UMA METODOLOGIA DE ESTIMAÇÃO DE ALTURA A PARTIR DE VISÃO MONOCULAR PARA VOOS ACOMPANHANDO O RELEVO

IGOR SALES DA GAMA CAMPOS

UMA METODOLOGIA DE ESTIMAÇÃO DE ALTURA A PARTIR DE VISÃO MONOCULAR PARA VOOS ACOMPANHANDO O RELEVO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Luiz Chaimowicz
Coorientador: Erickson Rangel do Nascimento

Belo Horizonte
Julho de 2016

IGOR SALES DA GAMA CAMPOS

A MONOCULAR VISION BASED HEIGHT ESTIMATION METHODOLOGY FOR TERRAIN FOLLOWING FLIGHTS

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Luiz Chaimowicz
Co-Advisor: Erickson Rangel do Nascimento

Belo Horizonte
July 2016

Campos, Igor Sales da Gama

C198m A Monocular Vision based Height Estimation Methodology for Terrain Following Flights / Igor Sales da Gama Campos. — Belo Horizonte, 2016
xxvii, 103 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais — Departamento de Ciência da Computação
Orientador: Luiz Chaimowicz
Coorientador: Erickson Rangel do Nascimento

1. Computação — Teses. 2. Robótica — Teses. 3. Visão por computador I. Orientador. II. Título.

CDU 519.6*82.9(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Uma metodologia de estimação de altura a partir de visão monocular para voos acompanhando o relevo

IGOR SALES DA GAMA CAMPOS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. ERICKSON RANGEL DO NASCIMENTO - COORIENTADOR
Departamento de Ciência da Computação - UFMG

PROF DOUGLAS GUIMARÃES MACHARET
Departamento de Ciência da Computação - UFMG

DR. GUSTAVO MEDEIROS FREITAS
Instituto Tecnológico Vale Mineração

PROF. VILAR FIUZA DA CÂMARA NETO
Departamento de Educação - FUCAPI

Belo Horizonte, 08 de julho de 2016.

Dedico este trabalho à minha família, especialmente à quem escolheu ser mais uma mãe em minha vida, Antônia Noronha Sobrinha.

Agradecimentos

A Deus, por sempre iluminar o meu caminho.

À minha família, por todo o apoio e suporte em toda a minha vida.

À Antônia Noronha Sobrinha, pelo carinho, cuidado e amor dedicados à mim.

Ao meu irmão, David Caminha, pelas longas conversas e conselhos.

Aos professores Eduardo Freire Nakamura, Walter Lucas e William Malvezzi, pelas inúmeras conversas, conselhos e aulas excepcionais que me inspiraram a seguir seus passos.

Aos professores Luiz Chaimowicz e Erickson Rangel do Nascimento por toda a orientação dada ao longo da jornada do mestrado.

A todos os membros do laboratório de Visão e Robótica da UFMG, o VeRLAB, do qual tive a honra de fazer parte durante os últimos três anos, em especial ao técnico, Gabriel, pelo auxílio em diversos momentos na construção do protótipo usado nessa pesquisa.

Aos amigos que tive a oportunidade de fazer em Minas Gerais, em especial (ordem alfabética): Clayson Celes, Daniel Mox, David Saldaña, Dennis Larkin, Elerson Rubens, Héctor Azpúrua, Levi Osterno, Omar Vidal, Paulo Drews, Ramon Melo e Vladimir Portela.

Ao Phelipe Vasconcelos pelo auxílio na realização de alguns experimentos.

Às minhas companheiras caninas, Alfa e Chocolate, pela lealdade e fidelidade.

Às agências de fomento: CNPq e FAPEAM, pelas bolsas de pesquisa concedidas.

Ao ITV (Instituto Tecnológico Vale), pela bolsa de pesquisa e também pela oportunidade de realizar experimentos em minas da Vale.

À banca de defesa desta dissertação pelo tempo e atenção dedicados à avaliação deste trabalho.

E a todos que de alguma forma contribuiram para a concretização deste trabalho.

"Anybody who has been seriously engaged in scientific work of any kind realizes that over the entrance to the gates of the temple of science are written the words: 'Ye must have faith'."

(Max Planck)

Resumo

Nesta dissertação apresentamos um arcabouço completo para voos acompanhando o relevo, composto de um VANT (Veículo Aéreo Não-Tripulado) experimental projetado especificamente para a nossa pesquisa para voos acompanhando o relevo, um algoritmo de estimativa de altura, um classificador de confiabilidade e um controlador de altura proporcional. Percebemos o impressionante crescimento da utilização de VANTs, especialmente em aplicações de mapeamento, que exigiriam a criação de novas tecnologias para permitir que esses sistemas percebam o ambiente ao seu redor. Especificamente nós escolhemos enfrentar o problema de voos acompanhando o relevo, uma vez que é relevante para missões autônomas, mas ainda sem solução para os sistemas disponíveis para consumidores. Praticamente todas as aeronaves de mapeamento carregam uma câmera, por isso optamos por explorar isto, a fim de usar o *hardware* atualmente disponível para extrair a informação da altura para realizar voos acompanhando o relevo. Nossa metodologia consiste na utilização de fluxo óptico para rastrear características em vídeos obtidos pelo VANT, bem como sua informação de movimento para estimar a altura de vôo. No entanto, uma vez que o nosso veículo é um multirotor, é suscetível a ventos e outros distúrbios que possam induzir guinada, o que nós não abordamos neste estudo. Portanto, para determinar se a nossa estimativa de altura é confiável, nós treinamos uma árvore de decisão que recebe informações do fluxo óptico como entrada e classifica se a estimativa é confiável ou não. Além disso, também propomos um controlador de altura baseado em *waypoints* para guiar a aeronave até a altura desejada. Nosso classificador atingiu acurácia de 80% para positivos e 90% para negativos, enquanto o nosso algoritmo de estimativa de altura é mais acurado que outras alternativas, porém menos preciso, o que é consequência de uma menor tolerância a ruído. Nosso controlador foi testado em simulações e mesmo quando sujeito a 10% de ruído funcionou com sucesso.

Palavras-chave: VANT, Voos Acompanhando o Relevo, Fluxo Óptico, Visão Computacional, Robótica.

Abstract

In this thesis we present a complete terrain following framework composed of an experimental UAV (Unmanned Aerial Vehicle) platform specifically designed for our research towards terrain following flights, a height estimation algorithm, a reliability classifier and a proportional height controller. We realized the astounding growth of UAV usage, notably in mapping applications, would soon require the creation of new technologies to enable these systems to perceive their surroundings. Specifically we chose to tackle the terrain following problem, as it is relevant for autonomous missions, yet still unresolved for consumer available systems. Virtually every mapping aircraft carries a camera, therefore we chose to exploit this in order to use presently available hardware to extract the height information toward performing terrain following flights. Our methodology consists of using optical flow to track features from videos obtained by UAV as well as its motion information to estimate the flying height. However, since our vehicle is a multirotor, it is susceptible to winds and other disturbances that might induce yaw, which we did not address in this study. Therefore, to determine if our height estimation is reliable, we trained a decision tree that takes the optical flow information as input and classifies whether the output is trustworthy or not. In addition, we also propose a waypoint based height controller to guide the aircraft to the desired height. Our classifier achieved accuracies of 80% for positives and 90% for negatives, while our height estimation algorithm was more accurate than other alternatives, but less precise, which is consequence of lower noise tolerance. Our controller was tested in simulations and even when subject to 10% noise it worked successfully.

Keywords: UAV, Terrain Following Flights, Optical Flow, Computer Vision, Robotics.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Mankind has always relied on maps for navigation and orientation. Varying from civilian to military purposes, they have long been ubiquitous and are essential for commerce and development of strategies. Nowadays, despite its importance, cartography is usually overlooked, probably because of how easy it is to access digital maps and navigate through satellites with an assistant giving us every direction. However, it was not always like that.

Today we have planes and satellites providing photographs that are extremely useful to make accurate and precise maps, but in the past there were not even computers available to aid in such paramount task. Early cartographers used mathematics and archaic measurement tools to evaluate distances and draw maps, which meant the resulting output could only be as accurate as the tools they used, leading to gross errors in early maps.

As aerial vehicles technology continued to evolve, nowadays instead of using manned aircraft we have drones capable of performing the same tasks as their manned counterparts, and to do so remotely, without risking any lives, also at a lower cost, without the need of complex space-like suits, pressurized cabins or g-force limitations.

As usual, military development eventually improves the civilian technology and this brought us Unmanned Aerial Vehicles (UAVs) capable of performing the most various tasks, from spraying pesticides to monitoring vegetation growth and health in a farm or even counting cattle, mapping mines and mountains, helping in search and rescue missions, among many other things.

The use of UAVs is becoming increasingly popular at an astounding rate. Especially small vehicles that weigh less than 25Kg, since their regulations are more flexible [Canis, 2015]. Probably, one of their main uses nowadays is in mapping applications, being at a farm or a mine, maybe even a construction site or environmental catas-

trophes monitoring. Since these use Commercial Of The Shelf (COTS) cameras that are not meant for extreme long range photography and in these applications a higher precision is usually required, such vehicles commonly fly at low Above Ground Level (AGL) altitudes.

While flying at low altitudes over plain terrain is a simple task that can be easily achieved through the GPS and Inertial Measurement Unit (IMU) available in these vehicles, flying over rough terrain is considerably more challenging as it requires use of specific and expensive sensors such as RADAR rangefinders or previous knowledge of the terrain elevation of the desired flight area.

Some applications specifically benefit from or even require low altitude flights, such as mapping, farming, flying under the RADAR and compliance with aviation regulations. A few ways of performing such flights is through the use of specific sensors, such as RADAR, SONAR, and laser rangefinders, however these are either expensive or provide limited ranges. To avoid using these sensors an UAV can either use previously known terrain information or fly above the highest point over the desired flight path.

There are public databases such as the one from the Shuttle Radar Topography Mission (SRTM), a mission performed by NASA where the space shuttle was fitted with a RADAR sensor while orbiting the planet in an effort to capture the terrain elevation of most of the earth's surface, however they tend to present low resolution, thus being unreliable for places with considerably rough terrain, such as mines or mountains.

To avoid complications, typically, when flying an UAV over a mountainous region, the waypoints are determined such that it flies at a constant Above Sea Level (ASL) altitude determined from the highest point on the flight area. However, this could generate suboptimal results depending on the desired mapping application, such as the case of aeromagnetic surveys, where the results are better the closer the vehicle is to the terrain and at a constant AGL altitude. To make things even worse, the resulting flight plans might be non-compliant with current *Agência Nacional de Aviação Civil* (ANAC) and Federal Aviation Administration (FAA) regulations, which require that the vehicle stays below a 120m AGL altitude.

The aforementioned factors, namely, cost, weight and payload limitations, reliance on public low resolution data, mapping quality and regulations compliance lead us to the following conclusion: it is beneficial to extract the maximum amount of information from UAVs commonly used sensors, which inspired us to use the image provided from a gimbal stabilized camera to estimate its Above Ground Level (AGL) altitude and later propose a simple, yet effective height regulator.

This work is also closely related to a research project between Vale Institute of Technology (ITV) and the Federal University of Minas Gerais (UFMG) which intends

to explore the use of UAVs in mapping applications on mining environments, such as the provision of Digital Elevation Models (DEM) or magnetometric surveys, which would greatly benefit from low, constant AGL altitude flights.

After we settled our problem and objective we performed a brief literature review and discovered works in the field of biology that suggest that flying insects and birds use optical flow to navigate and control many aspects of their flight, including the height. Since millions of years of evolution have proven the robustness of the technique we decided to begin our research towards height estimation and regulation from that starting point.

In this work we present a complete terrain following framework composed of an experimental UAV platform specifically designed for our work, a height estimation algorithm, a reliability classifier and a simple, yet effective proportional height regulator. The first three were tested in real world, while the last through simulations.

Part of this work, terrain classification, has already been published in a joint event between the 12th Latin American Robots Symposium and the 3rd Brazilian Symposium in Robotics [Campos et al., 2015].

1.1 Problem and Objectives

More precisely, our problem is to perform terrain following flights on UAVs using the hardware already available in these devices. As they typically carry a stabilized camera to perform mapping tasks, we decided to evaluate the feasibility of a monocular vision based height estimation algorithm. The problem is illustrated in Figure 1.1.

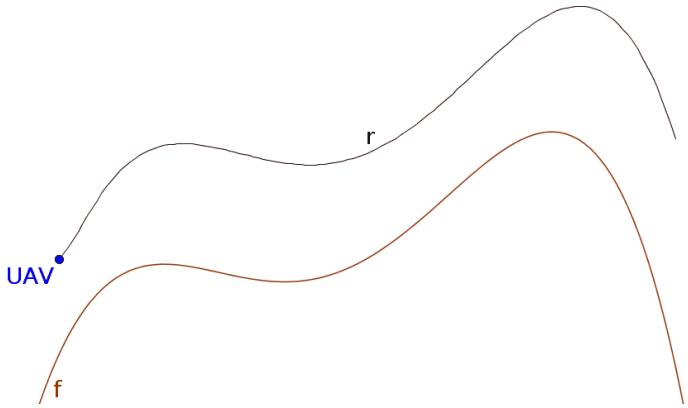


Figure 1.1: Terrain following problem. The problem is to make the vehicle follow the route defined by r, thus flying at a constant AGL altitude, where the terrain is represented by the function f shown above. (This figure can be animated on Adobe Reader on desktop computers by a click.)

Our goal was to develop a small, robust, low cost UAV and make use of its sensors to extract the as much information as we could and enable it to perform mapping missions over mountainous terrain while complying with present aviation regulation rules by keeping an approximately constant AGL altitude.

To achieve that we needed to estimate the height, compute the error to the desired predefined height and perform a terrain following flight, which we decided to do through the use of intermediary waypoints updating the vehicle's path to make it converge to a desired route. Therefore we ended up with the following specific goals:

- Design and construct a small, robust, low cost UAV platform;
- Estimate the UAV's AGL altitude from monocular vision, using optical flow and motion sensor data given by GPS and IMU;
- Perform terrain following flights by remotely controlling the UAV from a ground station computer through autonomous algorithms.

1.2 Organization

The remainder of this thesis is organized as follows: Chapter 2 contains a brief literature review. Chapter 3 describes the design and characteristics of our UAV platform. Chapter 4 details our terrain following methodology. Chapter 5 explains how our experiments were set-up and shows our results. Finally, Chapter 6, presents our conclusions.

Chapter 2

Related Work

In this chapter we discuss the most representative work in the biology, biorobotics, and robotics fields related to the use of optical flow for navigation and distance estimation. To better organize the present chapter we separated it in the following sections: Biology, Biorobotics and Robotics. The former section shows works about flying insects and their theorized sensing and flight control mechanisms. The second is about research that reproduced the insects behavior in lab controlled environments. The latter discusses works in the robotics field.

2.1 Biology

To the best of our knowledge, Horridge [1986] conducted the first study which suggests that insects use velocity parallax —similar to optical flow— to control their flight characteristics. He conducted many experiments with mantids to establish a theory of insect vision. To evaluate every possible mechanism, his methodology consisted of blackening a single eye of some of the insects with Indian ink, to whether identify if they rely on some kind of stereo vision. The environments were controlled and composed of high contrast objects, mainly black and white strips of cardboard.

The experiments consisted on letting the mantids walk on a stick until its end, where the mantid needed to chose another stick to move to. To do that, mantids perform peering movements with the head back and forth in many directions before reaching the next stick, which they chose to be the closest one, independent of their sizes. Some insects even flew when there was no stick in range. An important conclusion was that some insects would wrongly try to reach far twigs if they presented movement, which indicates they estimated the distance incorrectly due to local motion. He also observed the same behavior in both, single and double-eyed insects.

These observations encouraged him to postulate that insects' vision is based on velocity parallax, a term he used to describe the apparent difference in velocity between objects and background. Essentially, nearer objects move faster against the background than distant ones. This behavior is also irrespective of eye rotation since the eye would present the same angular velocity relative to every object. The author concludes his work suggesting insects use only motion perception to navigate around the world, instead of reconstructing the scenes like more sophisticated animals do. Lastly, he emphasizes the potential of such visual system for robotics applications since these insects' neurons are much slower than electronic components.

Later, Horridge's work continued with his colleagues Lehrer et al. [1988]; Srinivasan et al. [1989], in which they studied bees. They investigated claims that bees use the apparent size of familiar objects as well as their apparent motion to estimate the distance to them. They concluded that bees actually use the object motion across the retina to estimate distances rather than size clues. To achieve that conclusion they isolated size clues by performing several experiments in controlled environments.

Their experiments consisted of setting up a white meadow with several black circles representing flowers. To train the bees they set up a random flower (in both size and position) at a higher height than the others with a prize consisting of sugar water. After a few training sessions they let the bees in a test meadow without prizes and observed in which flowers they would land. They found that the landing distribution was highly correlated to the flower height ($r = 0.97$, $P < 0.01$), therefore concluding that the bees could estimate the relative heights of the flowers irrespective of their size.

Furthermore, they still investigated the role of bees' photoreceptors in the distance estimation task. To do so they set up another meadow, this time it was yellow and the flowers were blue. They could adjust the contrast in the green channel of this meadow by varying the intensity of the colors, therefore they could appear both to have the same intensity to the bees green photoreceptors. What they found is that with high contrast bees would perform the same as in the black and white meadow, whereas without contrast, they would land randomly at any flower, disregarding their height. This evidence lead to the conclusion that only the green photoreceptors of the bees are involved in the distance estimation process.

Finally, they decided to train the bees to intermediate height flowers to eliminate any bias toward the closest flower to the bee. They did that by setting a couple of perspex sheets at different heights with a flower glued to its bottom, but this time the reward was on the top perspex sheet, directly over the flower on the intermediate perspex sheet. Their findings corroborated the initial conclusion that they would indeed look for the flower they were trained too, instead of the highest one. By adjusting the

perspex sheets' heights they also found that the minimum height a bee could distinguish flowers from was of around 2 cm.

2.2 Biorobotics

In the biorobotics field, researchers have been trying to reproduce insects' and other animals behavior with robots. A particularly interesting study was first published in 2002 and continues to be developed today [Netter and Francheschini, 2002; Ruffier and Franceschini, 2004, 2005; Franceschini et al., 2009; Expert and Ruffier, 2015]. To emulate the insects' eyes, they designed a camera consisting of a 20 pixels linear array and used a plastic aspheric lens at a deliberately defocused focal length to blur the image adjusting the sensor response to their desired parameters. They assembled this camera on an aircraft attached to a mast through an arm, making it free to rotate around the mast at a fixed distance and limited heights to avoid any crashes with surrounding objects.

The aircraft used a single rotor to produce thrust and lift. To move forward, it angled the rotor toward the desired flying direction through a fin. To provide the vehicle with a rapid thrust response for the terrain following tasks they used a variable pitch propeller, which can quickly alter the thrust regardless of rotational inertia by changing the blades' angle of attack. Their camera had a 75º Field Of View (FOV) and was pointed to cover the forward and downward regions of the craft, enabling it to avoid frontal obstacles. Since the optical flow of the forward region is much lower than the downward, they used a weighted function to compute the optical flow average with greater weights to the frontal pixels.

To perform an autonomous flight the aircraft had a predefined takeoff sequence which consist of slowly increasing the lift and inclining itself until there was enough motion detected on the optical flow sensor. To generate features for the optical flow sensor the area directly below the vehicle flight path had black and white strips perpendicular to its flying direction. Their experiment consisted of using a ramp with a 30º inclination to create an elevation that was to be detected and avoided by the aircraft. At last, they successfully demonstrated that optical flow can be used to allow flying vehicles to perform terrain following flights in a lab controlled environment.

In the continued work of the research group [Franceschini et al., 2009], besides the thorough study on the biology of the insects' eyes and the aforementioned findings they also performed a behavioral analysis of their robots with insects flying patterns and found them to be consistent. Other interesting characteristics are when insects fly

against wind or over specular surfaces. In the former case, when the wind increases the insects' ground speed reduces, but it is not aware of that, since it can only estimate its airspeed, with the lower ground speed, the optical flow lowers, and the insects descend to keep the target apparent motion. On the latter one, without features at the specular surface —such as a ripple-free lake— the insects detect zero optical flow and descent until they crash onto the surface.

These findings represent two important characteristics for flying robots. The first one can be easily overcome by using the GPS, which will accurately provide the ground speed of the vehicle, whereas the second one is a more complex problem that could use other sensors that do not rely on the image to prevent catastrophic failures, such as a sonar. However it is unlikely that UAVs will fly over an extensive featureless area.

They also performed a couple of experiments with flying robots and optical flow sensors. The first one consists of a simulation of an aircraft using two optical flow sensors facing opposite directions, positioned at the sides of the robot and flying it in corridors. The controller compares both optical flow averages and move the vehicle to the left if the left sensor has a lower optical flow and, conversely, to the right if the right sensor presents a lower value. This produced a simulated behavior similar to that of bees when flying in narrow spaces, which results in a centering behavior so that both sensors present roughly the same optical flow. This depends only of optical flow values from both sensors, disregarding the speed or inertial measurements from the vehicle.

The second experiment consists of a robot using a two pixel sensor facing forward to track a single edge and adjust its yaw to follow the bar. They have shown the robot is capable of tracking 0.1^{o} edges at an angular speed of up to $30^{o}/s$. This robot, however, relies only on rotational optical flow, therefore it cannot estimate the distances to the objects. Inspired by the fly's visual system they assembled the optical flow sensor on a rotating platform that allows it to move a few degrees independently of the vehicle. This was fundamental to allow the robot to achieve hyperacuity, enabling it to track objects much smaller than the sensor's detection angle.

The authors conclude emphasizing the robots are built for specific tasks and integrating all the characteristics it would be possible to fly a small UAV indoors or in complex terrains such as mountainous canyons and urban environments autonomously.

2.3 Robotics

Chao et al. [2013] performed a survey of optical flow techniques for robotics navigation applications. First they define optical flow and give a brief explanation of its biolog-

ical inspirations, then they describe the most commonly used optical flow algorithms for UAV applications. Among others, there are Horn and Schunck [1980], Lucas and Kanade [1981] and feature based methods. Later they list some COTS hardware available to compute the optical flow, amidst them, the PX4FLOW, a sensor consisting of a global shutter camera and an IMU integrated to an ARM based microcontroller, which could benefit future development of this present work.

After mentioning the related work they conclude showing the current status and future research directions. They cite works in collision avoidance based on optical flow sensors from computer mice [Griffiths et al., 2006], altitude hold and obstacle avoidance [Geoffrey L. Barrows, 2001; Zufferey and Floreano, 2005], velocity and height estimation with IMU data fusion to achieve robust pose estimation during GPS dropouts [Ding et al., 2009], vision odometry [Hu et al., 2009; Ross et al., 2012; Song et al., 2011], hovering and terrain following [Garratt and Chahl, 2008; Humbert et al., 2005; Romero et al., 2009; Ruffier and Franceschini, 2005], and finally, hovering and landing [Herisse et al., 2008, 2012].

The authors also emphasize the challenges of standardized quantitative evaluation of optical flow systems, the complexity of the computer vision algorithms, specially with the high-resolution hardware currently available, which might require General Purpose Graphics Processing Unit (GPGPU) to allow computational scaling, and, lastly, the use of novel vision systems such as integration between several cameras that provide different views.

Zufferey et al. [2010] used optical flow sensors to perform collision avoidance of a fixed-wing aircraft flying outdoors through a predefined path using GPS. Their UAV uses several computer mice optical flow sensors at the tip, placed in an arrow-shaped pattern at different pitch angles, the one at the tip pointing down most and the farthest ones pointing forward-most, but parallel to the arrow edge. Their technique consists of mapping the optical flow to control signals to perform collision avoidance of the aircraft. To do that they first need to eliminate the rotation induced optical flow, that is achievable by using gyroscope data to subtract it from the measured flow. They weigh the optical flow estimates in a biologically inspired approach to achieve the desired control signals. Their flight controller consists of regular GPS waypoint following, with the exception when it finds a nearby object and applies an appropriate maneuver to avoid it.

The experiments took place in an approximately flat terrain region with two areas containing trees as obstacles to be avoided. They performed two 25 minutes flights in fully autonomous mode, resulting in 32 km and 46 laps. The vehicle flew at approximately 9 m above the ground and successfully avoided every potential collision

with front-lateral and frontal obstacles, at the former performing a slight pitch-up and roll-left maneuver and at the latter a stronger pitch up and slight roll-right maneuver. The authors, however, do not state how the resulting height was verified. In addition, to achieve their goal they used several low resolution optical flow sensors instead of a single camera that could provide both, optical flow data and aerial mapping images.

