENDIX B: SOFTWARE USER GUIDE

In the following, how to use the developed software is explained in detail.

B.1. Gazebo Simulation

When Gazebo is launched, two main executables are run actually: gzserver and gzclient. The gzserver can be seen as the core of the simulation. It is responsible for the physics of the simulation. Moreover, it is independent of the graphical user interface (GUI). On the other hand, gzclient is associated with the GUI of the simulation. Also, it enables to control the simulation utilizing GUI.

The simulation is defined by the following items:

- World description: These are defined by files with a .world extension. They include robots, objects, lights, sensors, and other elements describing the simulation environment.
- Models: They are defined by model files which contain only one model which can be a robot, an object, on any other element that can be reused. Thus, it can be used in a world file several times. Models are composed of links which form the object or the robot and joints connecting the links. Several models are can be downloaded from the model database on the internet.

Both model and world files can be written in either Simulation Description Format (SDF) or Unified Robot Description Format (URDF).

Plugins provide a simple structure to control simulation. Plugins can be specified in SDF or URDF files as well as inserted and removed from the command line. Furthermore, plugins provide functionality to models and make the connection between ROS and other simulation components such as models and sensors. After that, model and sensor data can be read and sent by ROS topics and services.

Gazebo uses a number of environment variables to locate files and set up communications between the server and clients. Most important of them which contain set of directories where Gazebo will search for models, other resources such as world and media files, and URI of the online model database are GAZEBO_MODEL_PATH, GAZEBO_RESOURCE_PATH, and GAZEBO_MODEL_DATABASE_URI respectively.

In our simulation, a world containing a quadrotor and an ArUco marker is created for autonomous landing on a fixed marker. Then, I utilized tum_simulator ROS package written by Computer Vision Group at the Technical University of Munich, which handles model descriptions, dynamic integration, and ROS connection of the AR.Drone Gazebo model. In order to launch the simulation, the procedure described below should be followed:

(i) First, run "roslaunch cvg_sim_gazebo only_marker_world.launch" command in terminal. The launch file starts the simulation and makes the connection between ROS and quadrotor model. Thus, state variables can be subscribed and several pre-defined topics and services can be published. The quadrotor motion can be controlled by a keyboard after running "rosrun teleop_twist_keyboard teleop_twist_keyboard.py " command.

(ii) Send take-off command by publishing an empty message to /ardrone/takeoff topic.

(iii) Switch to the bottom camera by calling /ardrone/togglecam rosservice since AR.Drone does not allow the bottom and the front camera to work at the same time.

(iv) Run "marker_follower" and "velocity_controller" nodes. The "marker_follower" node brings the quadrotor to the desired altitude first. Then, it takes AR.Drone bottom camera as input, the marker is detected by using the ArUco marker detection library. After the detection, it calculates the desired velocity commands according to developed controllers. In the meanwhile, the "velocity_controller" node subscribes the desired velocity commands and publishes the desired velocity

command to cmd_vel topic at high speeds. It continuously sends the most recent desired velocity command until it gets updated. Thus, "velocity_controller" node actually serves a thread function. Finally, if the position error between the marker and the quadrotor in x-y direction is less than a threshold, the quadrotor descends to the low distance.

(v) Send land command if you reach the low distance by publishing an empty message to /ardrone/land topic.

B.2. AR.Drone

To launch the software the procedure described below should be followed:

(i) Connect to AR.Drone WIFI network after powering it.

(ii) Run ardrone_autonomy node by "roslaunch ardrone_autonomy ardrone.launch" command to start communication over ROS.

(iii) Take off by publishing an empty message to /ardrone/takeoff topic.

(iv) Switch to the bottom camera by calling /ardrone/togglecam rosservice since AR.Drone does not allow the bottom and the front camera to work at the same time.

(v) Run marker_follower and velocity_controller nodes.

(vi) Land by publishing an empty message to /ardrone/land topic if the necessary conditions are satisfied.

B.3. ISL Quadrotor

After powering, gimbal calibrates itself and RPi3 create a wifi to connect over ssh. By running "roslaunch marker_follower all_system.launch" command, the whole system begins to work. Since high computational power is needed, the display of the Ubuntu Mate is disabled, which means that RPi3 boots to the terminal when powered. Due to a low computational power of RPi3, image resolution and frame per second (fps) are lowered to 240x180 pixels and 5 fps as well.