Herisse et al. [2010] used an optical flow based terrain following strategy for a Vertical TakeOff and Landing (VTOL) UAV using multiple views. Their work consisted of using two cameras, one facing down and one forward, mounted on a quadrotor. While the paper focused more on the control laws, the strategy is the same —used optical flow to avoid obstacles. Their experimental setup consisted of the UAV equipped with two video transmitters that send the videos to an offboard computer responsible to calculate the proper control signals and send them to the vehicle through a radio telemetry link. In their experiments they used the IMU to estimate the velocity and position of the vehicle, since it is indoors, limiting the use of GPS, therefore their flight tests were of short duration due to the increasing errors from integrating these values.

Soccol et al. [2007] devised a visual system for optic flow based guidance of UAVs. A radical difference in his approach is that they use a spherical mirror in front of the camera and remap the image to remove the distortion. This way what appears below the horizon is at the centermost half of the resulting image and what appears above the horizon is at the sides of the remapped image. They also remove the rotation induced optical flow by subtracting the scaled yaw component. The experiments consist of using a robotic gantry to move the camera setup in a lab controlled environment and have shown that after remapping and removing the rotational optical flow the computations are quite accurate. We believe that image remapping and the use of scaled rotational flow corrections could benefit further development of this work.

2.3.1 Height Estimation

One of the most thorough works on height estimation is a licentiate thesis on passive aircraft altitude estimation using computer vision [Moe, 2000]. The goal was to use passive sensing instead of the commonly used radar technology to estimate the vehicle's height to reduce the chance of military aircraft being detected. Among the main contributions, the author highlights the ability to choose the interval between the frames in optical flow estimation based on aircraft altitude and motion and the use of camera motion information to improve the optical flow computation.

The author begins by listing several possible ways of estimating an aircrafts height and explains his choice of monocular vision. Laser, RADAR or ultrasound rangefinders

are quickly ruled out because they are active sensors, just as the use of range cameras depending on active emitting devices. The remaining alternatives are out-of-focus blur, stereo vision, and structure from motion. The first is known not to provide good results. Stereo vision is impractical at an aircraft because of the large baseline required, which means that the cameras would need to be put at the tips of the wings, but these are flexible structures, unable to maintain a rigid calibration of the stereo pair. The remaining method is structure from motion, which the author distinguishes in two subtypes: unknown motion and known motion. The former is not considered since it only provides scale information, is more computationally demanding and not adequate, since camera motion is already known. The investigated approaches are based on optical flow and region tracking.

Some possible advantages of optical flow are that it provides a dense grid of height estimates, it is suitable for detection of obstacles or moving objects and the amount of computation needed is constant for each frame. While region tracking could benefit from more accurate depth estimates, it has a variable number of computations that could be decreased if necessary and is suitable for object tracking.

Moe makes important considerations about the errors that could be induced due to wrong aircraft motion estimation. There are two kinds of errors, translational and rotational. The first one is less preoccupying since the estimation error is linearly dependent on the translation estimation error, which, according to him, is likely to be small due to present technologies in localization and sensor fusion. The second is more problematic since it is non-linear and a 0.03 degrees would already result in considerable differences. To reduce this error one would want to maximize the translation between subsequent images, which can be done by pointing the camera straight down; on the other hand this would not allow the aircraft to detect obstacles far ahead, therefore the camera is positioned facing forward, with a small tilt down.

To reduce the error the author introduces a temporal filtering mechanism which consists of selecting the appropriate frame interval depending on aircraft motion and altitude. Two kinds of filtering are considered, weighted least squares and a Kalman filter, the former being used for optical flow due to its lower computational cost and the latter for region tracking, due to its better accuracy and smaller entries.

The experiments were both a controlled test sequence and using real flight data. The controlled test consisted of a box under a piece of cloth captured by a camera. Seven images were used and in this simple experiment the error was below 1%. The real flight experiments used four different footages, from a small archipelago, above a church, at the runway of an airport and the last a maneuver sequence. The first was chosen since it provides a fairly plane terrain, the second to test the opposite situation,

the third because it is of interest to possibly help in landing procedures, and the last to test the behavior with high rotation between frames.

In the archipelago sequence the aircraft flew at approximately 820 m altitude and the optical flow deviations from radar were around ± 10 m while the region tracking were approximately ± 5 m. On the church sequences the errors were greater due to the ground not being plane, they were around 10%. On the runway sequence the errors were around 2%, again indicating that it works more accurately over plane terrain. The last sequence, with maneuvers, has errors of up to 30%, which indicates the technique is not suitable for high rotations.

Another important aspect of this work is that it estimates the height of ground far ahead, instead of below the aircraft, which means that it does not really estimate the vehicle's height. To compute the actual height the values would need to be stored according to the distance from the aircraft and later used, when the aircraft reaches the position directly above the previously estimated height. It is shown, however, that it is possible to estimate the height even under suboptimal conditions, such as the camera facing forward. However, since it depends on previously estimated heights, the technique does not work when the aircraft performs a maneuver deviating from the original path, such as a turn, therefore it is not suitable for some situations like under the radar flights while following the course of a river or canon.

Garratt and Chahl [2008] conducted field experiments of an UAV terrain following flight using optical flow. They used a Yamaha RMAX helicopter carrying a camera, a laser range finder to be used as ground truth and a differential GPS unit to account for the camera translation. Their algorithm, however, does not account for the vehicle's vertical motion. In their experiments, the flight was conducted over flat terrain and the AGL altitude was only 2 meters. With this setup the error was around 10%. This prevented the analysis of the algorithm behavior at higher altitudes and at varied relief, such as a mountain.

Differently from these approaches, our research consists of developing and using a low cost UAV platform to perform height estimation and regulation. Other contributions are the classification of our estimation reliability through decision trees; the proposed height regulation technique, which relies on waypoints instead of thrust manipulation, thus better for unreliable communication channels; and the consideration of the vertical displacement of the vehicle in the optical flow computation, which is to be expected in situations that require terrain following algorithms. Since we are using COTS hardware we had to use a gimbal stabilized camera as the image sensor features a running shutter, which would make it impossible to digitally stabilize the image or use it to estimate the translational optical flow through derotation techniques.

Chapter 3

Unmanned Aerial Vehicle Platform

To perform the experiments and evaluate our methodology we designed and constructed an UAV Platform. Our target was to build a low-cost vehicle that was easy to assemble and configure with COTS parts. Furthermore, the UAV needed to be capable of performing autonomous missions preferably using open source software for better support, in case we needed to modify any part of the program. In this chapter we present the many design considerations and justify our choices. For assembly and configuration instructions of the vehicle please check Appendix A.

3.1 Airframe

The main decision was the airframe. At first we needed to decide what kind of vehicle we would use, then choose the specific airframe model. The selection of every other part must be in accordance to the airframe capacities and capabilities. We decided to use a multirotor platform because they have Vertical TakeOff and Landing (VTOL) capability, allowing operations in rough terrain, are simpler to operate and maintain while also being safer than traditional helicopters due to the much smaller propellers. As we were targeting a low-cost vehicle, among the multirotors platforms we chose the quadrotor, as it uses less motors, therefore being the least expensive.

Specifically, we chose an inexpensive and durable quadrotor frame, based on the DJI-F450 (Figure 3.1), as this frame had appropriate dimensions and payload capacities for our experiments. This is probably one of the most popular quadrotor frames available, therefore parts availability and compatibility would not be a problem.

Figure 3.1: HK-450 Quadrotor frame. The DJI-450 frame and its derivatives are very commonly used, which makes parts availability and compatibility much easier. The 450 stands from the diagonal distance between motor shafts, this is a medium frame.

Source: `hobbyking.com`

3.2 Motors

The motors were the second decision, and are a direct consequence of the airframe choice and desired payload capacity. The airframe typically recommends a few choices of motor and propeller sizes, which is stated respectively in Kv, a measure for each thousand RPM per volt applied to the motor without any load, and inches x inches, which represent the propeller diameter and pitch. We selected the most powerful motors available since we foresaw that our resulting prototype would possibly be heavy with

Figure 3.2: NTM-2826 1000Kv brushless motor. This motor presents the advantage of being capable of working with 3 cell and 4 cell batteries. The latter would allow a greater payload capacity of our prototype case necessary.

Source: `hobbyking.com`

all the equipment included. The motors were the NTM-2826 1000Kv (Figure 3.2). Another advantage is that these motors can operate with 3 or 4 cell batteries, so there was still room for added performance by just switching the batteries if necessary.

3.3 Propellers

The propellers must be compatible with the selected motor. Since they are inexpensive, we selected propellers of size that produced the highest thrust, but in three different materials to later choose which one to use. To evaluate the propellers we set up a thrust testing bench with a digital kitchen scale and a motor attached to a propeller on top. With the thrust vector going up, when we increased the throttle the weight on the scale increased, thus avoiding a lift overestimation caused by ground effect. We also evaluated the quadrotor behavior with each propeller type. Our findings are listed below:

Carbon Fiber

Carbon fiber propellers are the most rigid and expensive ones. They are durable and resulted in very quick response from controller inputs in flight, but they produce less thrust, resulting in a bigger power consumption, consequently reducing the effective time the vehicle could stay aloft. As a consequence of their stiffness, they transferred most of the impact force from crashes to the motor shaft, thus resulting in premature failure of the bearings;

Rigid Plastic

Rigid plastic propellers were inexpensive and produced similar thrust to the carbon fiber ones, but they turned out to be very brittle, breaking at minimal contact with the ground;

Flexible Plastic

Flexible plastic propellers produced more thrust and suffered the least damage, but presented the worst response to controller input. At 90% of throttle a single motor produced 900g of thrust on a 3 cells battery set-up. We have not broken a single one in our testing, and they are still usable after severe crashes.

Since our experiments would be conducted outdoors, allowing larger margin of error than their indoor counterparts, we prioritized robustness and endurance over controller responsiveness. Therefore we selected the flexible plastic propellers of 10x4.5" for our UAV platform (Seen in Figure 3.3). The orange color helps visualizing the propeller when it is spinning.

Figure 3.3: 10x4.5" orange flexible plastic propeller set with black shaft adapting washers.

Source: `hobbyking.com`

3.4 Electronic Speed Controllers

To control the brushless motors we needed Electronic Speed Controllers (ESCs). These devices are responsible for alternating the current between the motor phases to make them spin. ESCs are purpose built devices: there are specific variations for Remote Controller (RC) model cars, boats, planes, helicopters and, recently, multirotors. In cars and boats they are bidirectional and can be water cooled in the latter; in planes they have a slower response; in helicopters they start the motor slower and usually keep working at an approximately constant speed —the reason for the last is that helicopters actually change their thrust by varying the angle of attack of the wing blades instead of its speed.

In multirotors, however, ESCs need to be unidirectional and provide the fastest response possible since these vehicles use fixed pitch propellers and depend on quick changes of thrust to fly gracefully. The ESCs need to be able to handle the necessary load to spin the motor with the appropriate propeller and exceed this requirement to handle surges, our power train consumes 15 A at most. One of the challenges we faced when we assembled our model was that there were no COTS ESCs available for multirotors, although, a few enthusiasts figured they could reprogram their ESCs

and started developing firmwares specifically for multirotors. Therefore we bought RC plane ESCs and reprogrammed them with an adequate firmware for our vehicle. Considering our requirements, the model we chose was the Turnigy Plush 30A ESCs (Figure 3.4) with SiLabs microcontrollers and the firmware we flashed it with was the BLHeli v11.2 with the stock multirotor settings.

Figure 3.4: Turnigy Plush 30A Electronic Speed Controller. This model has different components dependent on its manufacture date. Our features SiLabs microcontrollers and both NPN and PNP transistors. Like our motors, these ESCs can work with both, 3 cells and 4 cells batteries.

Source: `hobbyking.com`

3.5 Flight Controller

Some vehicles, such as planes and helicopters, dispense a flight controller for basic flying and can be directly actuated by the RC controller channels outputs. On the other hand, multirotors require flight controllers even for the most basic and manual flights. This happens because they are unstable platforms and require stabilization all the time, which is achieved by the use of integrated IMUs. There are several different flight controllers available for multirotors, but only a few of them provide the higher level functionalities required by our project.

The basic flight controllers use only a simple IMU (gyroscope and accelerometer) and microcontroller to stabilize the vehicle in flight, usually allowing for two different flight modes, manual —which takes the controller sticks positions as the desired tilt angle and self-levels the multirotor— and acrobatic —which considers controller input as a desired angular rate and keeps the last tilt angles when no input is provided.

These flight controllers usually use 8-bit microcontrollers and are really inexpensive (commonly found for less then 20 USD).

Intermediary flight controllers provide the same functionalities as basic ones with the addition of a couple more sensors, being a magnetometer and a barometer, which helps the vehicle to lock at a specific heading and altitude. Sometimes their functionalities can be expanded by connecting external devices such as GPS receivers, allowing higher level tasks, such as loitering and return to home, to be performed. These boards commonly come with 32-bit microcontrollers and cost less than 50 USD.

Advanced flight controllers usually come in a kit with all the necessary hardware; in addition to the intermediary functions, they also allow waypoint navigation. They are usually based on the well consolidated and open source and hardware ArduPilot Mega (APM) flight controller, which uses an 8-bit microcontroller and is usually available in kits that cost 120–150 USD depending on the available sensors, that can include power monitoring, radio and video telemetry devices. Its functionalities can also be expanded through the use of optional sensors such as sonar and laser range finders. These controllers are capable of performing a fully autonomous mission, but unfortunately the APM firmware is no longer being further developed.

State of the art flight controllers present all the functionalities found in advanced ones with the addition of sensor redundancy and data storage, usually presenting two sets of IMU ICs and GPS receivers. They can also use their memory to store terrain elevation information from the Shuttle Radar Topography Mission (SRTM) and perform terrain following flights using previous knowledge. These flight controllers are usually based on the 32-bit PIXHAWK autopilot, which is an open-source and open-hardware project with infrastructure supported by the Computer Vision and Geometry Lab at ETH Zurich[1]. Optional sensors for these flight controllers include the aforementioned range finders and also an optical flow sensor, which is used to horizontally lock multirotors and helicopters in place when no GPS signal is available. These flight controllers cost 200–400 USD depending on the available sensors.

Since we were developing a low-cost UAV Platform, we selected the APM (Figure 3.5) as the flight controller, as it was still being actively developed at the time and presented all the necessary functionalities for our project. It also has the advantage of being an open project with a huge and helpful community. This allowed us to check the source code and schematics several times when we wanted to implement additional functionalities in our prototype.

[1] `https://pixhawk.ethz.ch/`

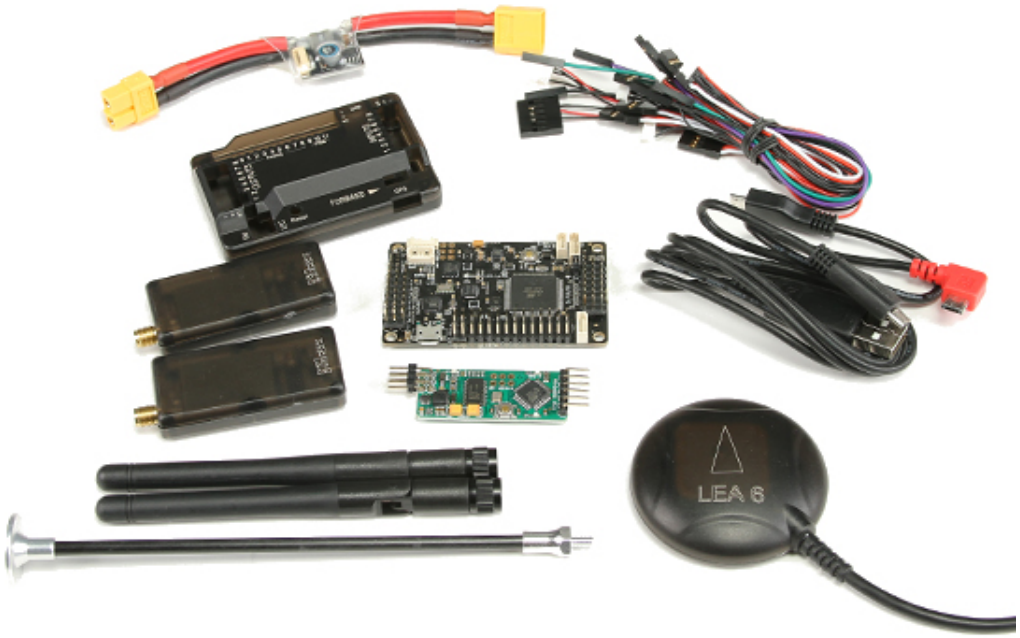


Figure 3.5: APM 2.6 flight controller kit with GPS, telemetry radios and miscellaneous parts.

Source: `hobbyking.com`

3.6 Camera, Gimbal and Video Transmitter

Since our methodology receives images as input we needed good quality images provided by a lightweight and robust camera that could be easily attached to UAVs using COTS parts. The GoPro cameras just happened to fill these criteria and was the obvious choice for our vehicles. However, since the quadrotor is an unstable platform, we needed to guarantee that the images would be free of artifacts caused by vibration and quick rotation of the vehicle. Therefore we decided to use an actuated gimbal to stabilize the image. We also found that it was important to use a video transmitter considering our goal was to perform terrain following flights from monocular vision and to keep cost low, we would probably need to perform the computation off-board.

There are two kinds of actuated gimbals:

Servo Gimbals

Servo gimbals just use servos to control the camera position. These are really cheap (around 20 USD) and rely on the flight controller attitude computation and control signals to properly stabilize the camera. Since servos have high torque

they usually work well with different sized cameras, but they may vibrate due to signal noise and therefore do not provide the sharpest image.

Brushless Gimbals

Brushless gimbals are named so because they use their homologous motors. These are much more expensive than servo gimbals and use vibration dampers and an IMU attached to the camera to control its position at extremely high rates, hence they do not require flight controller signals to stabilize the camera. But to keep things light these gimbals are crafted for specific camera models and are perfectly balanced to their weight, using the least powerful motors necessary to perform the task. They provide the best image sharpness at a higher price tag (starting at 100 USD).

As we were concerned about image quality we chose to use the brushless gimbal to counter the quadrotor vibrations. There are, however, two different kinds: they can be bi-axial or tri-axial, where the former also stabilizes the camera movements in the yaw axis. We chose to use the bi-axial version because it is lighter. To avoid yaw we performed our experiments using translation only and locking the heading with the help of the magnetometer. We ended up using a GoPro Hero 3+ Black (Figure 3.6a) attached to a Quanum Q-2D gimbal (Figure 3.6b) and a 900MHz 200mW analogue video transmitter (Figure 3.6c). To mount this hardware beneath the quadrotor we also needed to use longer landing skids. We bought the lightest model we could find for less than 10 USD (Figure 3.6d).

3.7 LIDAR Rangefinder

To validate our experiments with an accurate ground truth we decided to use a LIDAR Rangefinder (Figure 3.7) specifically developed for these small UAVs. We installed the PulsedLight LIDAR-Lite in our model alongside the GoPro camera. Thus, with proper alignment we would have the distance where the camera center is pointing. To keep the gimbal balanced we added some counterweight on the opposite side of the sensor. The APM firmware version 3.2.1 already has the source code to read the values from the rangefinder, but does not perform terrain following through it, which is just what we wanted to validate our estimation methodology.

(a) Camera.

(b) Brushless gimbal.

(c) Video transmitter and receiver.

(d) Landing skid.

Figure 3.6: Video hardware. (a) GoPro Hero 3+ Black camera, (b) Quanum Q-2D gimbal, (c) generic video transmitter and receiver pair from Hobbyking, and (d) generic landing skid from Hobbyking.

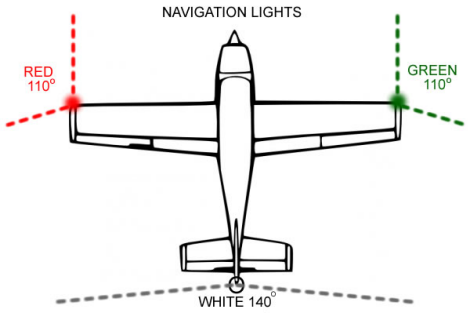
Sources: `fotostore.rs` (a) and `hobbyking.com` (b,c,d)

Figure 3.7: PulsedLight LIDAR-Lite laser rangefinder.

Source: `pulsedlight3d.com`

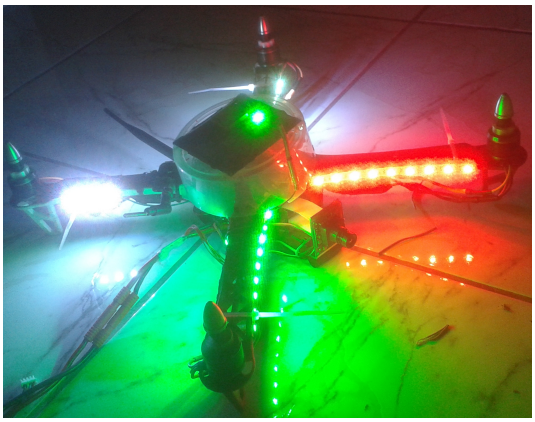
3.8 Navigation Lights

Since we were performing some waypoint based navigation experiments (not terrain following) at night we decided to add navigation lights to the UAV as a safety measure. As there is no defined standard for multirotor's navigation lights we decided to install them based on airplane standard navigation lights (Figure 3.8a), therefore we have red lights at the front left arm, green lights at the front right arm, and white lights at the back arms of the vehicle. They are made of led strips (Figure 3.8b). The resulting setup is shown in Figure 3.8c.



(a) Airplane navigation lights.

(b) Led strip.



(c) UAV lights.

Figure 3.8: Navigation lights.

Sources: `learntofly.ca/aircraft-navigation-lights/` (a) and `hobbyking.com` (b)

3.9 Remote Controller and Receiver

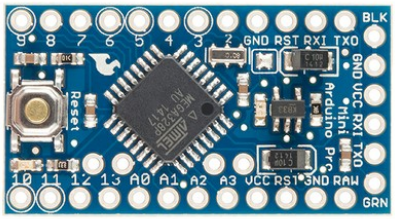
We bought an inexpensive controller —the Turnigy 9x (Figure 3.9a)— with support for rechargeable batteries and heavily modified it. We installed an additional transmitting module inside the case (Figure 3.9b), removed the antenna from the factory provided module, attached it to the outside module and installed a switch to select whether we wanted to use the internal or external module. The internal module we installed has support for telemetry, but we needed to reroute a few lines in the RC main board in order to free a serial connection from the microcontroller, that also required that we flashed it with a custom firmware. We also added a backlight to the display, which then allowed us to use it at night.

To use the telemetry functions on the RC we also needed to use an Arduino Pro Mini (Figure 3.9d) alongside a compatible receiver (Figure 3.9c) on the quadcopter, which was programmed to sniff and translate the packages sent in the MAVLink protocol from the flight controller to the 3DR telemetry radio into the receiver's telemetry protocol. This allowed us to monitor battery levels and GPS information —among other things— from the RC screen, increasing the safety of the UAV and allowing us to easily locate the vehicle in case of catastrophic failure.

(a) Remote controller.

(b) Transmitter.

(c) Receiver.



(d) Arduino Pro Mini.

Figure 3.9: Transmitter and receiver control hardware. Turnigy 9X remote controller (a), FrSky DHT DIY telemetry transmitter (b), FrSky D8R-XP telemetry receiver and Arduino Pro Mini (d).

Sources: `hobbyking.com` (a,b,c) and `arduino.cc` (d)

3.10 Batteries and Charger

As batteries are inexpensive we first bought two different capacities (4A and 5A) from two different manufacturers (Turnigy and Zippy) to experiment and determine which was the best to use. We determined the best battery was the three-cells Turnigy nano-tech 4000mAh LiPo with 25–50 times its capacity as discharge rate. Each costs around 25 USD (Figure 3.10b). We also used a three-cells Zippy 1800mAh low discharge LiFe for the RC, which costs around 12 USD (Figure 3.10c).

(a) Charger and accessories.

(b) UAV battery.

(c) RC battery.

Figure 3.10: IMax X100 Charger (a), Turnigy nano-tech 3S 4000mAh 25-50C LiPo (b) and Zippy Flightmax 3S 1800mAh 5C LiFe (c) batteries.

Source: `hobbyking.com`

The charger we bought is the X100 (Figure 3.10a), an AC/DC balance charger from IMaxRC, this allows us to charge the batteries from the power outlet in the lab or from a car during field experiments and it also has support for both batteries chemistries LiPo, and LiFe. It features a touchscreen display where we can see real time graphs of the charging process, and it only costs around 50 USD.

3.11 Parts Summary

After carefully selecting the previously mentioned components we finally had all necessary parts to build our prototype (Figure 3.11), which consists of the following items:

Table 3.1: Prototype UAV platform parts and cost in USD (Dec 2014).

Item	Model	Price	Qty.	Subtotal
Airframe	DJI F450	11.99	1	11.99
Motor	NTM2826-1000KV	15.99	4	63.96
Propellers	10x4.5 Flexible Plastic	0.75	16	12.00
ESC	Turnigy Plush 30A	12.16	4	48.64
Flight controller + GPS kit	APM 2.6 + Ublox Neo 6M	86.60	1	86.60
Telemetry	3DR Radio Pair	35.88	1	35.88
Camera	GoPro Hero 3+ Black	399.99	1	399.99
Video transmitter and receiver	900MHz 200mW	69.47	1	69.47
Gimbal	Quanum Q-2D	99.99	1	99.99
LIDAR rangefinder	LIDAR-Lite	89.99	1	89.99
Navigation lights	LED Strip	9.91	1	9.91
Remote controller	Turnigy 9x	53.82	1	53.82
RC transmitter and receiver	FrSky DJT + D8R-XP	52.93	1	52.93
Batteries	Turnigy nano-tech 3S 4000mAh 25-50C LiPo	27.30	4	109.20
Transmitter battery	Zippy 3S 1800mAh 5C LiFe	11.25	1	11.25
Charger	IMaxRC X100 AC/DC	49.99	1	49.99
Total				1,205.61

As seen in Table 3.1, the vehicle cost (not including the camera, RC, charger, batteries and video receiver) is less than 600 USD. Nonetheless, it could be further reduced through the use of new parts that only became available after we built our prototype. A comparable UAV platform —the 3DR Iris— costs around 1,000.00 USD in a similar configuration, without the camera. This shows we accomplished our first goal of building a low-cost UAV platform for our terrain following applications. Fur-

thermore, the vehicle's price also makes it suitable for multi-robots or even swarm applications.

Figure 3.11: Our UAV prototype.

Chapter 4

Terrain Following Methodology

As seen in Chapter 2, since the 1980s researchers have found supporting evidence that suggests flying insects and birds use optical flow to perform terrain following flights and some studies even reproduced the behavior in controlled environments. Therefore, as millions of years in biologic evolution and decades of scientific studies already demonstrated this to be a viable approach, we chose to follow this path. A brief schematic showing our methodology is shown in Figure 4.1.

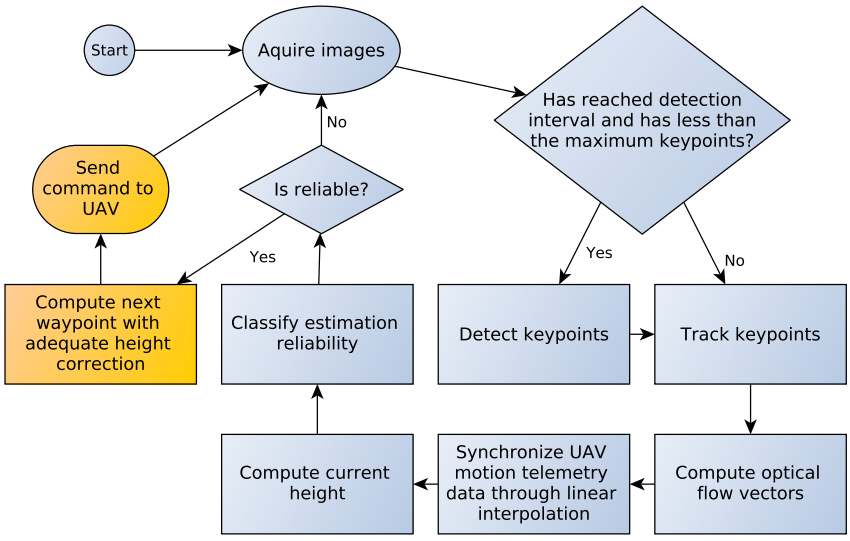


Figure 4.1: Terrain following methodology. Our system takes images as input and detects the keypoints at predefined intervals, afterwards we track them through optical flow, then we take the flow vectors and synchronize the image information with the UAV telemetry data, which allow us to compute the aircraft height. We finally verify if our estimation is accurate through a decision tree classification, compute the necessary corrections to the vehicle altitude and send waypoints commands accordingly. These last two steps were tested in simulation only.

In addition to the optical flow data, we also use the vehicle's motion information in order to estimate the height it is flying at. These data are used to control the UAV by manipulation of waypoints along its path rather than direct changes to its lift by adjusting the throttle. This is because our computation is made off-board through a high delay video system.

Now we present the methodology we adopted to perform terrain following flights through monocular vision using our UAV platform. This chapter is organized as follows: in Section 4.1 we briefly explain what optical flow is and describe the algorithms we used; in Section 4.2 we describe our preliminary investigation to evaluate which would be the best way to approach our problem; in Section 4.3 we demonstrate the core of our work, the height estimation methodology; and finally, in Section 4.4 we show our height regulation technique.

4.1 Optical Flow

One of the greatest challenges of field robotics is to reconstruct the observed scenes of the world into a 3D virtual model that can be used for localization and navigation. The motion field allows us to transform points between the world and the model; however, it is unknown and indeterminable, therefore our best alternative is to estimate it through the optical flow. According to Horn [1986], motion field is composed of the velocity vectors assigned for each image point, while optical flow is the apparent motion of brightness patterns observed when a camera is moving relative to the objects being imaged.

We can not determine the motion field from observed images because sometimes there will be motion, but it will not be apparent, as when the camera is moving parallel to an untextured surface, an observer might know the camera is moving if it is watching the camera motion, however, it can not tell if the camera is moving by just looking at the images it captures. Another less common situation is when there is no motion, but it appears to be. This can happen by varying a focused light source projection area, as if you are moving a flashlight in front of a wall, it would seem that something is moving on its surface, but instead it is just the light. Fortunately these are not very common situations and we can usually estimate the motion field from the optical flow.

Virtually all approaches used to compute the optical flow assume that the brightness is constant around the neighboring region from a given point in two subsequent images, thus we can represent it by:

$$I(x+u, y+v, t+1) = I(x, y, t), \tag{4.1}$$

where I represents the image point at the given coordinates (x, y) and time t while u and v represent the observed displacement of this point at the subsequent image at time $t+1$. This equation is known as the Brightness Constancy Assumption (BCA).

To estimate the motion vector we calculate the first degree polynomial of the Taylor series of the BCA (Eq. 4.1) as follows:

$$\begin{aligned} I(x, y, t) &= I(x, y, t) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}1. && (4.2)\\ -\frac{\partial I}{\partial t} &= \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v. && (4.3)\\ -\frac{\partial I}{\partial t} &= \nabla I \cdot \begin{bmatrix} u \\ v \end{bmatrix}. && (4.4) \end{aligned}$$

The issue with Eq. 4.4 is that it leads to an underconstrained system. There are, however, approaches for adding additional constraints to the system. The first one was developed by Horn and Schunck [1981] and imposes global smoothness, which assumes that optical flow vectors from a given region of the image have similar direction and magnitude, which is mostly true, but fails at edges and corners. The second approach was developed by Lucas and Kanade [1981]. It consists of finding the displacement vector that best fits a window of pixels, imposing local smoothness, and is well suited for tracking features, such as corners.

We used two different optical flow algorithms in this work: the Lucas and Kanade (LK) method and also a newer one created by Farnebäck [2003] (FB), which is based on an approximation of each neighborhood of both frames by quadratic polynomials. Farnebäck proposes an efficient and accurate approximation by using the polynomial expansion transform, providing a fast and robust algorithm to estimate the displacement field between two frames. In addition to the small error of optical flow computed by Farnebäck's approach, we can efficiently compute dense flows using the Compute Unified Device Architecture (CUDA) implementation available at the OpenCV library. Despite the aforementioned advantages, the algorithm fails to find optical flow in areas with low texture. Thus, to minimize the impact of such areas, we only used displacement vectors with a magnitude over a minimum threshold, which was determined empirically. We found that we could filter a good amount of bad vectors and keep most of the good vectors with a threshold of 1 pixel.

In our preliminary tests, we used LK optical flow to perform an empirical analysis with the objective of discerning different terrain types and elevation information by observing histogram patterns of the magnitude and angle of optical flow vectors [Campos et al., 2015]. We arranged the vectors in a histogram of 24 bins for magnitude, each consisting of a 0.5 pixel interval and another of 180 bins for the angles, each consisting of 2 degrees intervals, as shown in Figure 4.2. Based on our observations the terrain types were clearly discernible when the following assumptions were true:

- the scene is mostly static;
- there is small to no camera rotation.

We performed experiments with the UAV at UFMG's Pampulha campus. It carried a downward gimbal-stabilized facing camera and flew over known terrain with measured heights through manual annotations of GPS coordinates while walking over the terrain. Afterwards we studied the data to check if there was any discernible difference when flying at different situations. Our analysis was based in images such as Figure 4.2.

As there were clear differences between the images, we decided to perform an autonomous classification of the terrain, for that we used the FB algorithm to compute the dense optical flow over the entire image and distributed this data in bins as show in Figure 4.4, then we trained a few decision trees with this data and classified it, achieving promising results.

Finally, to properly estimate the height, and consequently regulate it, we needed reliable data and real-time processing. Therefore we decided to use the LK optical flow and reorganized the feature vectors as the average magnitude and angles of all optical flow vectors on the image as well as their standard deviations to classify whether our height estimation was reliable or not.

4.2 Optical Flow Evaluation

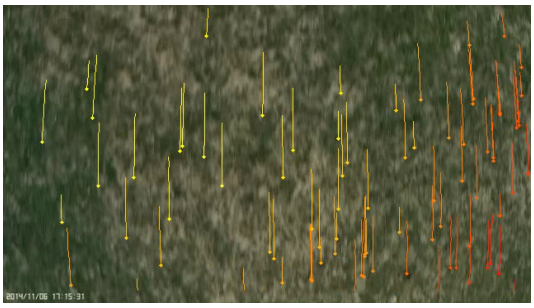
Optical flow provides us rich information about the scene; however, by itself, it provides relative depth information only, allowing us to perceive which objects are closer or farther relative to each other, but not their actual distances to the camera. As a consequence, we knew that to estimate the actual height we would need additional data, nevertheless, we decided to investigate if optical flow only would still provide valuable information to perform the terrain following task.

4.2.1 Feature Analysis

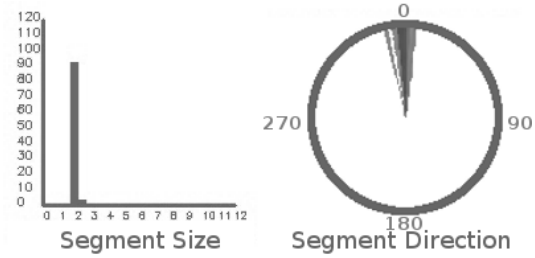
To determine whether we could gather information that would help in the terrain following task we decided to use supervised machine learning techniques. As the final goal is to perform the necessary computations in embedded hardware, we settled for decision trees, since they are extremely quick to classify data.

We initially had do determine what kind of information could be extracted from optical flow that would be possibly helpful in the terrain following task. To begin our analysis we computed the sparse optical flow of flights over diverse regions and

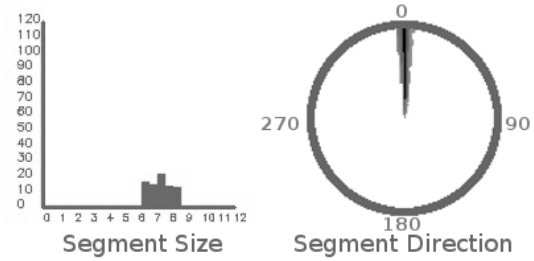
(a) Optical flow of flat terrain image.



(b) Optical flow of inclined terrain image.



(c) Segment size and direction histograms of flat terrain



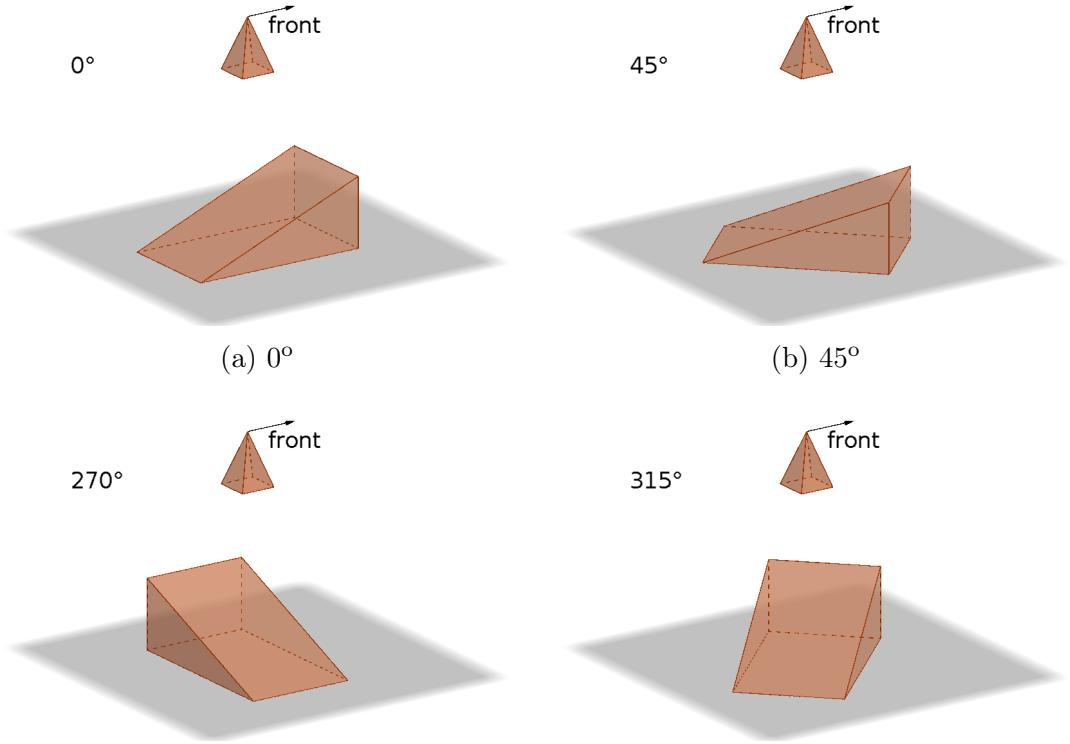
(d) Segment size and direction histograms of inclined terrain

Figure 4.2: Sparse optical flow of different terrain types and their respective histograms. It shows the sparse optical flow of a flat terrain image (a) and inclined terrain image (b) along with their respective histograms (c) and (d). The colors of the flow vectors vary according to their average segment size, bigger vectors are yellow and smaller vectors are red. The histograms represent the vector magnitude (left) and direction (right). In the latter, color intensity represent the distribution among the bins; darker bins represent more vectors laying inside them. In these samples the segment direction is highly concentrated close to 0º, however the segment size histogram show a completely different behavior, being concentrated between 1,5 and 2 pixels in (c) and spread between 6 and 8,5 pixels in (d). This illustrates the expected difference according to the terrain type. We also observed that the data in the segment size histograms would move from left to right (the magnitude would increase) if the terrain was elevating with relation to the vehicle's position while presenting the opposing behavior for dropping terrain.

converted the vectors to polar coordinates in order to organize the data in binned histograms that allowed us to empirically study it to see if we could distinguish different situations. Resulting images from this study are shown at Figure 4.2.

4.2.2 Terrain Classification

We identified three groups of classes that could be easily distinguished upon observing the optical flow data: the first one was regarding the terrain inclination, that is, whether the terrain was inclined, flat or other. The second was a specialization of the inclination class focused on determining its direction (Figure 4.3) while the last considered the terrain elevation, in other words, whether the terrain was going up or down with respect to the vehicle's trajectory.



(a) 0º (b) 45º

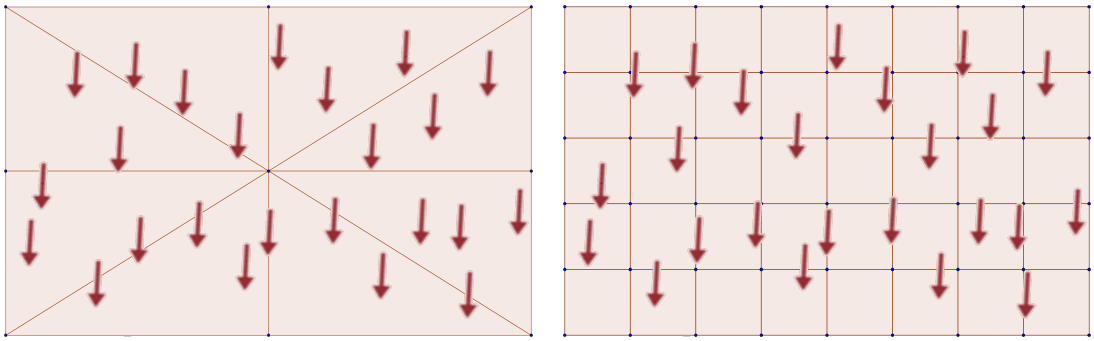
(c) 270º (d) 315º

Figure 4.3: Images showing some examples of direction of terrain inclination.

These groups of classes do not only show whether or not optical flow provided terrain information but are also useful to the height regulator, allowing a quicker response by predicting the height in a near future according to these classifications and taking appropriate actions in advance of changes, or before the error reaches a large enough value.

4.2.2.1 Feature Vectors

In order to classify the data, we designed three different descriptors, two of them more focused on terrain inclination and the remaining on terrain elevation. The first two consist of the discretization of dense optical flow in bins using two different patterns, as seen in Figure 4.4, and the latter consists of a history of the global average magnitude of optical flow vectors in the whole image for the last ten frames, thus resulting in a vector with ten features.



(a) Octant descriptor. (b) Grid descriptor.

Figure 4.4: Patterns determining the descriptors for terrain inclination. They consist of slicing the image in bins determined by the lines and calculating the average magnitude of optical flow vectors in each bin. Afterwards we concatenated this data to generate our feature vectors.

4.2.2.2 Decision Trees

Since we are trying to mimic a natural behavior, the obvious path is to use machine learning. We chose to use decision trees because they are fast, which means they can be easily implemented in the embedded hardware currently available in UAVs. Furthermore, they have shown to be robust enough for our needs.

A decision tree consists of a binary tree where the nodes are chosen according to the information gain of each attribute of a given feature vector in a training set. For the root we want a condition with an attribute with the highest information gain, or, conversely, the lowest entropy, that is, the condition that provides the best split in the data. It repeats this process until it is left out of attributes. Later, to classify data the algorithm compares a given feature vector with the root, then it descends through the nodes according to the previous conditions. The leaves correspond to the classes we seek to organize the data in. The implementation we used is the ID3, as is provided by the OpenCV library.

4.3 Height Estimation

Our main objective is to perform terrain following flights from monocular vision. Therefore, the core of our work is the height estimation algorithm, which computes the height through optical flow together with motion information provided by the UAV's flight controller after GPS and IMU data fusion.

To estimate the UAV's height we inspired our approach on the classical stereo vision technique, however we utilize images provided from a single camera acquired at different times, thus we assume that:

- the scene is mostly static;
- the camera translation is approximately known;
- there is no independent camera rotation.

We use the LK optical flow to perform the correspondence between points at subsequent images and then compute the displacement from the velocities provided by the UAV's IMU in the time between the acquisitions. With this data we finally estimate the vehicle's height. There are, however, situations where the estimation becomes inaccurate, such as when there is very small motion between the frames or when there is too much yaw, therefore we decided to create an additional failsafe by training a decision tree to determine whether our estimation is reliable or not, and only use the estimation when it was classified as the former.

Figure 4.5 observations leads to the following equations:

$$T'_x = T_z \frac{x_1}{f}, \tag{4.5}$$

and,

$$\frac{Z}{T_x - T'_x} = \frac{Z + f}{x_1 + x_2 + T_x - T'_x}. \tag{4.6}$$

When solved for Z they become:

$$Z = \frac{fT_x - T_z x_1}{x_1 + x_2}, \tag{4.7}$$

We note that Eq. 4.7 is not very different from the classic stereo equation, with the exception of the term $T_z x_1$, which is responsible for adjusting the depth estimation with relation to the camera translation along the z axis.

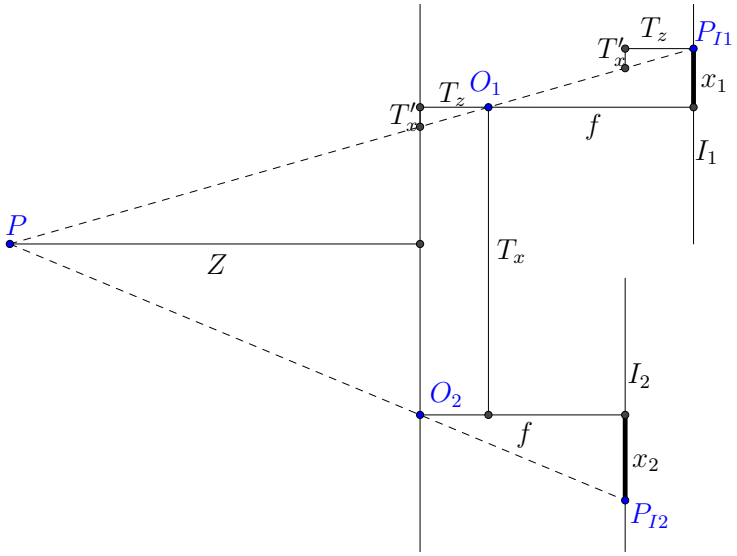


Figure 4.5: Adapted Stereo Schematics for 2D motion. It illustrates a point P projected on two different images. The points O_1 and O_2 indicate the focal points from each acquisition, note the images were acquired at different depths. The points P_1 and P_2 indicate the projected points at the image plane. Z stands for the depth, f for the focal length, T_x and T_z for translations, x_1 and x_2 to the distance from the camera center and T'_x to a projected compliment of the translation along the x axis.

However, Figure 4.5 only shows a 2D scenario of the depth estimation technique, whereas in real world we need a few more steps to correctly calculate distances taking 3D motion into account. Figure 4.6 shows how we compute the depth in a 3D translation scenario.

It is important to note how the translation along the z axis is represented, in this case we consider a negative translation when the UAV moves up and positive when it moves down, this way the coordinates are aligned with the camera coordinate system. A change in this representation would imply a change on the signs on the equation.