To speed up onboard computer, marker_follower package which takes the video stream as input makes image processing to detect the marker, calculates the desired velocity commands according to developed controller methods is built in release mode instead of default debug mode, which increases the speed. Furthermore, the highest possible 921600 baud rate is used for communication between the PIXHAWK and RPi3. These steps on their own can not speed up RPi3 to give a continues desired velocity commands faster than 2 Hz although PX4 COM_OF_LOSS_T parameter, the time-out to wait when OFFBOARD connection is lost before triggering OFFBOARD lost the action, is set to 10 seconds which is so large for a quadrotor. Therefore, PIXHAWK cannot switch to OFFBOARD flight mode. To solve this problem, thread logic is utilized. A separate "velocity_control" node publishes velocity setpoint command commands at 10 Hz. These commands are updated as the desired velocity commands are updated by marker_follower node.

To launch the software the procedure described below should be followed:

(i) Connect to "islquad" WIFI network after powering ISL Quadrotor
(ii) Connect to Raspberry Pi 3 over ssh by "ssh islquad@10.42.0.1 -X" command.
(iii) Run usb_cam node by "roslaunch usb_cam usb_cam-test.launch" command in order to receive camera stream and publish it to a rostopic by changing the resolution to 240x180 pixel and frame per second to 5.
(iv) Start serial commmunication between PIXHAWK and RPi 3 utilizing MAVROS by "roslaunch mavros px4.launch fcu_url:=/dev/ttyS0:921600" command.
(v) Run marker_follower and velocity_controller nodes by commands "rosrun marker_follower marker_follower" and "rosrun marker_follower velocity_controller". After that, velocity setpoints are sent with 10 Hz.
(vi) The system switches to OFFBOARD mode meaning autonomous mode when the flight mode is manually changed from the RC transmitter for security reasons.

APPENDIX C: LANDING PLATFORM

For the landing platform, we have proposed two different platforms. The first is placed on top of a mobile robot while the second platform is designed and built for testing and development purposes.

C.1. The Landing Platform on Jaguar Robot

The ultimate goal is to land the quadrotor on a robot. For this, one of the Jaguar robots is considered A landing platform is designed and placed on top of the robot as shown in Figure C.1. This platform can be used in the future UAV-UGV applications.

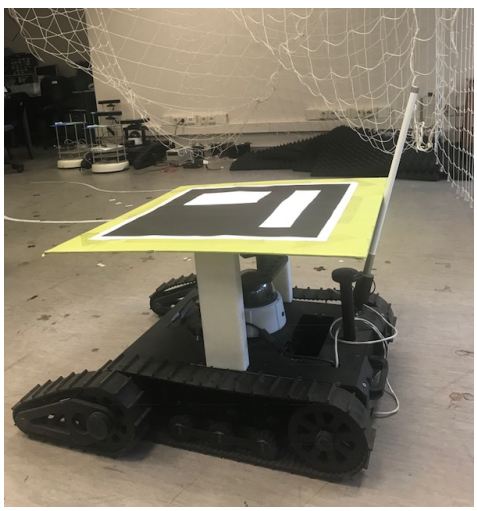


Figure C.1. Landing platform on Jaguar robot

C.2. Oscillatory Landing Platform

The second one is a mobile platform that can move vertically as shown in Figure C.2. Thus, it will enable experimentation with landing on an oscillating pad. In addition, the orientation of the landing pad can be changed. The design includes a 65 mm x 65 mm x 5 mm plexiglass landing pad, 3 linear servos, 3 motor drivers, 3 fastener mechanisms and a Waveshare 10 DoF IMU to get platform orientation feedback as listed in Table C.1. The fastener mechanism that connects the linear servos to the landing pad is as shown in Figure C.3. The fastener mechanism consists of a damper