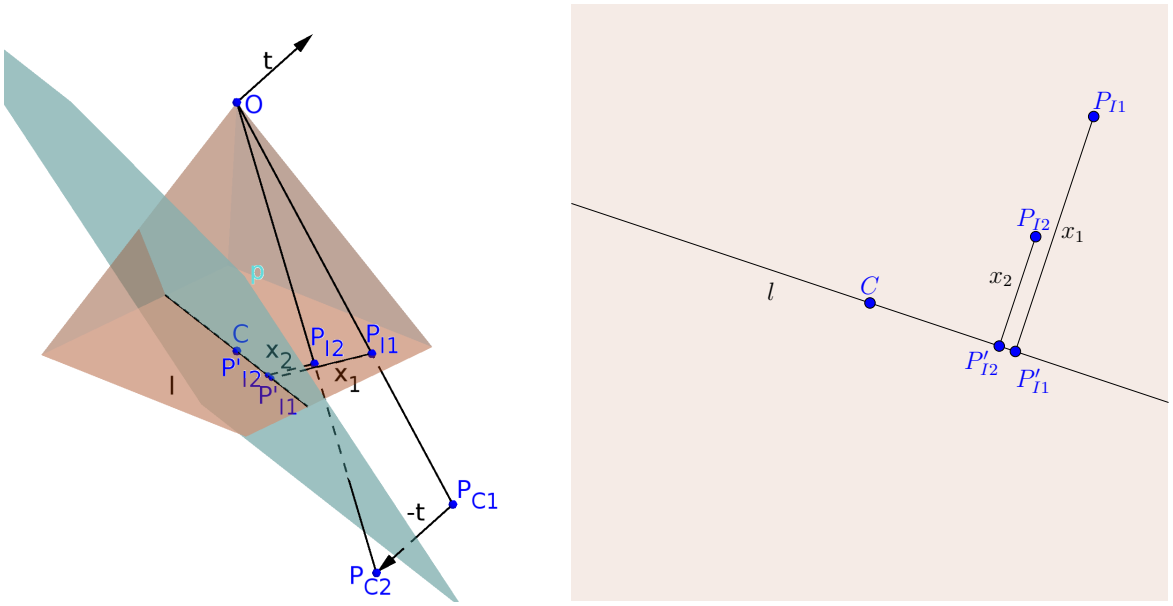
To estimate the depth we then use Eq. 4.9 with the computed values, but we also need to take account on which side of line l the points P_{I1} and P_{I2} lie, changing the sign accordingly. To verify that, we compute:

$$S = sign((\overrightarrow{P'_{I1}P'_{I2}} \times \overrightarrow{CP_{I1}}) \cdot (\overrightarrow{P'_{I1}P'_{I2}} \times \overrightarrow{CP_{I2}})). \tag{4.8}$$

And with the final adjustments to Eq. 4.7 we get:

$$Z = \frac{fT_{xy} - T_z x_1 S}{x_1 + x_2 S}, \tag{4.9}$$

where T_{xy} is the horizontal movement of the camera and T_z the vertical one, such that $T_{xy} + T_z = \vec{t}$.



(a) Projection schematic. (b) Image plane.

Figure 4.6: Adapted Stereo Schematics for 3D motion. (a) shows the projection of a point P_W in two consecutive images after a camera motion represented by $\vec{t}$. For viewing purposes we aligned both camera frames and displaced the point P_W by $-\vec{t}$, represented in the camera coordinate system as points P_{C1} and P_{C2} for the first and second frame respectively. These points are then projected onto the image plane as points P_{I1} and P_{I2}. To properly estimate the depth we project a plane p, which is perpendicular to $\vec{t}$ and passes through point C, which is the camera principal point, and finally compute the distances from P_{I1} and P_{I2} to the line l, which is the intersection between p and the image plane. (b) shows the respective image plane.

4.3.1 Ground Truth Acquisition

To establish a ground truth we installed a LIDAR-Lite laser rangefinder. To make it always point to the same direction as the camera, we attached it to the gimbal alongside the camera, but since the gimbal is projected specifically for a single camera, we had to add some counterweight to restore its balance after attaching the sensor. The sensor values are stored in a flight log that can be used to replay the mission or can be instantaneously read though the serial telemetry link by the ground station computer.

To synchronize the video with the flight log we needed a common time information that could be used to link both data, since the flight logs already register the GPS data, including very precise timestamps we decided to record this time on video before each flight through a mobile phone application that displays GPS information in real time, a rudimentary process that has proven sufficient for our needs.

After synchronizing the data we could use the vehicle's velocities from the flight

logs to determine its displacement between frames and easily compare the data from our height estimation with the LIDAR-Lite data, which has an accuracy of 2.5cm.

4.3.2 Reliability Classification

Since we were aware of situations where our height estimation would be inaccurate, such as when there is too much rotation or the UAV's velocity is too low, we decided to create a classifier to determine when it was safe to use it. Just as with our preliminary analysis we decided to use decision trees because they are fast and can be easily implemented on embedded hardware.

4.3.2.1 Feature Vector

We defined a feature vector composed of the average and standard deviations of the optical flow vectors' magnitudes and directions. To prevent discontinuities when the direction changed from 0° to 359° we used the sine function instead of the vector's raw orientation. We also tried slicing the image in a grid pattern and binning the vectors before computing the average and standard deviations, but it was less efficient than the global average.

4.3.2.2 Dataset Acquisition

To train the decision tree we assigned classes to data according to how close they were to the ground truth. At first we tried to assign these labels automatically, by specifying a threshold for data that would be good or bad, but the results were not satisfactory, then we tried using hysteresis, defining low and high thresholds, which improved the results, but they were still not good enough. Later we tried to manually assign these labels according to our perception if our estimative was good or bad, this is what presented the best classification.

4.4 Height Regulation

The final part of our methodology is the height regulator. Since we are working with a high delay video system we decided to use an unconventional approach instead of a Proportional Integral Derivative (PID) controller that would directly control the lift of the UAV through its thrust. Our regulator works more like a proportional controller, but, alternatively, it changes the waypoint the vehicle is heading to, therefore

altering its path. This has both advantages and disadvantages over the traditional PID approach which will be discussed further in this section.

Our UAV captures digital video through the GoPro Hero 3+ camera, which features an analogue video out connection attached to an analogue video transmitter. The data is then received on a ground station computer by the means of an analogue receiver attached to an USB video card that converts the signal back to digital compressed video. All these steps of capturing, converting, transmitting, receiving and converting the signal back to digital, add delay to the system. First, we had to determine the delay before designing our regulator.

We used a simple but effective way to measure the delay. We displayed the received video on a computer screen alongside a milliseconds chronometer and pointed the camera towards the screen. Therefore, on the received video we registered both the image from the chronometer and the one displayed on the received video. Finally, by opening the received video and pausing it we could compare the time difference between the two clocks and determine the delay, which we found to be around 230ms. A similar setup is shown in Figure 4.7.

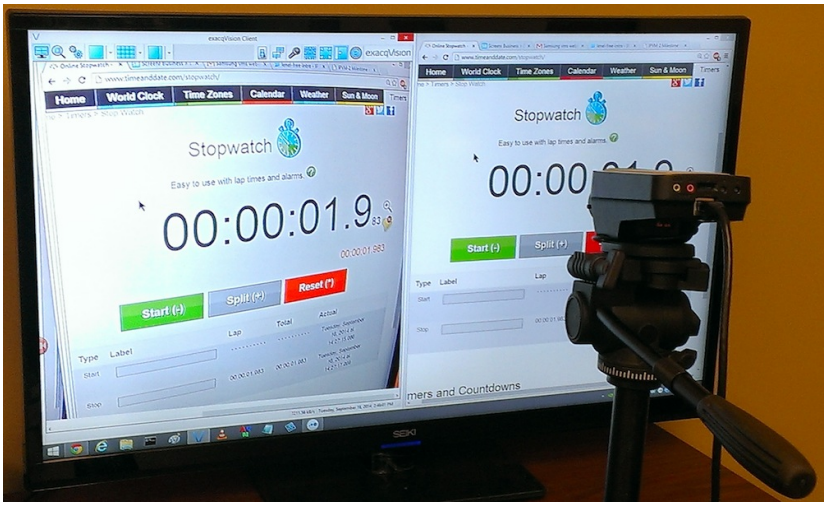


Figure 4.7: Similar camera latency test setup.

Source: `http://ipvm.com/report/testing_ip_camera_latency`

After determining the delay and considering that some communication packages with the UAV are lost, we decided to use a regulation technique more robust to the delay and failures. Therefore, instead of manipulating the throttle, an approach which is more suitable for on-board processing, we control the vehicle by changing its next waypoint, and consequently, its path.

The flight controller always tries to keep the UAV at the line formed by its

previous waypoint and the next one, traveling at a predefined speed along this line until it reaches its destination. This is to prevent wind from deviating the vehicle from the desired path. The next waypoint can even be changed or manipulated while in flight and the vehicle updates its path. We took advantage of this characteristic and determined that the best solution was to use intermediate waypoints between a starting point and a desired goal for the vehicle.

To make the regulator robust to package loss and delay, we set each waypoint five seconds away according to the predefined speed, given in meters per second. Therefore, it would take considerable time without any command packages reaching the vehicle to stop the regulator. Furthermore, in case of failure the vehicle would simply stay at the last waypoint it received. We did not use a greater distance since it would not be safe considering the waypoint would be too far away, neither a shorter distance, because it could make the vehicle stop, preventing the height estimation algorithm to work, as it needs motion. This also contributes to make the height changes smoother.

For this approach to work we also needed to determine how fast we wanted to perform the corrections: too fast could mean the vehicle would need to reduce its horizontal speed as it diverts the thrust to accelerate on the vertical axis and too slow could result in large errors, and, in the worst case scenario, collisions. In simulations we found a value of one second appropriate, that is: the computed error should be corrected within one second, but the error computation runs at a higher rate 30–60 Hz depending on the video frame rate. This happens because the vehicle does not need to reach a given waypoint to proceed to another. Instead, we are constantly updating the current waypoint the UAV is headed to.

Summarizing, our height regulation algorithm determines the next waypoint height by computing the height error and projecting a vector from the UAV to a point that would eliminate the error —if the terrain is flat— in the displacement the vehicle performs in one second. This can be better visualized in Figure 4.8. Our waypoint is defined as follows:

$$\mathsf{WP} = \mathsf{UAV} + 5(\vec{v} - \vec{e}), \tag{4.10}$$

where $\vec{p} = 5(\vec{v} - \vec{e})$.

The main issue with this approach is that when the terrain is not flat it leads to a steady state error. This happens because we assume the terrain is flat since what is further ahead is unknown. This could be changed if we assume the slope will remain the same, but we did not address this situation in the present work. The steady state error happens when:

$$\frac{|\vec{e}|}{|\vec{v}|} = \dot{g}(x(\mathsf{UAV})), \tag{4.11}$$

where $\dot{g}$ is the derivative of the function describing the terrain, or, in other words, the slope of the terrain.

We performed a simple simulation where we fitted a 4th degree polynomial to points we added on a plane, which represents a more realistic terrain. The result is presented in Figure 4.9, where the ground is represented by the fitted function f. To perform the simulation we used the following recursive function:

$$h_n(x) = h_{n-1}(x) + \Delta s \left(\frac{t - h_{n-1}(x)}{|\vec{v}|} - \dot{f}(x(\mathsf{UAV})) \right). \tag{4.12}$$

Which basically means that the height at a given instant will be equal to the previous height plus the vertical component of the UAV displacement along its path minus the change of elevation on the terrain defined by the function f on a given horizontal displacement defined by Δs. This also shows why Eq. 4.11 holds true, as

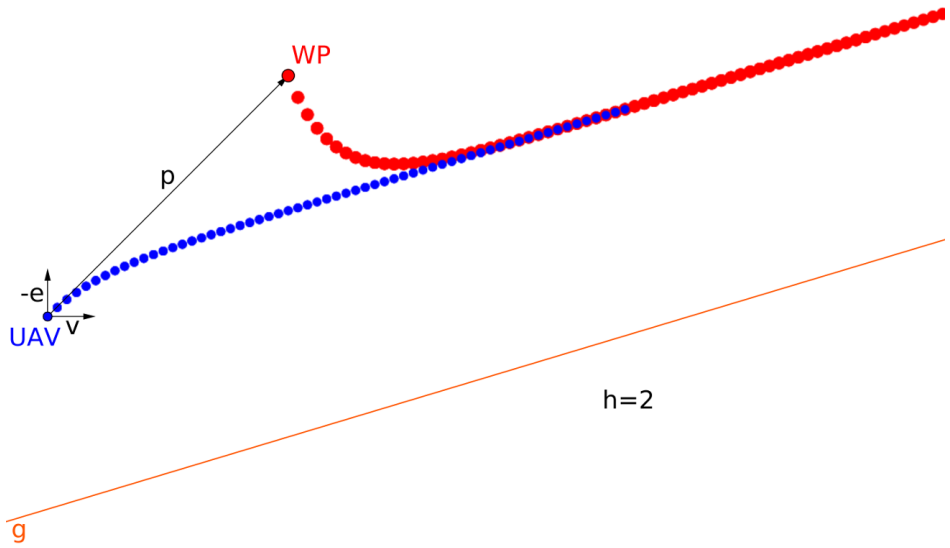


Figure 4.8: Height regulator schematic. It represents the UAV by the blue dot and the waypoint by the red dot, respectively at the beginning and end of vector $\overrightarrow{p}$, which represents the path the vehicle would take until the next waypoint is received, vector -$\overrightarrow{e}$ represents the error and vector $\overrightarrow{v}$ the predefined horizontal velocity of the vehicle, the function g is the ground plane. The trails of colored dots represent where the waypoint and UAV would be on the next iterations of the algorithm.

we can clearly see that the error, which is defined by $t - h_{n-1}(x)$, when divided by $|\vec{v}|$ cancels with the slope of the terrain at $x(\mathsf{UAV})$ —defined by the derivative of f— when both are equal, thus making the height constant even though it did not reach the target height defined by t.

The simulation shows the height regulator is viable. However, the disadvantages are that it might come to a steady state error over non-flat terrain and induce a bigger delay since the computation is made off-board. Ideally the computation would be made on-board and the controller would be a PID one, which would reduce the errors and delay, but it is not feasible over the high delay video system in a remote computer where command packages might get lost. Unfortunately we did not have a single board computer capable of performing the necessary computations on-board due to their demanding needs.

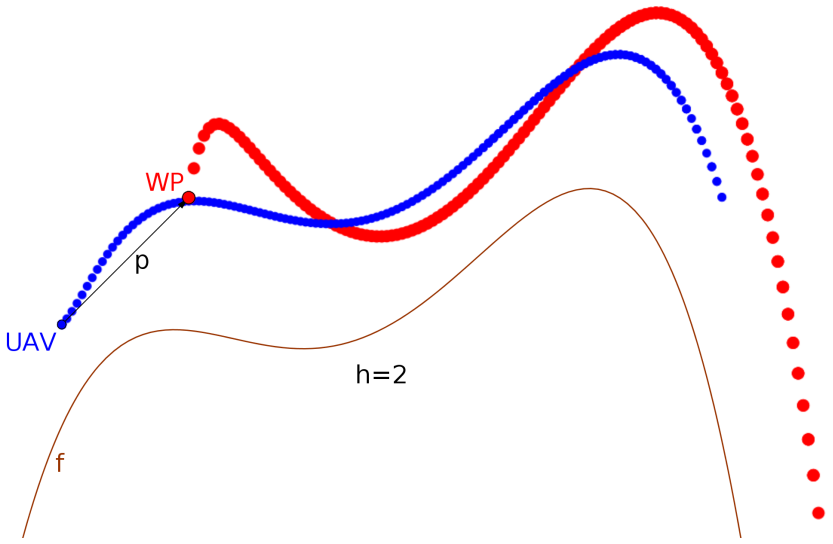


Figure 4.9: Height regulator simulation. The UAV is represented by the blue dot, and the waypoint by the red one. The vehicle goes to the waypoint through the shortest path p. Therefore following the terrain f. (This figure can be animated on Adobe Reader on desktop computers.)

Chapter 5

Experiments and Results

In this chapter we outline our experiments and discuss the results. The experiments consist of several flights in different areas that comprise diverse terrain and relief types. Most of our tests were performed at UFMG's Pampulha campus, some were performed at a farm at Carmópolis de Minas, a small town 128 km away from Belo Horizonte, and one at a deactivated mine owned by Vale called Capanema.

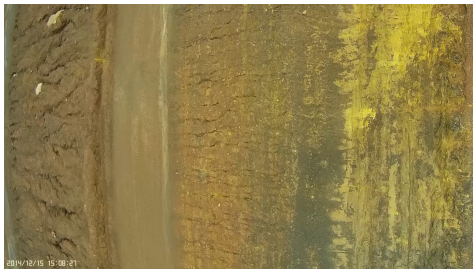
The remainder of this chapter is organized as follows: Section 5.1 describes the experiments for our preliminary investigation of optical flow in UAVs, Section 5.2 specifies the steps we took to simulate our height estimation technique, Section 5.3 reports some height estimation experiments we performed at plane terrain, Section 5.4 details experiments we performed at a Mountain at the aforementioned farm, Section 5.5 depicts our final experiments with our ground truth sensor installed on our vehicle, Section 5.6 contains a quantitative analysis of our experiments and, finally, Section 5.7 shows the simulation we used to validate our height regulation methodology.

5.1 Optical Flow Evaluation

Our first experiment aimed at gathering video data from a camera carried by the UAV. We used optical flow in order to convert the images into meaningful information, which was organized in feature vectors and classified through decision trees.

During this experiment we used the Mobius Lens B, which is an action camera with a similar performance to the GoPro Hero 3+ White. We set it up with fixed white balance for daylight scenes and to capture video at 60 FPS with 1280×720 resolution using the narrowest field of view. We performed fixed ASL altitude flights in manual and autonomous modes over various terrains at a configured speed of 3.5m/s and with little or no wind, some samples are displayed in Figure 5.1.

(a) Flat terrain



(b) Mining site

(c) Buildings

(d) Trees

Figure 5.1: Snapshots of videos composing our dataset: (a) picture of a plaza, it represents an area of flat terrain, (b) picture of a mining site, it shows an area of inclined terrain, (c) picture over buildings and (d) over trees, these represent areas of other terrain.

To manually annotate the data, we walked over the areas we flew and registered the coordinates and ASL altitude in several points. This allowed us to have an accurate representation of the inclination and elevation profiles rather than if we had just classified the videos according to our perception. Our dataset ended up organized as described in Tables 5.1 and 5.2.

Table 5.1: Dataset composition for terrain type and elevation.

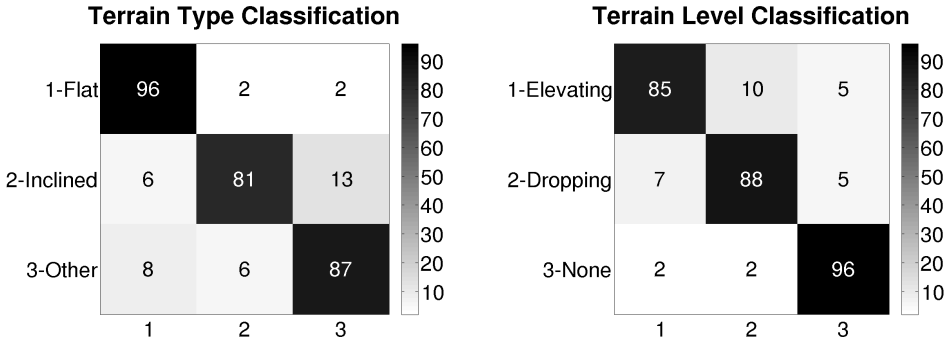
Tree	Class	Number of Images
1	Flat	264
	Inclined	324
	Other	360
2	Elevating	192
	Dropping	264
	None	492

Table 5.2: Dataset composition for terrain inclination direction.

Class	None	0º	45º	90º	135º	270º	315º
Number of Images	264	36	108	156	72	252	60

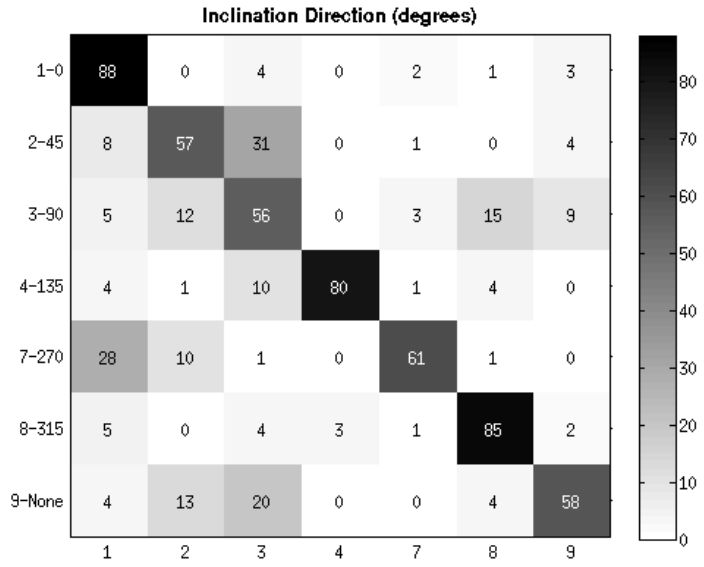
To classify terrain information (regarding type, inclination direction and elevation), we trained three ID3 decision trees. The first one was meant to classify whether the terrain was flat, inclined or other, such as over buildings or trees. The second was to determine the direction of inclination. The third one was to classify if the terrain was going up, down or neither of those. The trees were trained using 10-fold cross-validation and 80% of the data was used as the training set, the remaining 20% composing the test set. The confusion matrices are shown in Figure 5.2.

In the first scenario we achieved an accuracy of 86.75% with the combination of the octant and history descriptors, with a higher influence from the octant descriptor,



(a) Terrain type

(b) Terrain level



(c) Terrain inclination direction

Figure 5.2: Confusion Matrices. The figure shows the confusion matrices for our three decision trees. The predicted is along the lines while the ground truth is along the columns. In (c) we omitted two classes (180º and 225º) in which we did not have any data.

which is expected, since it was designed to detect inclination. We also observed that the error is higher between inclined and other terrain types. This happens because sometimes an area over buildings or trees could be easily confused with inclined terrain.

In the second scenario we achieved an accuracy of 77.34% with the combination of all descriptors, however, by crossing data from Table 5.2 and Figure 5.2c, we conclude that the low accuracy (56–61%) in some classes is a consequence of few data in them. In addition, we observed that the error is higher between 45° and 90°, which are neighboring classes and since we are working with discretized data, this could be explained by entries that lie close to their borders. In this tree all descriptors contribute to classification, with the history one contributing the least, but still providing valuable data for inclination directions closer to 0° and 180°.

In the last scenario we achieved an accuracy of 91.85% with the combination of the grid and history descriptors, where most of the influence comes from the latter. This is a consequence of the elevation being evaluated as a function of time, and as seen in Chapter 2, the optical flow is able to provide height information, thus resulting in elevation information if evaluated in subsequent frames. The error is higher in elevating and dropping classes. We attributed this to a delay induced by the history vector, which would be responsible for misclassified frames just after changes in the terrain elevation.

5.2 Height Estimation Simulations

To set most of the input parameters in the experiments and evaluate the proposed methodology at a controlled environment, we first performed several tests using a realistic simulation scenario. For this, we use FlightGear[1], as it is highly integrated with ground elevation databases, as well as the very same GCS (Mission Planner[2]) used in our field experiments.

The FlightGear Flight Simulator presents the advantage of using 3D models, satellite imagery and custom textures to render a scenario based on a mesh created from the elevation databases. Furthermore, it is open source, which allows changes in several parameters. For example, it is possible to set the camera positioning, simulating the behavior of biaxial or triaxial gimbals. A sample picture is shown in Figure 5.3.

Figure 5.3: Screenshot of a simulation at the Innsbruck airport using the FlightGear Flight Simulator.

We combined FlightGear with the ArduCopter[3] SITL (Software In The Loop) simulator, which was responsible for computing the vehicle motion and control, sending these data to the simulator for rendering and also logging data in the same format our

[1] http://www.flightgear.org/
[2] http://ardupilot.org/planner/docs/mission-planner-overview.html
[3] http://ardupilot.org/copter/index.html

vehicle would log in the real world. In fact, it is the same software that runs on our UAV.

We performed the simulations at the Innsbruck airport, which, according to the FlightGear wiki, is one of the places with the best scenery data available. Moreover, it also presents the necessary slopes to properly investigate the feasibility of our height estimation approach for terrain following flights. We performed two flights, one at a fixed ASL altitude of 500 m above the runway and the other at a fixed AGL altitude of 200 m.

5.2.1 Fixed ASL Altitude at 500 m Above Runway

Figure 5.4 shows the results of the height estimation for this simulation. The x axis indicates the frame in the video file; the left y axis shows the AGL altitude of the vehicle; and at the secondary y axis, the relative estimation error of the current frame is presented.

It is possible to observe that, in most of the frames, the estimations are close to the actual AGL altitude of the vehicle, but when the terrain starts to rise, reducing the effective height, we observe a delay for the estimations to reach the actual height of the vehicle. The reason for this behavior is because at 500 m, the field of view comprises so much area that many optical flow vectors are accumulated at the part of the frame opposite the vehicle movement. Indeed, this is confirmed by the fact that the delay decreases proportionally with the AGL altitude. However, when the AGL altitude starts to rise again, we see two sudden spikes at the estimations. This is due to the fact that the simulations rely on synthetic textures, and the keypoint detector might identify more representative corners in some textures than others, causing sudden changes in the tracked features when they go out of view.