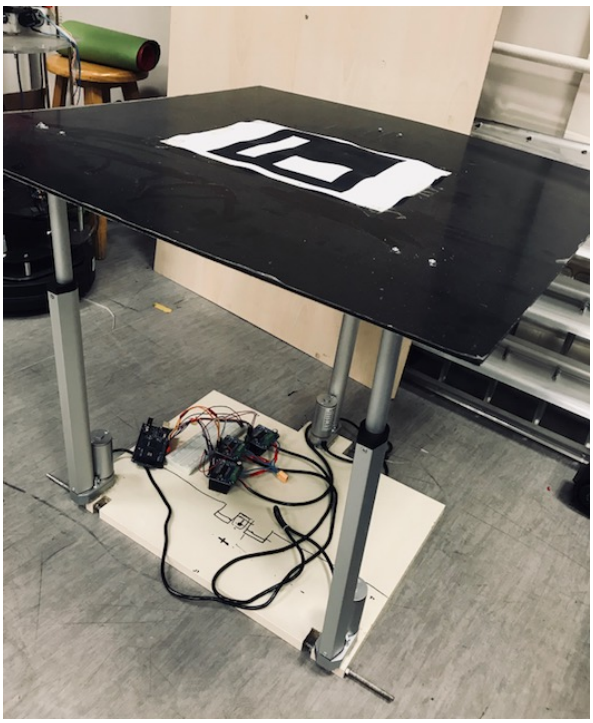
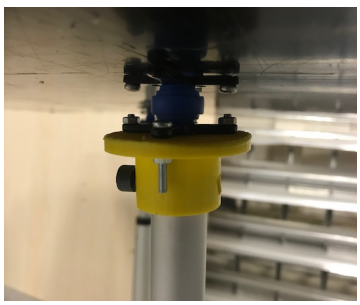
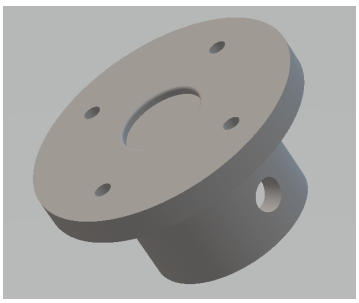


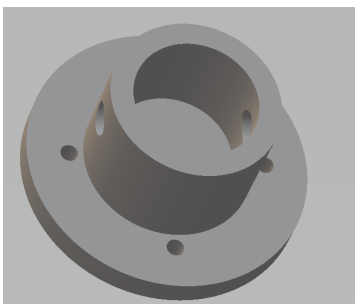
Figure C.2. Stationary landing platform with vertical movement capability



(a) Fastener mechanism consisting of a damper and a fastener



(b) Top view of the fastener



(c) Bottom view of the fastener

Figure C.3. The fastener mechanism of the stationary landing platform.

and a 3D-printed fastener as shown in Figure C.3(a). It affixes a linear motor to the landing pad. The dampers allow 3 DoF motion to the joints which can be seen as a soft robotic solution to the conventional 3 DoF joints. The design details of the fastener are as shown in Figure C.4.

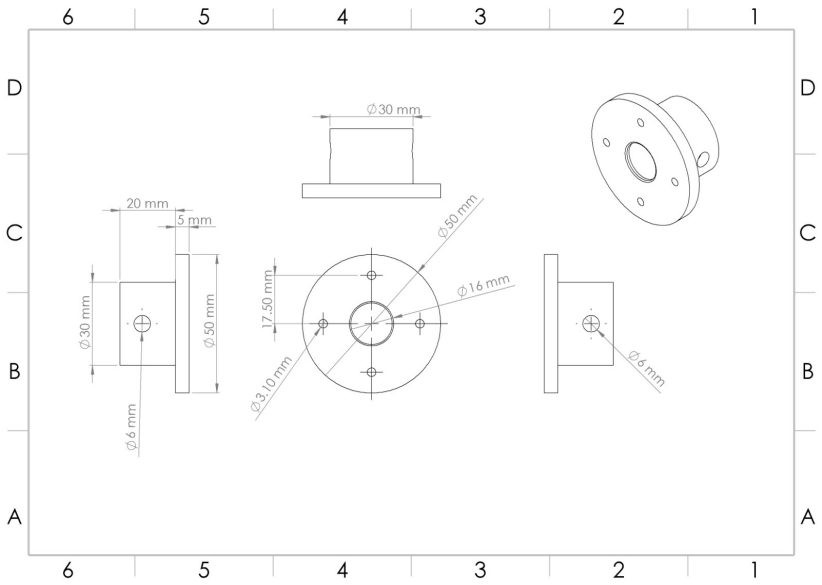


Figure C.4. Fastener design schematics

Table C.1. Main components of oscillatory landing platform

Component	Quantity	Model	Function
Plexiglass	1	65 mm × 65 mm × 5 mm	Landing pad
Linear servo	3	CNMAWAY 300mm Stroke 12V 45mm/s	Platform motion
Motor driver	3	World Chips BTS7960B	Linear servo motion
Damper	3	-	3 DoF Motion and noise cancellation
Fastener	3	3D Printed	3 DoF Connection Between Liner Servo and Plexiglass
IMU	1	Waveshare 10 DoF	Orientation feedback

APPENDIX D: FORCE SENSING

Force sensitive resistors are used as force sensors. A force sensor is attached to the bottom of each landing gear. A special design is done for the attachment. First, a casing that allows the sensor to be placed in it is designed and 3D-printed. The details of the design are given in Figure D.1. Once the sensor is placed inside the casing, a silicon rubber is placed on top of it in order to enable uniform distribution of force on the sensor. Once the sensor is connected to a processor, the signal from it can be read. The connection to an Arduino is as shown in Figure D.2

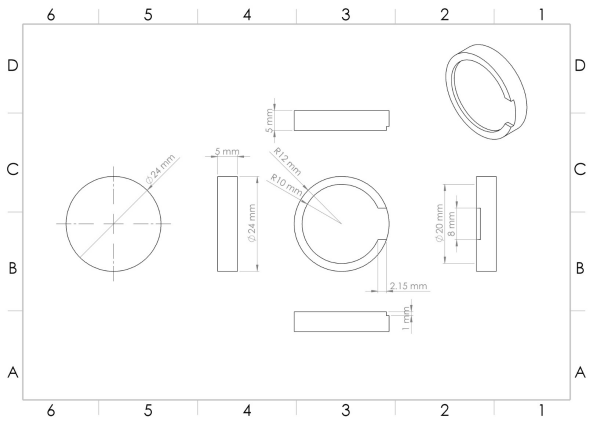


Figure D.1. Force sensor casing design schematics

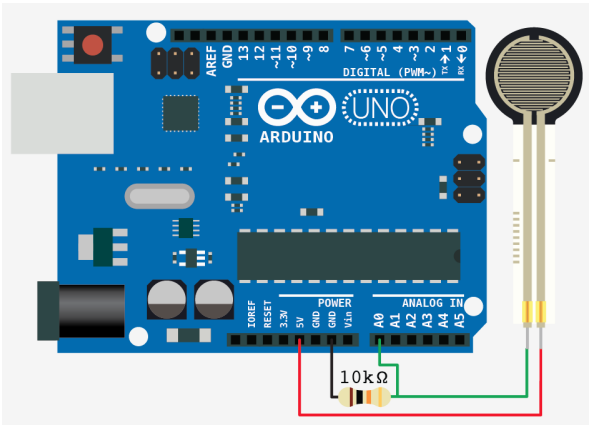
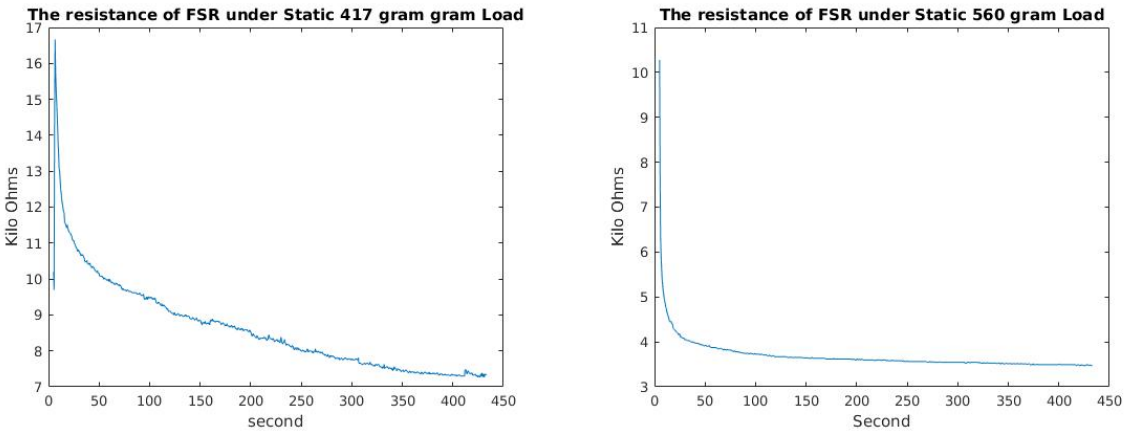


Figure D.2. FSR-Arduino connection [40]

Their resistance changes depending on the applied force. The resistance is inversely proportional to the applied force. If there is no force applied to the FSR, it behaves like an open circuit. As the applied force on the sensor increases, the resistance

decreases. It has very low +/- %15 accuracy and +/10 hysteresis effect. Moreover, the resistance of the FRS changes with the static forces. In other words, the resistance of the sensor under a specific force applied changes with time as shown in Figure D.3. Due to this complex behaviors, it has not been modeled yet.



(a) 417 grams Load (b) 560 grams Load

Figure D.3. Time variation of resistance of FSR under fixed load