To allow the classifier to determine if the estimations were reliable, we trained it for such heights. However, we did not mix data from the simulations at the training of the field experiments' decision tree, which will be discussed in the next sections. There are two points, close to Frames 103,435 and 102,775, that present some spikes in the height estimation values. These are points where the vehicle stops and changes direction and are classified as not reliable by our decision tree (the frames in green on the graph are classified as reliable). This behavior is also observed elsewhere in our experiments.

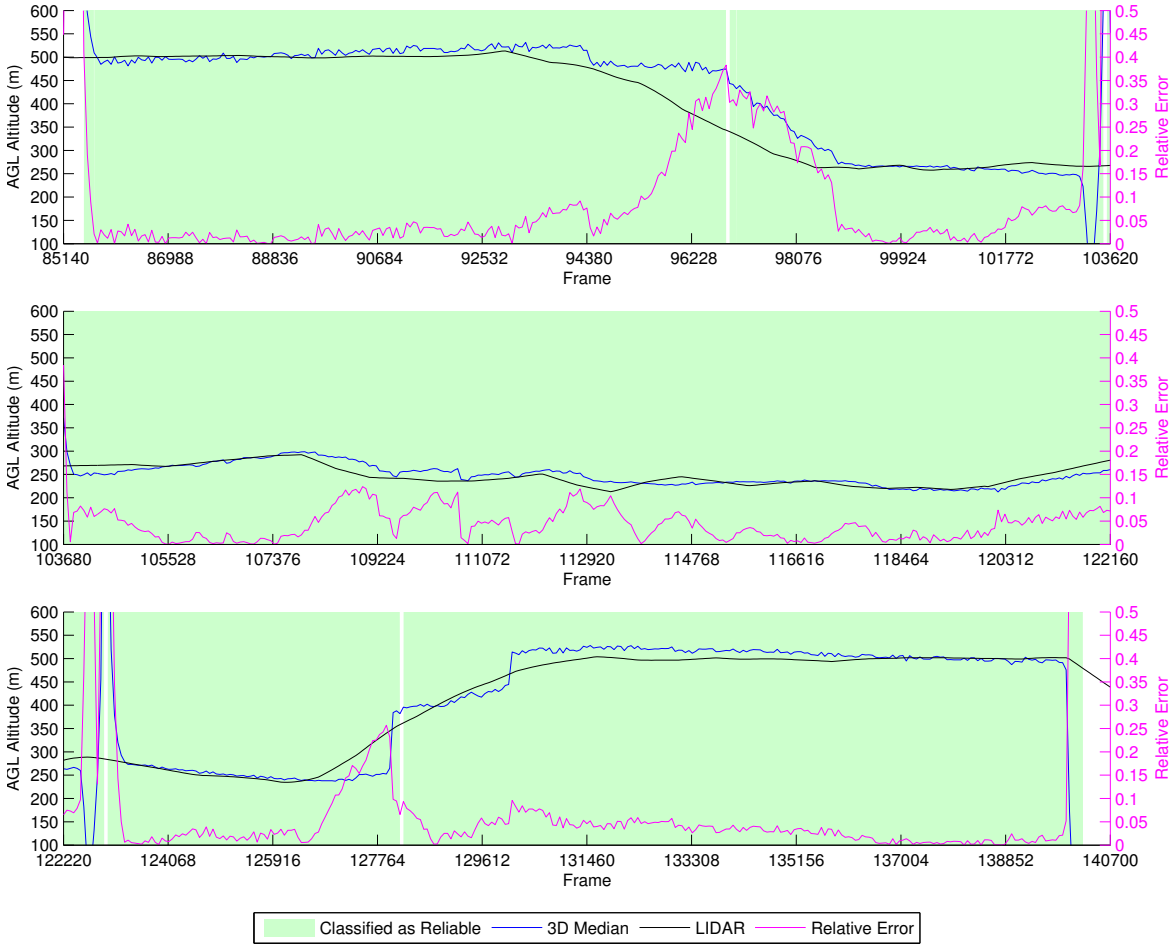


Figure 5.4: Height estimation results for the fixed ASL altitude at 500 m above the runway simulation.

5.2.2 Fixed AGL Altitude at 200 m

Figure 5.5 depicts the simulation results for the fixed AGL altitude at 200 m experiment. One can clearly observe that the data have more jitter in this simulation, which implies that the tracked points are being exchanged more frequently, a consequence of flying lower than the previous simulation. However, the great majority of estimations are still within 10% of the actual height. Around Frames 19,296–21,750, the vehicle flies over a river, which presents reflections, therefore compromising the estimations. This effect was minimized in the previous simulation because the area being captured by the camera was larger. We also observe two spikes when the vehicle stops and changes directions, at Frames 28,195 and 48,430, a characteristic also shown in the previous simulation at the same locations.

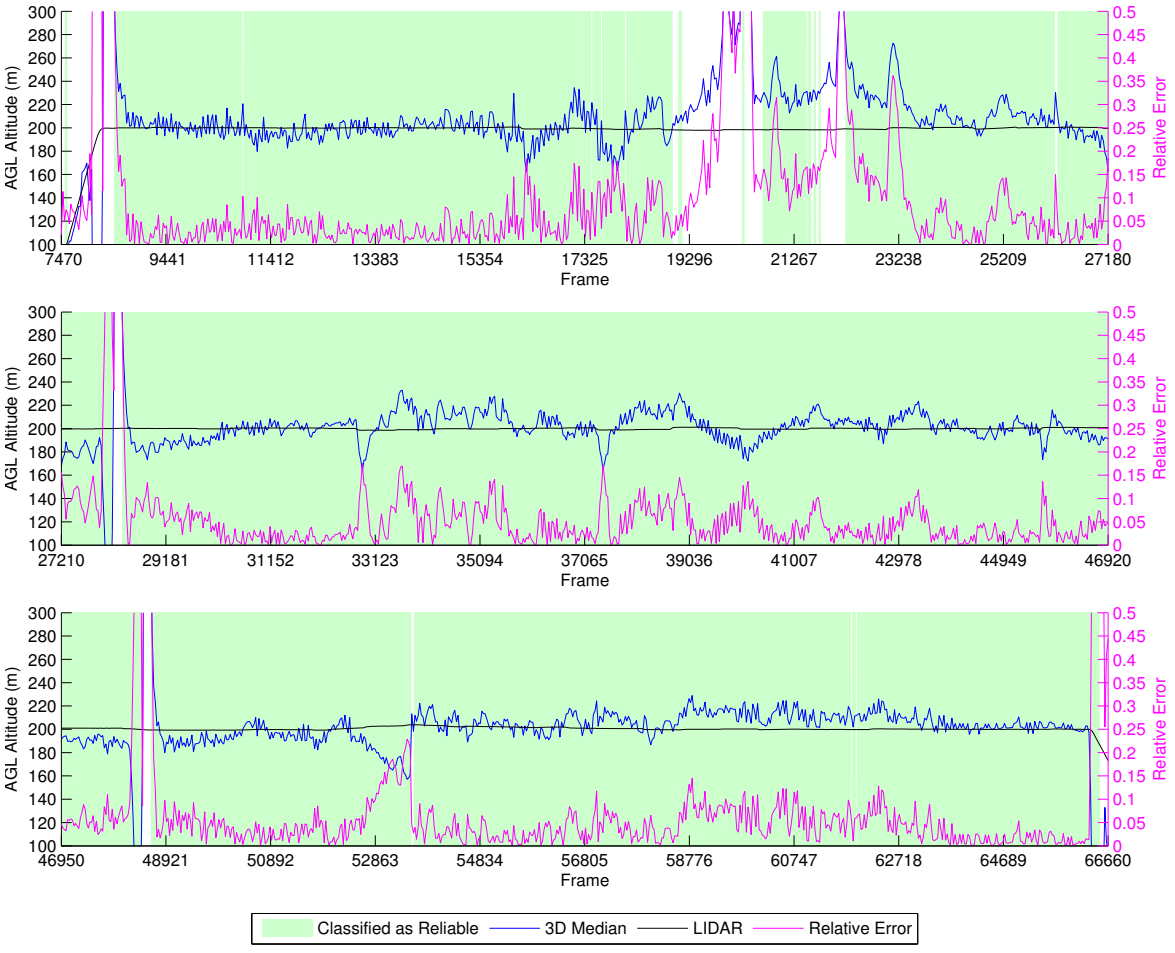


Figure 5.5: Height estimation results for the fixed AGL altitude at 200 m simulation.

5.3 Plane Terrain

To evaluate height estimation capabilities using our methodology based on optical flow data we performed the second set of experiments on a soccer field at UFMG. As it is a plane terrain the ASL altitude relative to the takeoff point measured by the GPS and barometer would be equal to the AGL altitude we wanted to estimate, which could be used as a ground truth. We defined a flight plan consisting of a rectangle with altitudes varying from 2–16 m, increasing at 2 m intervals at a predefined target speed of 5 m/s. In this stage we were still using the Mobius camera with the narrowest field of view and fixed white balance for daylight at 1280x720 resolution and 60 FPS. Figure 5.6 shows the resulting flight path from GPS data of the UAV's telemetry logs.

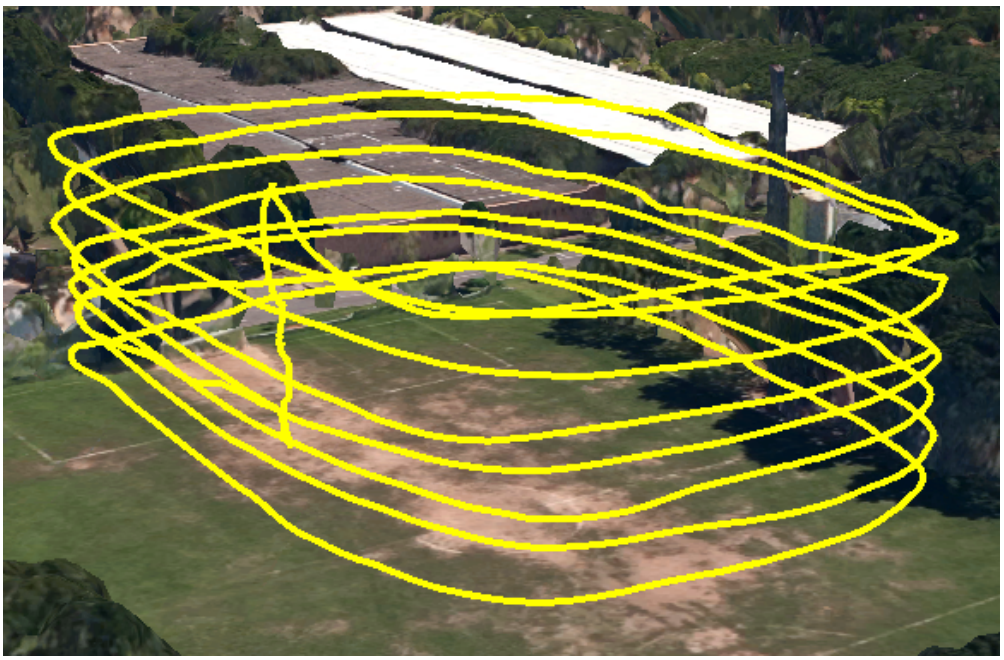


Figure 5.6: Soccer field flight path.

Since this was the initial phase of our height estimation experiments it was performed under certain constraints, those being the lack of a ground truth sensor and motion information from the telemetry logs. This last is because our vehicle was using an older firmware version that did not record timestamps on the motion information available at the telemetry logs, therefore making it difficult to properly synchronize the video and motion data as we use the timestamps to interpolate velocities and compute the camera displacement between frames. To tackle this lack of information we set our algorithm to consider the camera motion to be a constant value between the frames according to the configured aircraft speed in the mission plan.

Figure 5.7 shows the resulting estimations. The graphs show the AGL altitude estimation on the left y axis and the camera frame number in the video file at the x axis, whereas the green background shows where the decision tree —described in subsection 4.3.2— classified the estimation as reliable. The next graphs also include a secondary y axis that shows the vehicle's yaw variation between camera frames.

There are three algorithms that we experimented with. The first is a traditional fronto-parallel stereo approach, referred as 2D mean, since it uses the arithmetic mean of the estimated heights at each tracked feature point as the output value. The second and third are the modified algorithm, accounting for the displacements in the z axis, referred as 3D mean, where we use the mean and 3D median, where we use the median to filter large outliers disturbing our estimations. Some experiments also include LIDAR data that we used to evaluate the estimation quality.

In the soccer field experiment the vehicle takes off and shortly after, between frames 824–2060 it flies over an area with just sand, therefore the algorithm fails to track features and consequently wrongly estimates the heights. After that the experiment behaves normally, but we noticed that only low height estimates were classified as good. This is a consequence of our decision tree being trained exclusively with experiments with LIDAR data, where flights were performed at low AGL altitudes due to sensor restrictions (less than 5 m).

The estimations are also unstable at this experiment since the vehicle was programmed to perform a rectangular pattern at a fixed speed and we lacked motion information, but in reality the UAV actually accelerated and decelerated close to the corners of the rectangle, resulting in the peaks and valleys shown in the graphs. When the vehicle is closer to the predefined speed, it is a valley, when it decelerates by approaching a corner, it creates a peak. Despite the instability, we can still see that the readings are close to the predefined heights at the valleys, that is, the predefined speed used in our estimation algorithm. For better visualization we injected the commanded height in the graph.

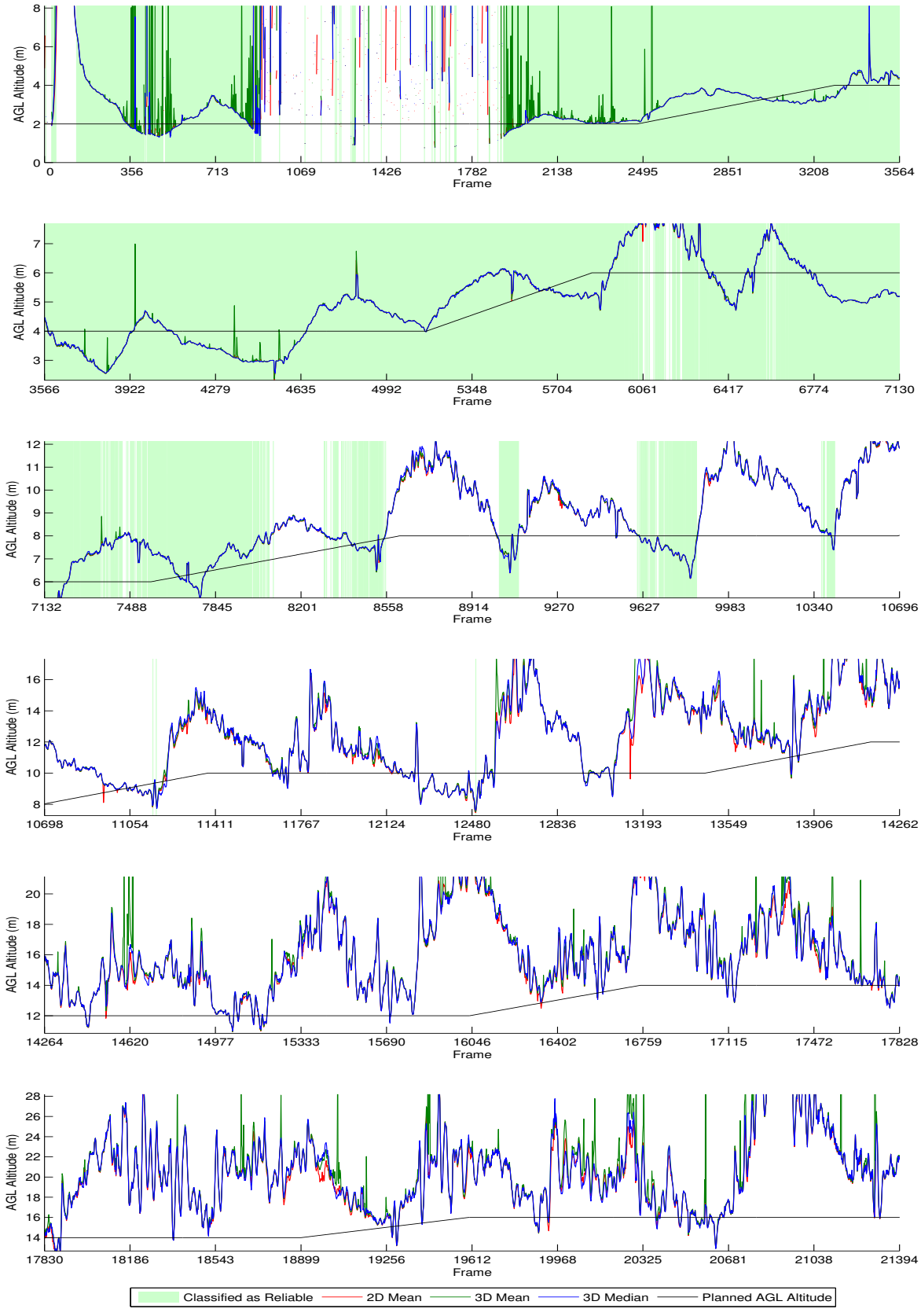


Figure 5.7: Height estimation results for the soccer field experiment.

5.4 Mountain

To test the height estimation algorithm with sloped terrain we performed the next experiment on a mountain. We did not have the laser rangefinder yet, therefore, to establish a ground truth we logged the projection on the terrain of the desired flight path through a mobile phone GPS application and adjusted the waypoints accordingly to keep the UAV at a fixed AGL altitude of 25 m.

The flight consists of going uphill in two different parts of the mountain, one mostly covered by grass and the other by small trees and bushes. This time the vehicle already featured the GoPro camera and it was configured to record 1920 × 1080 video at 60 FPS with automatic white balance and narrow FOV. Figures 5.8–5.11 show flight plan and telemetry data of the experiment.

Figure 5.8: Mountain flight plan. It shows the flight plan for the mountain video sequence, the first slope is between waypoints 1 and 16, and the second between waypoints 19 and 25.

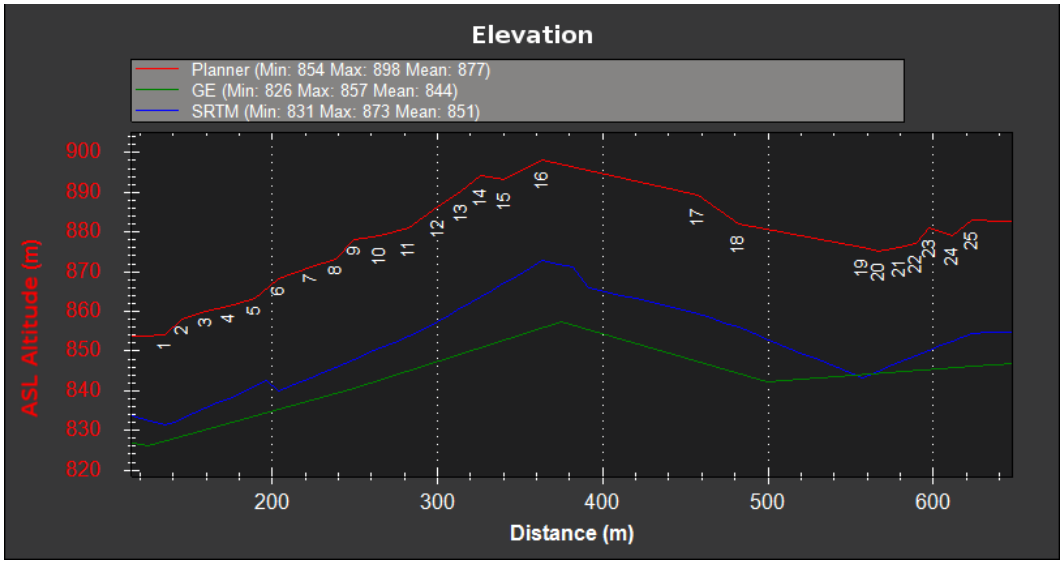


Figure 5.9: Mountain flight elevation. It shows the elevation at the specified coordinates provided by the flight plan (25m above ground), Google Earth (GE) and the Shuttle RADAR Topography Mission (SRTM) database, the shape of the last one closer resembling our flight plan.

Figure 5.10: Mountain flight path from GPS data of the telemetry logs. This image is from a perspective view, where the red lines represent the flight paths of the two areas of interest and the green shade the projection of this flight path on earth's surface.

(a) First part.

(b) Second part.

Figure 5.11: Mountain flight path elevation from GPS data of the telemetry logs. The horizontal axis shows the distance the UAV has traveled and the vertical axis represents the ASL altitude.

This page is intentionally left blank for better visualization
of the following data on the printed version of this thesis.

Figure 5.12 shows the results of our experiment at the mountain. This time the UAV had its firmware upgraded and the logs properly store timestamped motion data, allowing us to normally execute our algorithm, but it still lacked the laser rangefinder.

The beginning of the experiment shows the vehicle flying over a small tree around frames 51300–51779, then it flies uphill with some yaw interference up to frame 53379, after that it flies straight to the next waypoint, at frame 53857. Despite the yaw interference we see the vehicle flying close to the configured height of 25 m. Afterwards, the UAV rotates to face the next waypoint at frame 56500, in the meantime we see the AGL altitude increasing as this part of the flight actually consists of going at a fixed ASL altitude and the terrain altitude is decreasing. This behavior changes only around frames 55457–55617 as the vehicle flies over a tree. Upon reaching the next uphill sequence, the aircraft rotates to face the following waypoints, at frame 58900, this time the altitude estimates oscillate more because of the bushes and trees below, as the terrain consist of an orange trees field. After that the UAV flies at a fixed ASL altitude to the next waypoint and this is again shown by the height variation. Around frames 60417–60600 the vehicle flies closely over a tree and we observe the classifier identifying it as a reliable estimation.

We see the three algorithms close when flying at fixed ASL altitudes, but the 3D techniques differ from the 2D when flying following the terrain, specially over the orange trees. The classification as reliable estimation at low altitudes over the tree also confirms our previous suspicion that the issue is with lack of training data at higher altitudes. Another subtle information is that the UAV yaws more when it is flying uphill than when it is flying at a fixed ASL altitude. We believe this is due to the higher demand of the motors making the platform less stable, a consequence of our vehicle being a little heavy for its current power configuration. Last, it is also clear how the yaw interferes with the height estimation, as we can clearly observe the estimations are more stable when there is no yaw.

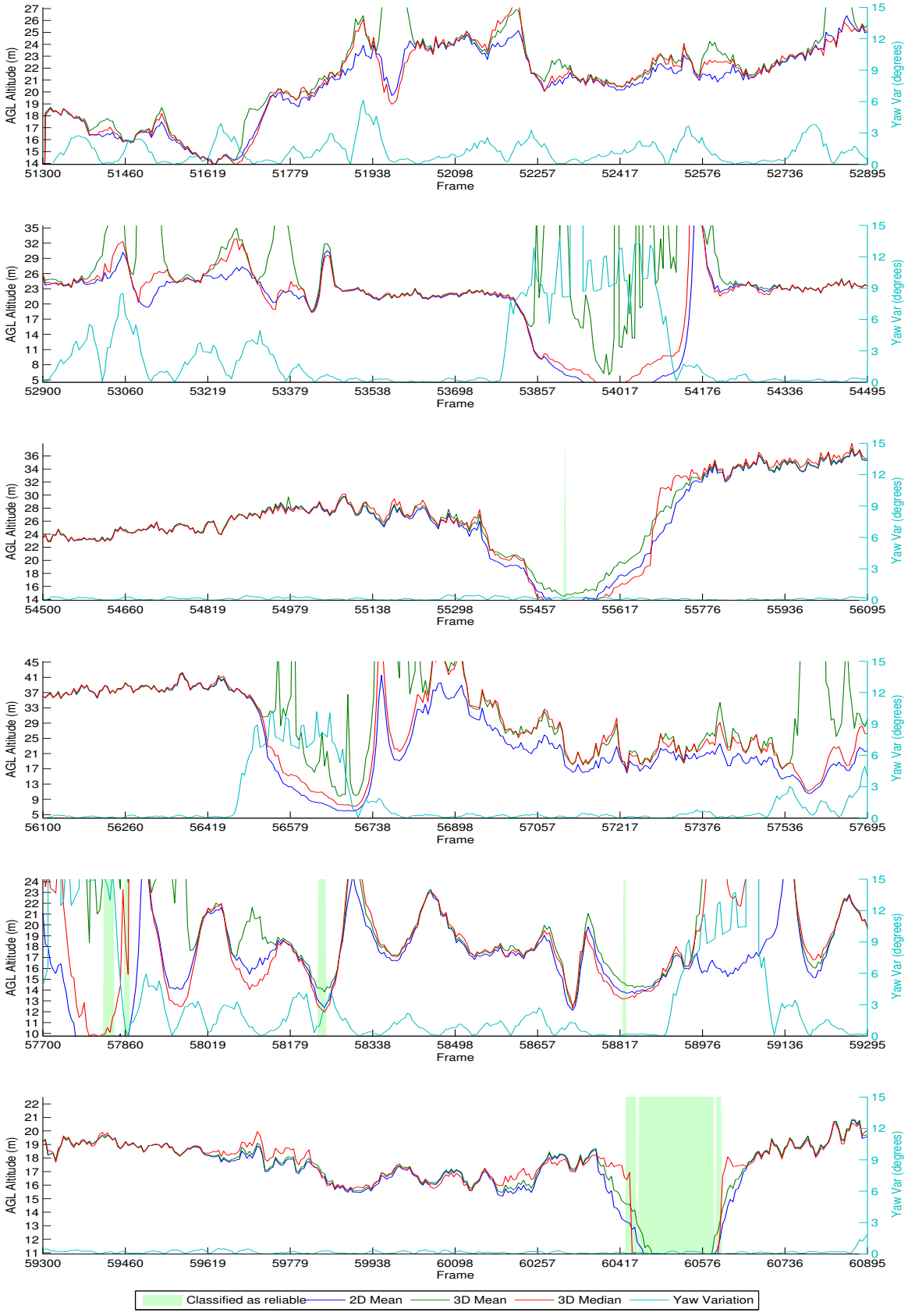


Figure 5.12: Height estimation results for the farm experiment.

5.5 LIDAR

With our complete prototype —including the laser rangefinder— we performed several flights at UFMG, most of them in front of the dean's office building, where there is a clear field with small inclination and few obstructions. These experiments were performed using a GoPro camera configured to record 1920×1080 video at 60 FPS, narrow FOV and automatic white balance. We performed different flights, autonomous (Figure 5.13) at fixed ASL and AGL altitudes and also manual flights (Figures 5.14 and 5.15) at a vertical zig-zag pattern to test the algorithm response to height changes.

Figure 5.13: Autonomous flight path at the dean's office.

Figure 5.14: Manual flight path with vertical zig-zag pattern.

Figure 5.15: Selected sequences of the manual flights for clear visualization.

This section contains the results of the experiments with the final version of our prototype, which includes the LIDAR rangefinder as a ground truth sensor. As several experiments were performed we decided to split it in subsections accordingly.

This page is intentionally left blank for better visualization
of the following data on the printed version of this thesis.

5.5.1 Fixed Above Sea Level (ASL) altitude

Figure 5.16 shows the results of our fixed ASL altitude flight in front of the dean's office building at UFMG. From the beginning up to frame 28000 the vehicle takes off, but it is unstable, after flying towards the first waypoint we observe the classifier marking the estimation as reliable and see that all the algorithms closely match the LIDAR data until the vehicle stops at frame 29570 and reverses direction, going back to the previous waypoint before performing a Return To Launch (RTL) maneuver at frame 30800.

The RTL maneuver is when the vehicle autonomously fly to the point it took off from, however, as a safety measure the vehicle first climbs to a safe height, predefined at 15 m before going to the landing position, which can be observed in the last part of the graph. Unfortunately, in this experiment the logged yaw data got corrupted after frame 29900. This experiment shows that there is a very distinct reaction in the height estimation when the vehicle is stopping and reverting direction, which can be seen between frames 29591–29757.

Since this is a low height flight the classifier correctly works in this experiment filtering bad altitude estimations with unreliable optical flow data. There are, however, a few frames of spiky estimation due to noise induced by the optical flow failing to properly track some features. This noise was minimized in the 2D Mean and 3D Median algorithms.

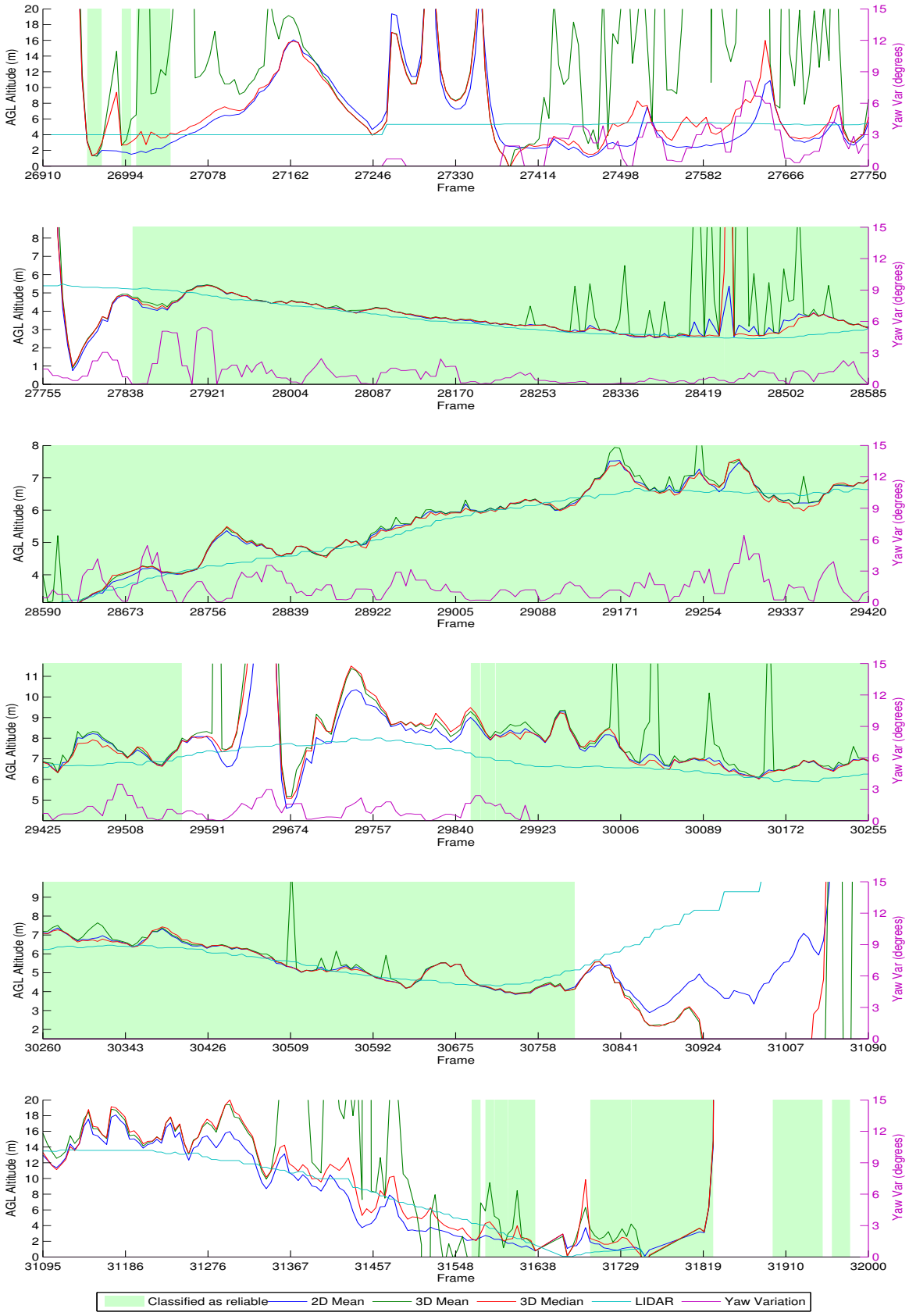


Figure 5.16: Height estimation results for the LIDAR experiment at fixed ASL altitude.

5.5.2 Fixed Above Ground Level (AGL) altitude

Figure 5.17 shows the results from our terrain following flight also in front of the dean's office building at UFMG. In this experiment we used the same approach as at the farm, we walked over the desired flight path with a GPS logging the coordinates to properly set a flight plan in which the vehicle would fly at a constant AGL altitude.

This time the flight happened in windy conditions, therefore the vehicle took of manually, and before switching to the autonomous mode at frame 52300, we just hovered with the vehicle to evaluate if it was safe to proceed with the experiment. After switching to the autonomous mode we see the classifier quickly identifies the data as reliable, but due to the wind shaking the UAV —clearly shown by the higher than usual yaw variation— the estimations were less accurate than what they commonly are.

This experiment also demonstrates that the vehicle needs to be in motion in order to properly estimate the height, as we already knew from theory.

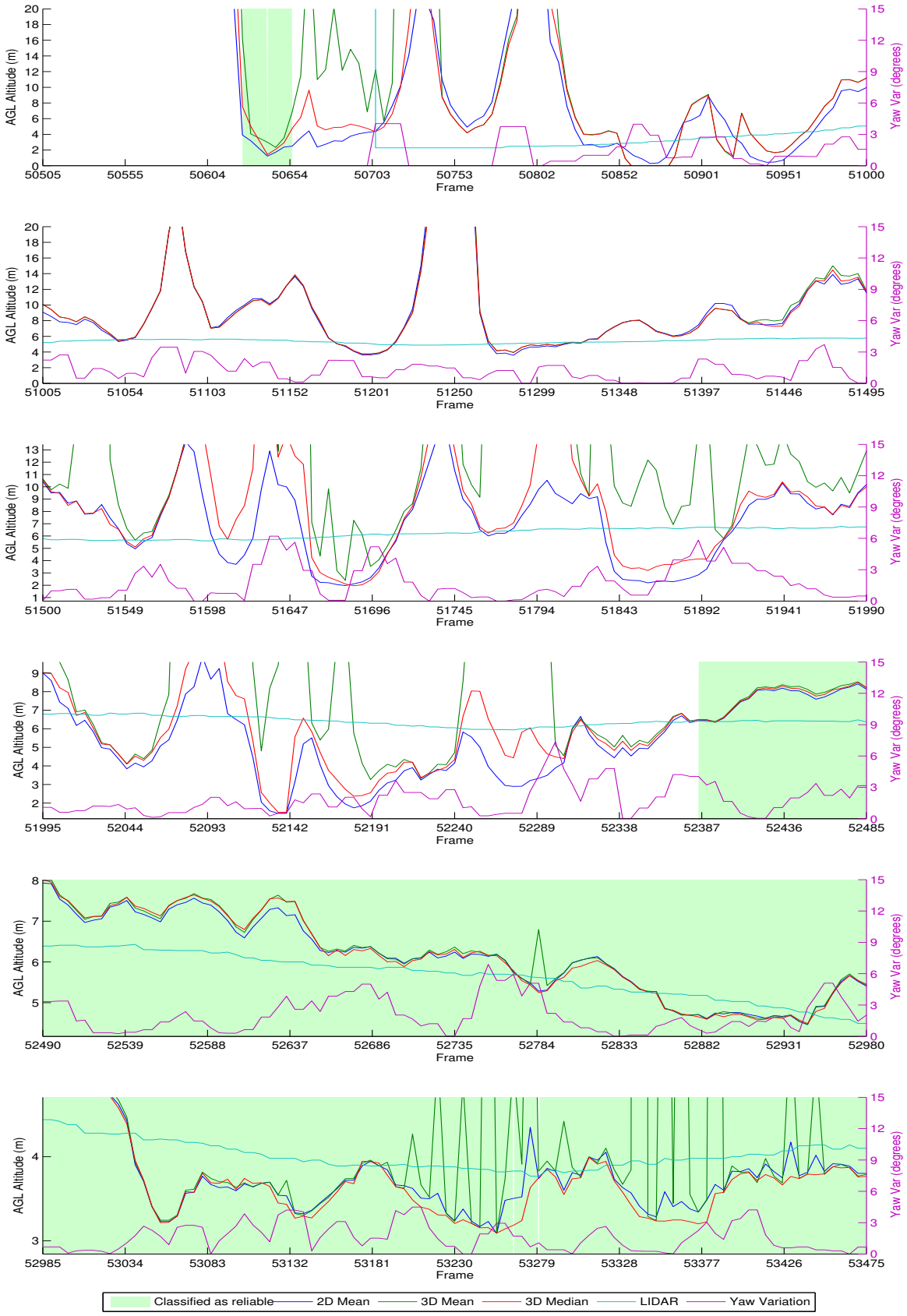


Figure 5.17: Height estimation results for the LIDAR experiment at fixed AGL altitude.

5.5.3 Autonomous vertical zig-zag

To test the capabilities of our algorithm and compare with the conventional stereo fronto-parallel one, we performed experiments varying the height intensively, at approximate angles of 45 degrees in a vertical zig-zag pattern. However, since the vehicle is currently too heavy for its power configuration, the momentum accumulated in the descent makes the flight controller overshoot the target waypoints and we interrupted the experiment with a rough landing.

Figure 5.18 shows the results of the vertical zig-zag flight in autonomous mode. As conditions were also windy, we still took of with the vehicle manually, switching to autonomous mode at approximately frame 70027. In the beginning the vehicle flights at a fixed ASL speed to the first waypoint, around frame 71409. Until there, the estimation works fine, but when the vehicle has to perform the vertical zig-zag pattern —which can be seen by the LIDAR graph— the classifier no longer trusts the estimations, which are off due to a combination of speed variation due to the overshoot and high yaw caused by the weight of the vehicle. Shortly after that the experiment was interrupted.

Despite the challenges, this experiment reveals that the vehicle is underpowered to perform flights with rough climbs in autonomous mode. We predicted the vehicle could become heavy in the design phase of our UAV prototype and chose parts that would be compatible with 11.1 V and 14.8 V batteries, the latter adding more power to the aircraft. However we did not order the 14.8 V batteries as we were still being able to perform all the experiments so far.

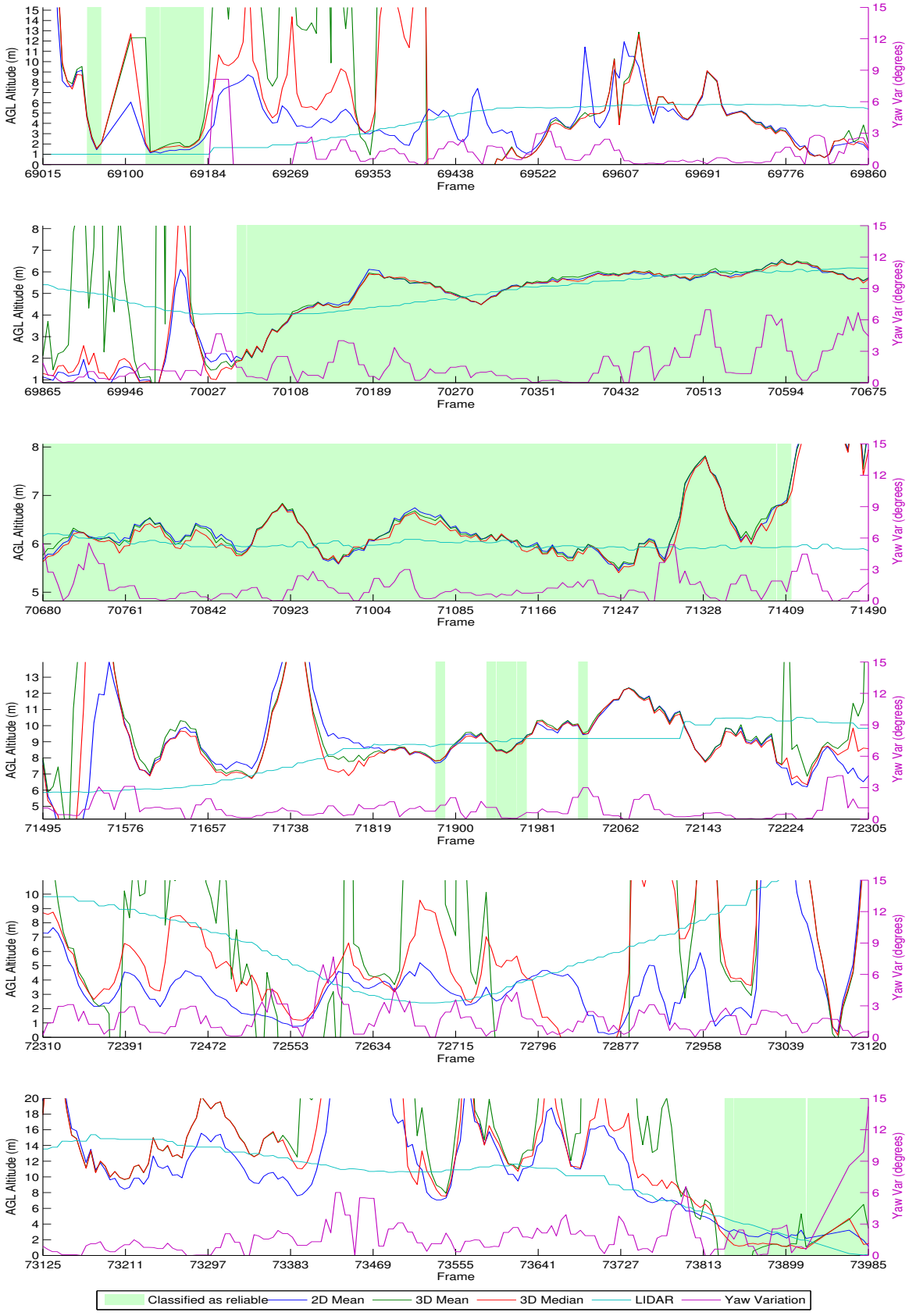


Figure 5.18: Height estimation results for the autonomous vertical zig-zag experiment.

5.5.4 Manual vertical zig-zag

As the only apparent viable solution to perform the vertical zig-zag appeared to be cutting vehicle weight —which is already at almost the bare minimum— we performed this experiment in manual mode. This way we would be able to control the vehicle and keep it at stable speeds and control the yaw.

Figure 5.19 shows the results from the manual vertical zig-zag flights. We observe that the estimation is close to the LIDAR after the vehicle starts moving (around frame 102365), and the up/down spikes we noted from the previous experiments when the vehicle stops and changes direction. Around frame 112420 the UAV crash-landed, but was ready to fly again close to frame 121000.

The height estimations in this experiment looks good and close to the LIDAR data where the classifier identified them as safe. Furthermore, the accidental crash —among others— revealed the vehicle is robust and could even be ready to fly again shortly after.

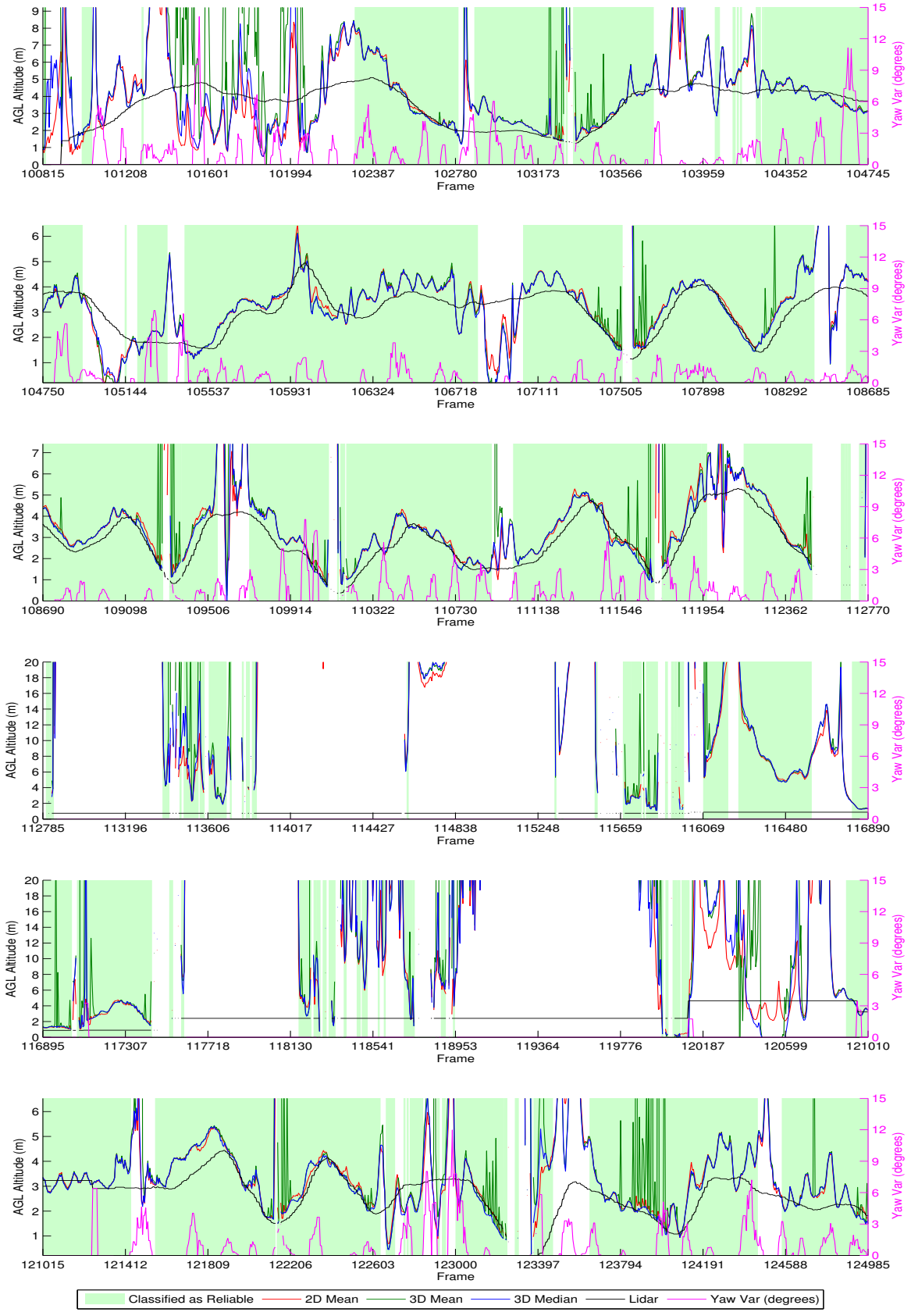


Figure 5.19: Height estimation results for the manual vertical zig-zag experiment.

5.6 Quantitative Analysis

After performing the visual analysis of the graphs, we proceeded to the quantitative analysis, in which we performed tests to evaluate the algorithm performance with the LIDAR experiments data. The results are presented in table 5.3.

To achieve these results, we used only the data classified as good by our decision tree. The tree ended up with height 2 and achieved accuracies of 80% for true positives and 90% for true negatives. According to our labels, a positive result means our estimative was reliable. We determined that the decision tree highly improves the accuracy and precision of our height estimation and is fundamental for its safe use.

First we analyzed the algorithm accuracy and precision through the average and standard deviation of the relative height estimation errors. However, it became clear that despite our efforts to filter bad estimations through our classification technique, few disturbing points remained, resulting in considerable alterations of the averages and standard deviations. Therefore, to filter this data, we limited the relative errors to up to 250%, discarding higher errors. To verify if we were discarding a considerable amount of data, we counted the used samples for each metric. The counts revealed our 3D Median algorithm is using almost as much data as the traditional 2D Mean approach, while our 3D Mean algorithm is a just a bit behind.

We observed that the results from the 3D Mean algorithm are worse than the others. This is because, as already mentioned, our methodology is more susceptible to noise from poorly tracked features. On the other hand, the 3D Median lead to better accuracies —indicated by the averages— than the traditional 2D Mean algorithm for every experiment that had vertical motion —all, except the Fixed ASL altitude—, achieving the mark of 2.4% for autonomous flights, that produce a more constant speed, while the 2D mean achieved 4.2%.

Regarding precision, none of the algorithms excelled. The traditional 2D Mean produced better results due to its greater noise robustness. Considering a normal distribution, the 3D Median algorithm would achieve 68% of the estimations within 24% from the correct value, and 95% within 48% from it, while the traditional 2D Mean would achieve 17.8% and 35.6% respectively. This, however, could be improved by installing an IMU directly attached to the camera and using the motion data to derotate the optical flow and enhance estimations.

Results from the manual flights are worse than the others due to large variations in speed and motion of the vehicle. The more constant the motion of the vehicle is, the more stable the estimations are. A quadrotor platform is great for its ease of use, but greatly unstable, being less than ideal for this application. Planes, on the other

hand, would perform better, but were impractical for our experimental needs.

We also computed common error metrics, the Mean Squared Error (MSE) and the Mean Absolute Error (MAE), this time without the additional filtering limiting the relative errors to 2.5. Both, show again the 3D Mean producing worse results and the 2D Mean and 3D Median behaving similarly, with a slight advantage to the former.

Table 5.3: Quantitative results of the classifier filtered data. The first row represents the experiments, the first column the metrics used, the second row contains the average height —in meters, computed exclusively with the LIDAR data— of each experiment, the second column contains the evaluated algorithms. In addition to the classifier filter, for the averages and standard deviations we filtered the errors up to 250% and represented the results in relative numbers. The MSE and MAE are based on absolute numbers, filtered only by the classifier.

	Experiment	Fixed ASL altitude	Fixed AGL altitude	Auto vertical zig-zag	Manual vertical zig-zag
Metric	Average Height (m) / Algorithm	4.800	4.972	5.323	2.648
Average	2D Mean	0.051	0.009	0.042	0.199
	3D Mean	0.135	0.060	0.053	0.200
	3D Median	0.070	0.005	0.024	0.171
Standard Deviation	2D Mean	0.169	0.171	0.178	0.384
	3D Mean	0.343	0.284	0.268	0.389
	3D Median	0.226	0.180	0.240	0.366
Count	2D Mean	555	224	304	2460
	3D Mean	541	222	300	2400
	3D Median	554	224	304	2447
MSE	2D Mean	0.370	0.482	0.408	4.148
	3D Mean	13.469	23.144	1.862	70.519
	3D Median	1.507	0.567	0.646	5.398
MAE	2D Mean	0.388	0.552	0.440	0.964
	3D Mean	0.984	1.087	0.629	1.578
	3D Median	0.395	0.601	0.495	1.010

5.7 Height Regulation Simulation

Our simulation consists of a naïve implementation on a dynamic geometry software, Geogebra[4], to help us visualize the behavior of the aircraft from a 2D perspective. We added noise to the height estimation using a normal distribution with 10% standard deviation. The results are presented in Figure 5.20. As we can observe the generated waypoints do not come in contact with the terrain, making it safe for the vehicle to stop and hover in case it loses connection with the ground station. Additional safety measures could be added, like returning to launch at a safe ASL altitude through the same path it flew, in case it failed to receive commands from the ground station for a predefined time limit.

The simulations consist of modifying Equation 4.9 to add the desired noise (resulting in Equation 5.1), in this case, a variable consisting of a random number following a normal distribution within 0 mean and standard deviation corresponding to 10% of the actual vehicle height.

$$h_n(x) = h_{n-1}(x) + \Delta s \left(\frac{t - h_{n-1}(x)}{|\vec{v}|} - \dot{f}(x(\mathsf{UAV})) + \mathcal{N}(0, 0.1 \times h_{n-1}(x)) \right), \qquad (5.1)$$

where the terrain is defined by the function f, a given horizontal displacement is defined by Δs. t stands for the target height and $|\vec{v}|$ the predefined velocity of the vehicle, and the normal distribution $\mathcal{N}(0, 0.1 \times h_{n-1}(x))$, which is responsible for adding Gaussian noise to the height estimation.

Even after repeating the simulation a thousand times (Figure 5.21), waypoints only presented below the terrain a handful of times, but since they are constantly updated by the height regulator, the vehicle never comes close to the terrain. This effect could be reduced by reducing the waypoint distance from the vehicle. However, this was also an extreme scenario with the greatest inclination over 40°. Furthermore, any UAV mission needs to be properly supervised by a qualified operator, which, in case of any danger can manually override the controls of the vehicle and land it safely.

[4] `http://www.geogebra.org/`

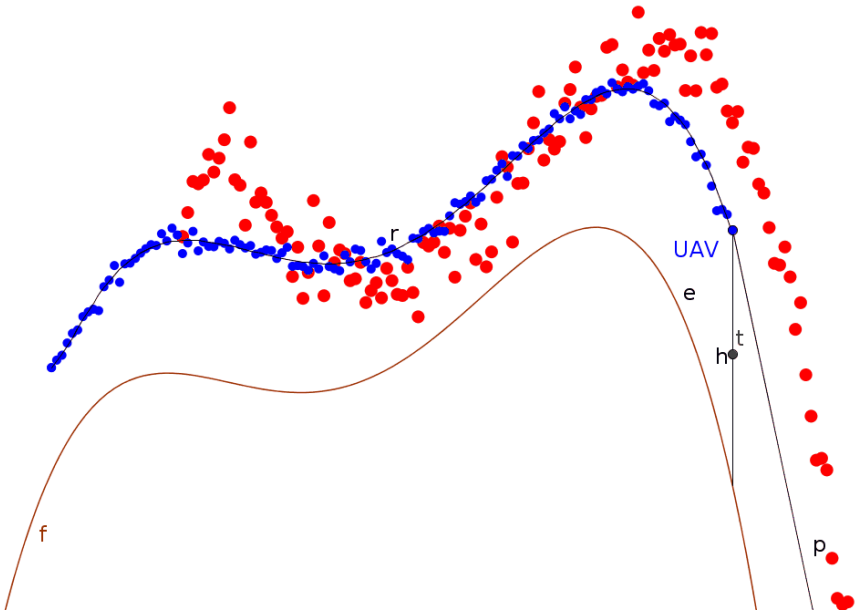


Figure 5.20: Height regulator simulation with added noise. It shows the path the UAV performed with the blue dots, while the red dots represent the waypoints it received. t stands for the target height, h represents the current height, e the error and $\vec{p}$ the path the vehicle follows to the designated waypoint (This figure can be animated on Adobe Reader on desktop computers.)

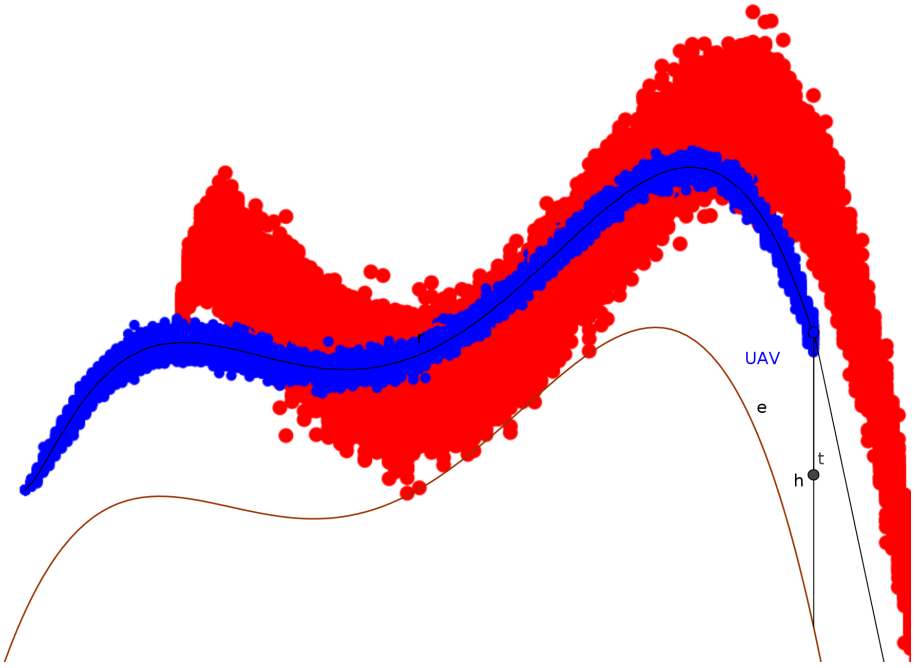


Figure 5.21: Thousand runs of the height regulator simulation with added noise. It shows the path the UAV performed in blue, while the red color represent the waypoints it received. The waypoints only present below the terrain a handful of times. However, the vehicle does not come any close to it.

Chapter 6

Conclusion

This work presented the design and construction of an UAV platform specifically intended to perform terrain following flights as well as a complete methodology to estimate the vehicle's flying height through the use of optical flow and proposes a regulator to keep it flying at the desired AGL altitude.

According to our experimental results we showed that using optical flow in real world scenarios to estimate an aircraft AGL altitude is possible, therefore performing terrain following on UAVs through monocular vision is a valid approach, as also demonstrated by our simulations.

Our experiments were all conducted in real world environments in an incremental way, gradually adding complexity to both, our vehicle and software. When we proceeded to the soccer field experiment we successfully obtained rough estimates of the UAV's AGL altitude. Adding the motion data from the IMU made the estimations even more accurate, as seen at the farm experiment and later confirmed by the LIDAR rangefinder added to our platform.

Unfortunately we were unable to perform actual terrain following flights from our height estimation algorithms due to a problem with our video transmitter, which broke and presented considerable noise after our attempts of repairing it. Therefore, to demonstrate the validity of our methodology we resorted to simulations in addition to the theoretical proof of how the technique works. We even added substantial noise to the simulations and the results still kept promising.

Our quantitative data reveals the algorithm works, but still needs to be further improved towards increased precision. Our accuracy, on the other hand, was demonstrated to be greater than comparable techniques, such as the traditional 2D approach. We are confident that using optical flow derotation will increase its robustness to yaw, therefore increasing the precision.

In summary, we showed the feasibility of using monocular vision to enable UAVs to perform terrain following flights. In addition we also created a robust, low-cost UAV platform capable of performing these and many other experiments that will benefit the laboratory in future research.

As additional contribution, the first part of this work, terrain classification, has already been published in a joint event between the 12th Latin American Robots Symposium and the 3rd Brazilian Symposium in Robotics [Campos et al., 2015]. We hope to get the remaining parts, namely, height estimation and terrain following, published in a near future.

6.1 Future Work

There are several paths for future research, from which we see the next steps as follows:

To perform experimental terrain following flights in real world scenarios, however, for this we need to replace the damaged video transmitter in order to perform the height computation in real time, sending commands to the UAV from a ground-station computer.

Evaluate the algorithms performance with a 3-axial gimbal, as it would eliminate the disturbances induced by vehicle yaw. This should greatly improve the performance of all algorithms, but even more for our techniques, which appeared more sensible to noise.

Perform software optical flow derotation on the 2-axial gimbal images and compare the results of this software stabilization with the ones from the 3-axial gimbal assessment. The physical stabilization should produce better results, but maybe software stabilization is enough.

Switch to a global shutter camera, which would allow us to completely remove the gimbal if software stabilization using IMU motion data presents enough for accurate height estimation.

Perform the computations on-board, through the use of embedded computers such as the NVIDIA's Jetsons TK1 or TX1, which feature very powerful GPUs capable of running CUDA optimized code.

Expand the capabilities of our algorithm one step further, towards collision avoidance, either by using several cameras pointed at different angles or using wide angle lenses.

Bibliography

Campos, I. S. G., Nascimento, E. R., and Chaimowicz, L. (2015). Terrain classification from UAV flights using monocular vision. In *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, pages 271–276.

Canis, B. (2015). Unmanned AircraftSystems (UAS): Commercial outlook for a new industry. Technical report R44192.

Chao, H., Gu, Y., and Napolitano, M. (2013). A survey of optical flow techniques for UAV navigation applications. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 710–716.

Ding, W., Wang, J., Han, S., Almagbile, A., Garratt, M. A., Lambert, A., and Wang, J. J. (2009). Adding optical flow into the GPS/INS integration for UAV navigation.

Expert, F. and Ruffier, F. (2015). Flying over uneven moving terrain based on optic-flow cues without any need for reference frames or accelerometers. *Bioinspiration & Biomimetics*, 10(2):026003.

Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In Bigun, J. and Gustavsson, T., editors, *Image Analysis*, volume 2749 of *Lecture Notes in Computer Science*, pages 363–370. Springer Berlin Heidelberg.

Franceschini, N., Ruffier, F., Serres, J., and Viollet, S. (2009). *Aerial Vehicles*, chapter Optic flow based visual guidance: from flying insects to miniature aerial vehicles. InTech.

Garratt, M. A. and Chahl, J. S. (2008). Vision-based terrain following for an unmanned rotorcraft. *Journal of Field Robotics*, 25(4-5):284--301. ISSN 1556-4967.

Geoffrey L. Barrows, Craig Neely, K. T. M. (2001). *Fixed and Flapping Wing Aerodynamics for Micro Air Vehicle Applications*, chapter Optic Flow Sensors for MAV

Navigation, pages 557–574. Progress in Astronautics and Aeronautics. American Institute of Aeronautics and Astronautics. 0.

Griffiths, S., Saunders, J., Curtis, A., Barber, B., McLain, T., and Beard, R. (2006). Maximizing miniature aerial vehicles. *Robotics Automation Magazine, IEEE*, 13(3):34–43. ISSN 1070-9932.

Herisse, B., Hamel, T., Mahony, R., and Russotto, F.-X. (2012). Landing a VTOL unmanned aerial vehicle on a moving platform using optical flow. *Robotics, IEEE Transactions on*, 28(1):77–89. ISSN 1552-3098.

Herisse, B., Oustrieres, S., Hamel, T., Mahony, R., and Russotto, F.-X. (2010). A general optical flow based terrain-following strategy for a VTOL UAV using multiple views. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3341–3348. ISSN 1050-4729.

Herisse, B., Russotto, F.-X., Hamel, T., and Mahony, R. (2008). Hovering flight and vertical landing control of a VTOL unmanned aerial vehicle using optical flow. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 801–806.

Horn, B. K. and Schunck, B. G. (1980). Determining optical flow. Technical report, Cambridge, MA, USA.

Horn, B. K. P. (1986). *Robot Vision*. McGraw-Hill Book Company, Cambridge, Massachusetts. MIT Electrical Engineering and Computer Science Series.

Horn, B. K. P. and Schunck, B. G. (1981). Determining optical flow. *ARTIFICAL INTELLIGENCE*, 17:185--203.

Horridge, G. A. (1986). A theory of insect vision: Velocity parallax. *Proceedings of the Royal Society of London B: Biological Sciences*, 229(1254):13--27. ISSN 0080-4649.

Hu, J.-S., Chang, Y.-J., and Hsu, Y.-L. (2009). Calibration and on-line data selection of multiple optical flow sensors for odometry applications. *Sensors and Actuators A: Physical*, 149(1):74 – 80. ISSN 0924-4247.

Humbert, J. S., Murray, R., and Dickinson, M. (2005). *AIAA Guidance, Navigation, and Control Conference and Exhibit*, chapter Pitch-Altitude Control and Terrain Following Based on Bio-Inspired Visuomotor Convergence. Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics. 0.

Lehrer, M., Srinivasan, M. V., Zhang, S. W., and Horridge, G. A. (1988). Motion cues provide the bee's visual world with a third dimension. *Nature*, 332:356–357.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674--679, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Moe, A. (2000). Passive aircraft altitude estimation using computer vision. Lic. Thesis LiU-Tek-Lic-2000:43, Dept. EE, Linköping University, SE-581 83 Linköping, Sweden. Thesis No. 847, ISBN 91-7219-827-3.

Netter, T. and Francheschini, N. (2002). A robotic aircraft that follows terrain using a neuromorphic eye. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 129–134 vol.1.

Romero, H., Salazar, S., and Lozano, R. (2009). Real-time stabilization of an eight-rotor UAV using optical flow. *Robotics, IEEE Transactions on*, 25(4):809–817. ISSN 1552-3098.

Ross, R., Devlin, J., and Wang, S. (2012). Toward refocused optical mouse sensors for outdoor optical flow odometry. *Sensors Journal, IEEE*, 12(6):1925–1932. ISSN 1530-437X.

Ruffier, F. and Franceschini, N. (2004). Visually guided micro-aerial vehicle: automatic take off, terrain following, landing and wind reaction. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, pages 2339–2346 Vol.3. ISSN 1050-4729.

Ruffier, F. and Franceschini, N. (2005). Optic flow regulation: the key to aircraft automatic guidance. *Robotics and Autonomous Systems*, 50(4):177 – 194. ISSN 0921-8890. Biomimetic Robotics {ISR} Biomimetic Robotics.

Soccol, D., Thurrowgood, S., and Srinivasan, M. (2007). A vision system for optic-flow-based guidance of UAVs. In *Proceedings of the Australasian Conference on Robotics and Automation. Brisbane, Australia.*

Song, X., Seneviratne, L., and Althoefer, K. (2011). A kalman filter-integrated optical flow method for velocity sensing of mobile robots. *Mechatronics, IEEE/ASME Transactions on*, 16(3):551–563. ISSN 1083-4435.

Srinivasan, M., Lehrer, M., Zhang, S., and Horridge, G. (1989). How honeybees measure their distance from objects of unknown size. *Journal of Comparative Physiology A*, 165(5):605–613. ISSN 0340-7594.

Zufferey, J.-C., Beyeler, A., and Floreano, D. (2010). Autonomous flight at low altitude with vision-based collision avoidance and GPS-based path following. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3329–3334. ISSN 1050-4729.

Zufferey, J.-C. and Floreano, D. (2005). Toward 30-gram autonomous indoor aircraft: Vision-based obstacle avoidance and altitude control. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2594–2599.

Appendix A

Unmanned Aerial Vehicle Assembly and Configuration

In this appendix we provide a guide with assembly and configuration instructions of the vehicle. This chapter is organized as follows: Section A.1 details the powertrain setup, section A.2 describes the gimbal and video camera installation procedures, section A.3 illustrates the connection of electronic components, section A.4 provides configuration instructions for the UAV, finally section A.5 shows how to plan basic missions.

A.1 Powertrain

The powertrain assembly is pretty straightforward, it consists of attaching the motors to the arms, connecting the motors to the ESCs, soldering the ESC power cables to the power distribution board integrated at the bottom plate of the quadrotor central section and connecting the arms to the bottom and top central plates.

Before this part it is important to verify if the ESC has a firmware suitable for multirotor applications, case contrary, it is necessary to replace the firmware by temporarily soldering cables to the ESC's PCB. To avoid this step chose ESCs that come with either the SimonK firmware or the BLHeli multi firmware. Afro is a well regarded ESC brand that is specifically designed for multirotors.

Figure A.1 shows how to connect the motors to the arms through screws as well as the arms to the central section, however, we recommend connecting the arms to the central section only after installing the ESCs and soldering their power cables to the power distribution board, which should be done as illustrated in Figure A.2. As standard, the red wire is the positive and the black wire the negative, failing to solder accordingly will destroy the ESCs. To attach the ESC to the arm we recommend using

zip ties. We do not recommend connecting the propellers until the vehicle is completely setup.

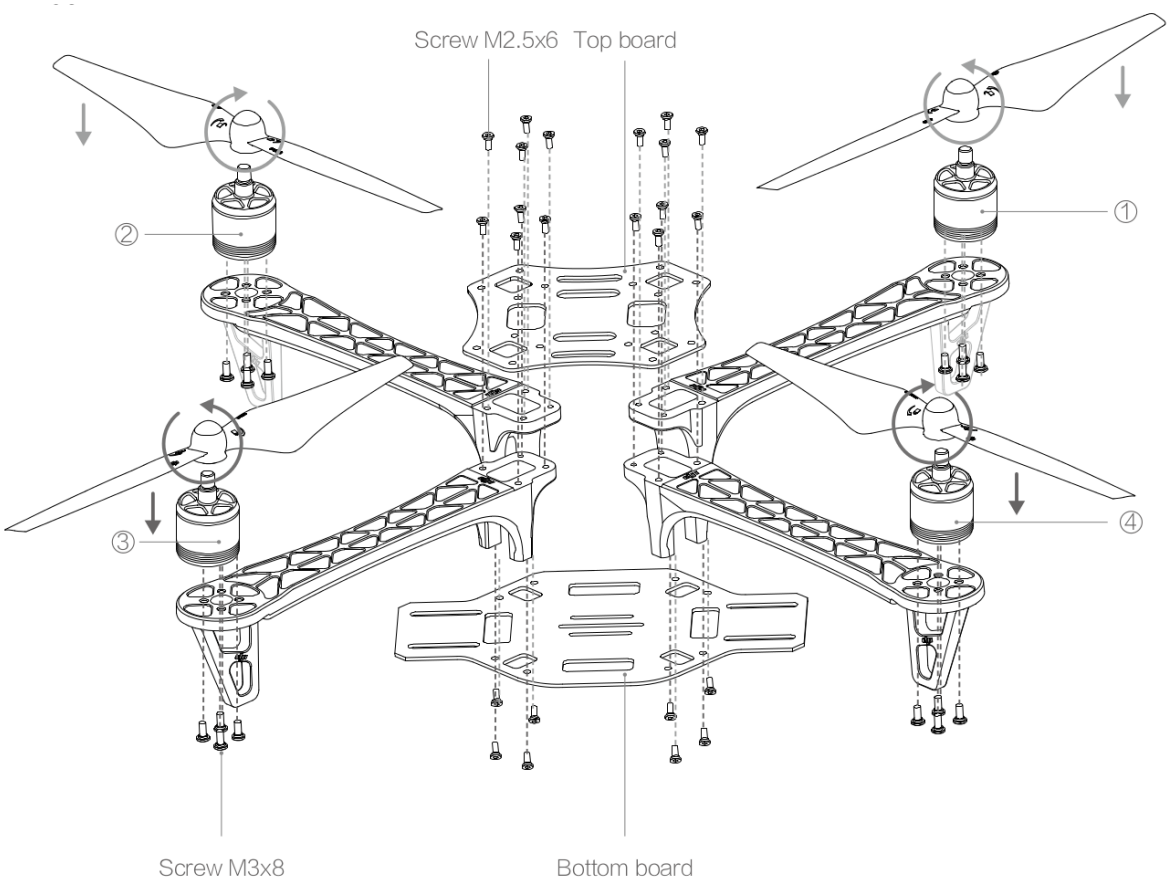


Figure A.1: Quadrotor frame assembly instructions.

Source: DJI F450 User Manual V2.2.

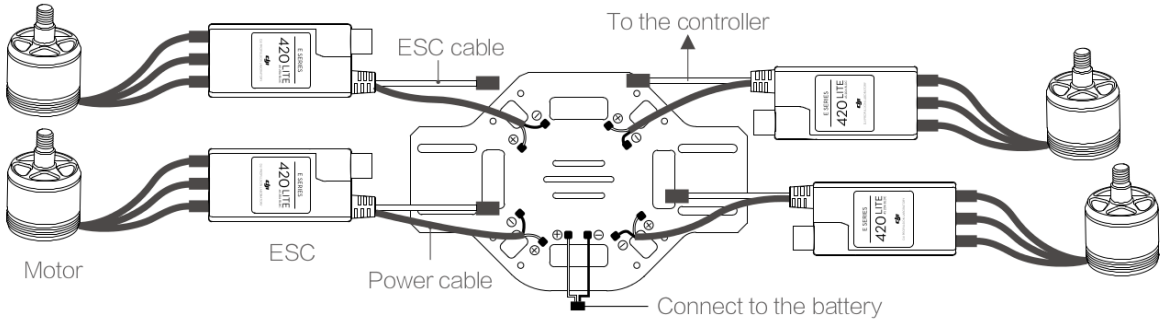


Figure A.2: ESC wiring instructions.

Source: DJI F450 User Manual V2.2.

A.2 Gimbal and Video Camera

To install the gimbal we need to increase the clearance from our UAV to the ground, to do so, we add a landing gear, since the landing gear will add thickness, as it will be sandwiched between the screws and the bottom plate, it is necessary to replace the screws that come with the frame kit for longer ones, that come with the landing gear kit. Figure A.3 shows the installation procedure.

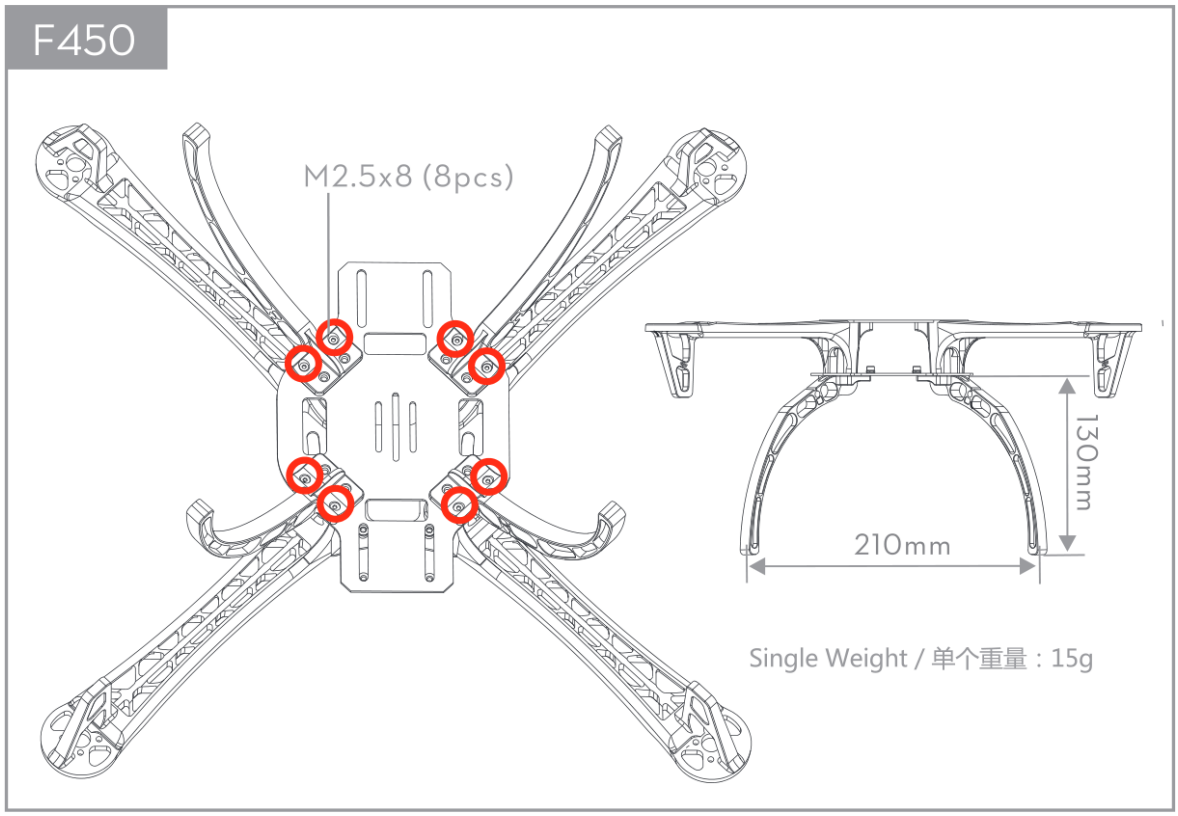


Figure A.3: Landing gear assembly instructions.

Source: Landing gear for flamewheel instructions.

After installing the landing gear, the gimbal installation is what follows, it is done according to Figure A.4. However, the kit did not come with corresponding nuts for the M3 screws, therefore we needed to get them separately.

To attach the camera to the gimbal one simply has to remove the two screws that hold the metal lock which fixes the camera in place. However, since there is little clearance at the camera connector side, we recommend connecting the power/video cable to the camera before fixing it to the gimbal. As there is low space only a slim cable (Figure A.5) will fit.

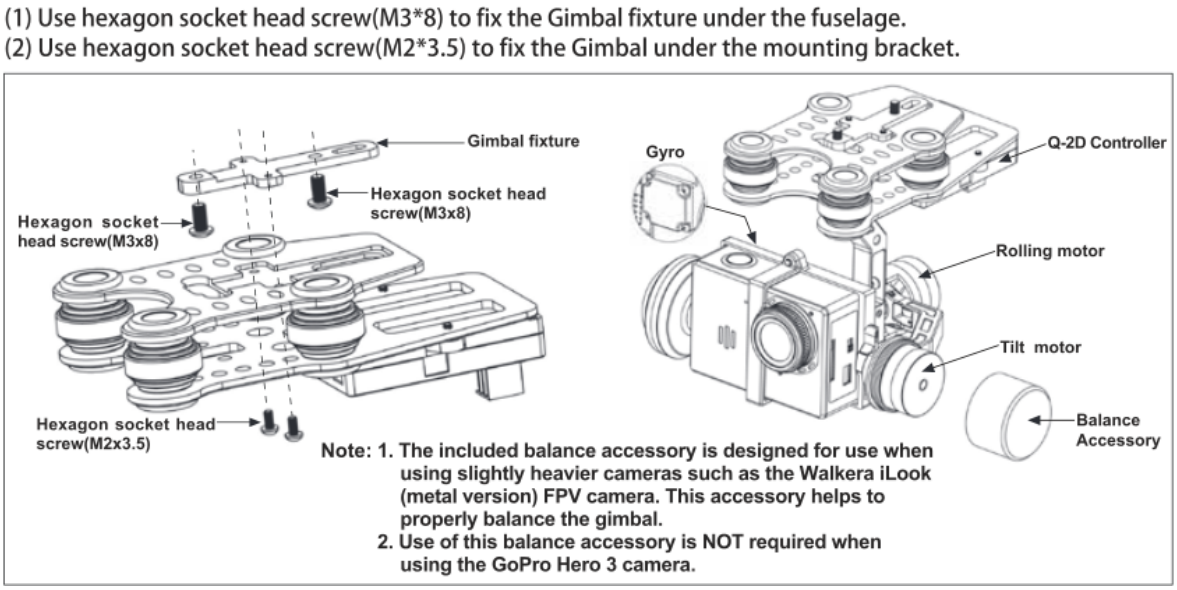


Figure A.4: Gimbal installation instructions.

Source: Quanum Q-2D user manual.

The video cable is connected to the input of the video transmitter and the power cable to any terminal on the output side of the flight controller. The transmitter should be below the rear of the bottom plate, attached with a zip tie and the antenna pointing down, this way there will be less interference with the other electronic components of the UAV. The best video transmitter frequency is 5.8 GHz for less interference with the GPS, telemetry and remote controller frequencies.

Figure A.5: GoPro slim power/video cable.

Source: `hobbyking.com`

A.3 Electronics

The core of the UAV is its electronic components, which consist of a flight controller board, a GPS receiver, a magnetometer, a telemetry radio, a voltage regulated power supply, and a remote controller receiver. A wiring schematic is shown in Figure A.6.

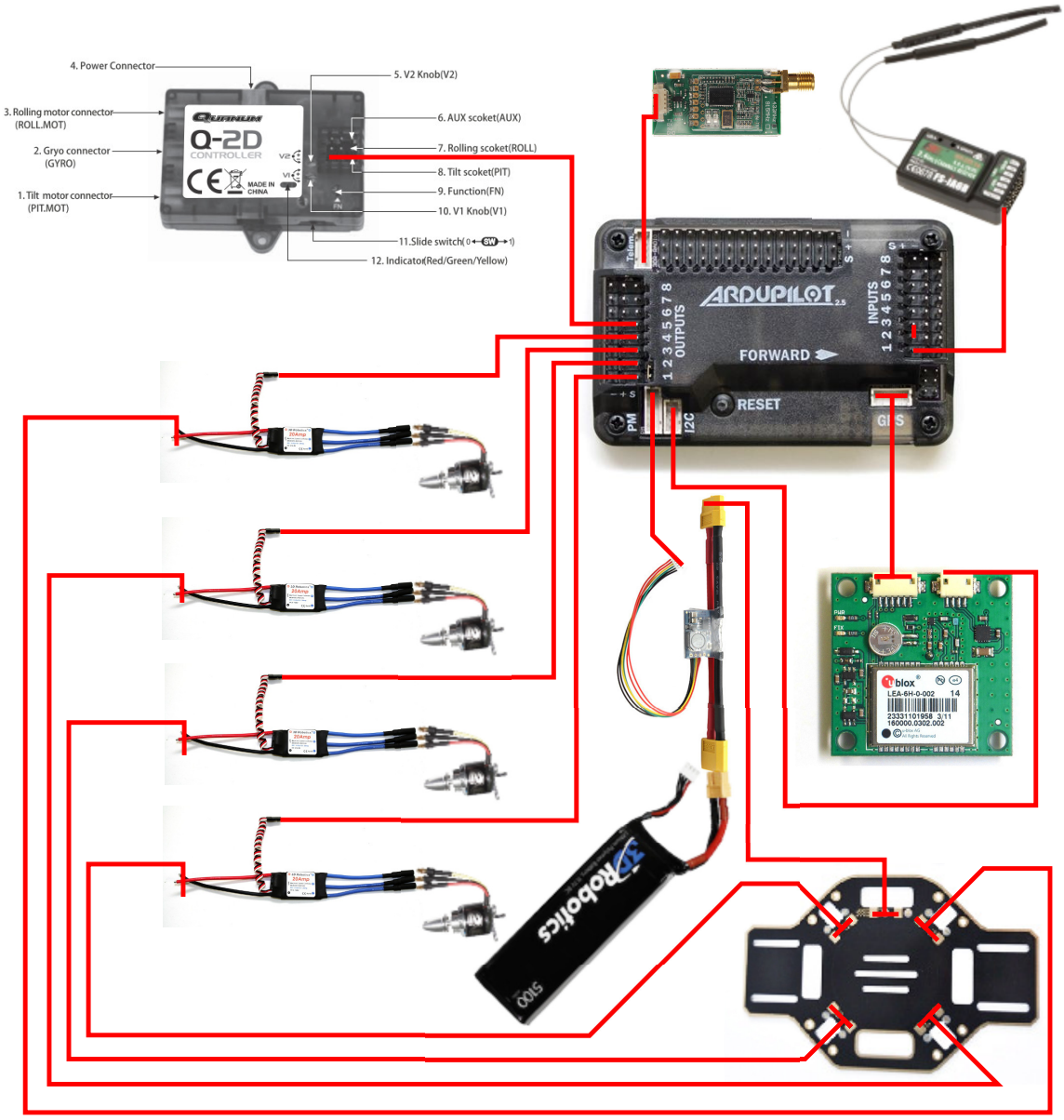


Figure A.6: Electronic components wiring diagram.

After connecting the components as shown in Figure A.6, all that is left is to arrange them the best way possible on the quadrotor, we placed the flight controller above the top plate of the frame, aligned according to the arrow pointing to the front, fixed it with adhesive foam, so it absorbs the vibrations caused by the motors and

propellers. We fixed the remote controller receiver the same way, by the side of the flight controller and covered them with a 25 CD pack lid. On top of the lid we placed the GPS, also observing the orientation, as it features a compass on the same board. To use the external compass one must also cut a PCB jumper to disable the onboard compass of the APM 2.5, this is important as quadrotors are very susceptible to electromagnetic interference from the motors. This is shown in Figure A.7.

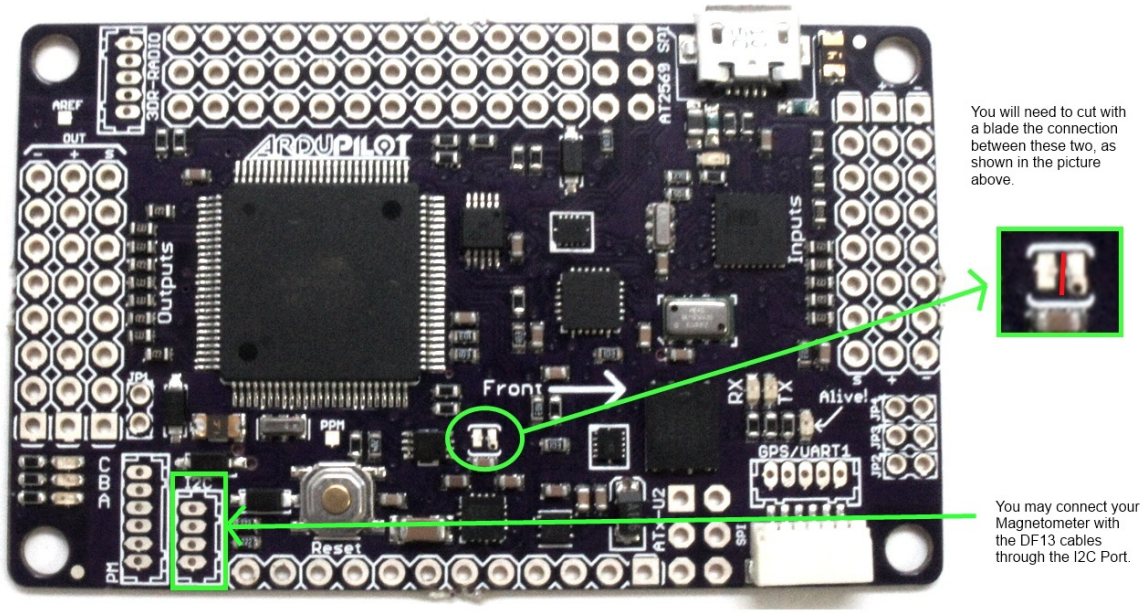


Figure A.7: APM board onboard compass PCB trace jumper to cut.

Source: `http://ardupilot.org/copter/docs/`

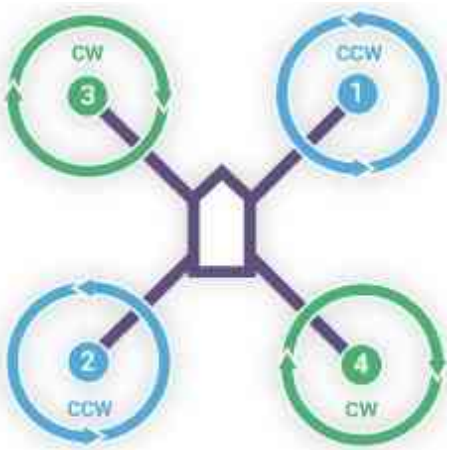


Figure A.8: Quadrotor motor layout.

Source: `http://ardupilot.org/copter/docs/`

Note that the motors of the quadrotor should be connected to the flight controller outputs according to Figure A.8. Motor 1 is front right and should rotate in counter-clockwise direction. Motor 2 is back left and should also rotate in counterclockwise direction. Motor 3 is front left and should rotate in clockwise direction. Motor 4 is back right and should also rotate in clockwise direction. Verify the spinning direction of the motors during the ESC calibration procedure before attaching propellers, if they are rotating in the wrong direction just swap any two wires of the given motor and it will spin in the correct direction.

In addition to these components we also used navigation lights (which can be connected directly to 3 cell batteries), a laser rangefinder (an I2C device connected in parallel with the compass) and an arduino pro mini to translate the MAVLink telemetry from the UAV to a telemetry enabled remote controller receiver, this way we could have the telemetry information not only on the ground station but also on the remote controller screen, however, this adds a lot of complexity as several difficult mods to the remote controller were necessary, therefore we present here the build we would currently choose, which features the TGY-i6 remote controller and the TGY-i6AB receiver.

A.4 Configuration

ALWAYS REMOVE THE PROPELLERS BEFORE CONFIGURING THE UAV.

In this section we present the configuration instructions for the UAV. First we will discuss the gimbal and remote controller setup, later we show the software setup instructions for the flight controller.

A.4.1 Gimbal

For the gimbal, put the switch “sw” into position 1, then use the supplied plastic screwdriver to adjust the “V2” knob completely counterclockwise and the “V1” knob completely clockwise. This will disable the roll and allow the pitch to be controlled through a knob on the remote controller.

A.4.2 Remote controller and receiver

For the remote controller, if you have a PPM capable receiver then connect only the channel 1 to the first input of the flight controller, and add a jumper linking the second

and third inputs of the flight controller (just as shown in Figure A.6). Otherwise, just connect each channel on the receiver to the corresponding input of the flight controller.

The first step is to bind the receiver and remote controller, to do so one must do the following:

1. connect the binding cable to the bind port on the receiver;
2. power on the receiver by connecting the battery to the UAV;
3. press and hold the bind/range test button on the remote controller;
4. turn on the remote controller;
5. disconnect the battery from the UAV;
6. remove the binding cable from the receiver;
7. turn off the remote controller.

A video with more detailed binding instructions is shown in Figure A.9. If your PDF reader does not support the content, open the link in the caption.

Figure A.9: Remote controller binding.

Source: `https://www.youtube.com/watch?v=SnbP73vutww`

With a PPM receiver you need to set the receiver to PPM mode through the remote controller, one must do the following:

1. turn on the remote controller;
2. turn on the receiver by connecting the battery to the UAV;
3. press the OK key on the remote controller;
4. select the system option through the up and down keys;
5. press the OK key to enter the system options;
6. select RX setup and enter through the OK key;
7. select PPM output and enter through the OK key;
8. use the up key to toggle the option so it shows on;
9. exit to the previous screen by pressing the OK key;
10. disconnect the battery from the UAV;
11. turn off the remote controller.

To configure the remote controller and receiver it is necessary to enable the auxiliary channels of the remote to use the top right switches. We will later use the the 3-position switch to set the flight mode and the left knob to control the camera pitch angle.

1. turn on the remote controller;
2. press the OK key on the remote controller;
3. select the functions option through the up and down keys;
4. press the OK key to enter the functions options;
5. select Aux. channels and enter through the OK key;
6. use the up key to change the channel 5 source to SwC;
7. change to the channel 6 setup through the OK key;
8. use the up key to change the channel 6 source to VrA;
9. press the OK key to save the configuration;
10. press the cancel key to return to the previous menu;
11. turn off the remote controller.

The last step in the remote controller and receiver setup is to enable the failsafe. The failsafe allows the flight controller to know when the UAV lost connection with the remote controller and take an action accordingly, such as RTL (Return To Launch).

1. turn on the remote controller;
2. turn on the receiver by connecting the battery to the UAV;
3. press the OK key on the remote controller;
4. select the system option through the up and down keys;
5. press the OK key to enter the system options;
6. select RX setup and enter through the OK key;
7. select failsafe and enter through the OK key;
8. select the 5th channel through the top/down keys;
9. enter the 5th channel failsafe with the OK key;
10. use the up key to toggle the option so it shows on;
11. put the 3-position switch all the way to the bottom;
12. exit to the previous screen by pressing the cancel key;
13. repeat the previous step twice;
14. disconnect the battery from the UAV;
15. turn off the remote controller.

A video with more detailed remote controller and receiver configuration instructions is shown in Figure A.10. If your PDF reader does not support the content, open the link in the caption.

Figure A.10: Remote controller and receiver configuration.

Source: `https://www.youtube.com/watch?v=F4YDrRnfVl8`

A.4.3 Flight controller

Now we can proceed to the flight controller configuration. The first step is to get and install Mission Planner (MP), the ground station software that will manage the UAV. It can be downloaded at `http://firmware.ardupilot.org/Tools/MissionPlanner/MissionPlanner-latest.msi` This is a Windows software, but it is possible to run it on Linux through Mono. There is also a cross-platform ground station called APM Planner, but it lacks several features Mission Planner and it seems its development has stalled.

After that we need to load the last ArduCopter version into the APM, to do that, the following steps are necessary:

1. connect the USB cable to the micro USB port of the flight controller;
2. open Mission Planner;
3. select the appropriate serial port at the top right corner (Figure A.11), it should say "COM# Arduino Mega 2560";
4. select the 115200 baud rate;
5. select the initial setup menu on the top bar (Figure A.12);
6. select the install firmware submenu on the left bar (Figure A.12);
7. select the quadrotor option, listed as ArduCopter V3.2.1 Quad;
8. if MP asks you to disconnect and reconnect the board, do so;
9. wait for the flashing process to complete, it will say "Upload Done";
10. close MP;
11. disconnect the flight controller from the computer.

Figure A.11: Serial port and baud rate selection.

Source: `http://ardupilot.org/copter/docs/`

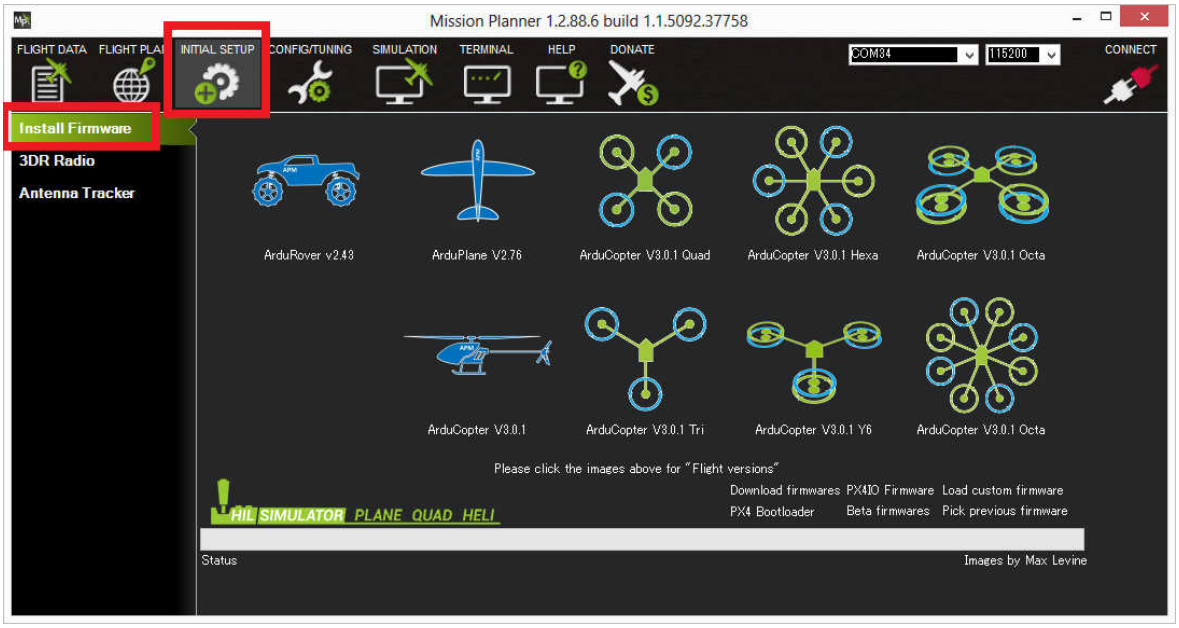


Figure A.12: Initial setup screen.

Source: `http://ardupilot.org/copter/docs/`

After flashing the firmware there are several configurations, they are all on the initial setup menu on MP. But this time we connect it through the telemetry radio instead of the USB cable. The steps are as follows:

1. connect the ground station radio to the computer;
2. connect the battery to the quadrotor;
3. select the appropriate serial port at the top right corner (Figure A.11), this time it is going to be a different COM port;
4. select the 57600 baud rate, as the telemetry defaults to this;
5. press the connect button at the top right corner (Figure A.11);
6. select the initial setup menu on the top bar (Figure A.12);
7. select the mandatory hardware/accel calibration submenu on the left bar (Figure A.13);
8. click the calibrate accel button and follow the instructions (Figure A.14);
9. do NOT click the calibrate level button;
10. go to the mandatory hardware/calibrate compass submenu on the left bar (Figure A.15);
11. on the top click on APM and external compass button;

12. click the live calibration button on the bottom left and follow the instructions (Figure A.16);
13. click the mandatory hardware/radio calibration submenu on the left bar (Figure A.17);
14. turn on the remote controller;
15. move the throttle stick all the way to the bottom;
16. click the calibrate radio button and follow the instructions;
17. move the sticks to all the sides as well as all buttons and knobs on the remote controller;
18. turn off the remote controller after the radio calibration;
19. click the mandatory hardware/flight modes submenu on the left bar (Figure A.18);
20. for flight modes 1 and 2 select stabilize, this mode will be active when the top-right 3-position switch of the remote controller is at the top;
21. for flight modes 3 and 4 select loiter this mode will be active when the top-right 3-position switch of the remote controller is at the middle;
22. for flight modes 5 and 6 select auto this mode will be active when the top-right 3-position switch of the remote controller is at the bottom;
23. click the save modes button;
24. select the optional hardware/camera gimbal submenu on the left bar (figure A.19);
25. select RC5 on the drop-down selector at the side of tilt option;
26. uncheck the stabilize tilt option;
27. set servo limits at 1000 and 2000;
28. set angle limits at -135 and 90;
29. select RC6 on the drop-down selector at input ch;
30. disconnect the quadrotor from MP by clicking at the top right button;
31. disconnect the battery from the quadrotor;
32. ESC calibration is the remaining step (Figure A.20);
33. **REMOVE THE PROPELLERS BEFORE CALIBRATING THE ESCS.**;
34. turn on the remote controller;

35. put the throttle stick to the maximum;
36. connect the battery to the quadrotor and wait around ten seconds;
37. disconnect the battery from the quadrotor;
38. reconnect the battery to the quadrotor;
39. the ESCs should beep, indicating they entered the programming mode;
40. wait at least ten seconds with the throttle still at the maximum;
41. put the throttle to the minimum and wait a few seconds;
42. move the throttle stick and check if the ESCs respond to it;
43. disconnect the battery from the quadrotor;
44. turn off the remote controller.

To enable all the six flight modes with the TGY-i6 remote controller, follow the instructions at `http://diydrones.com/profiles/blogs/flysky-fs-i6-flight-modes`.

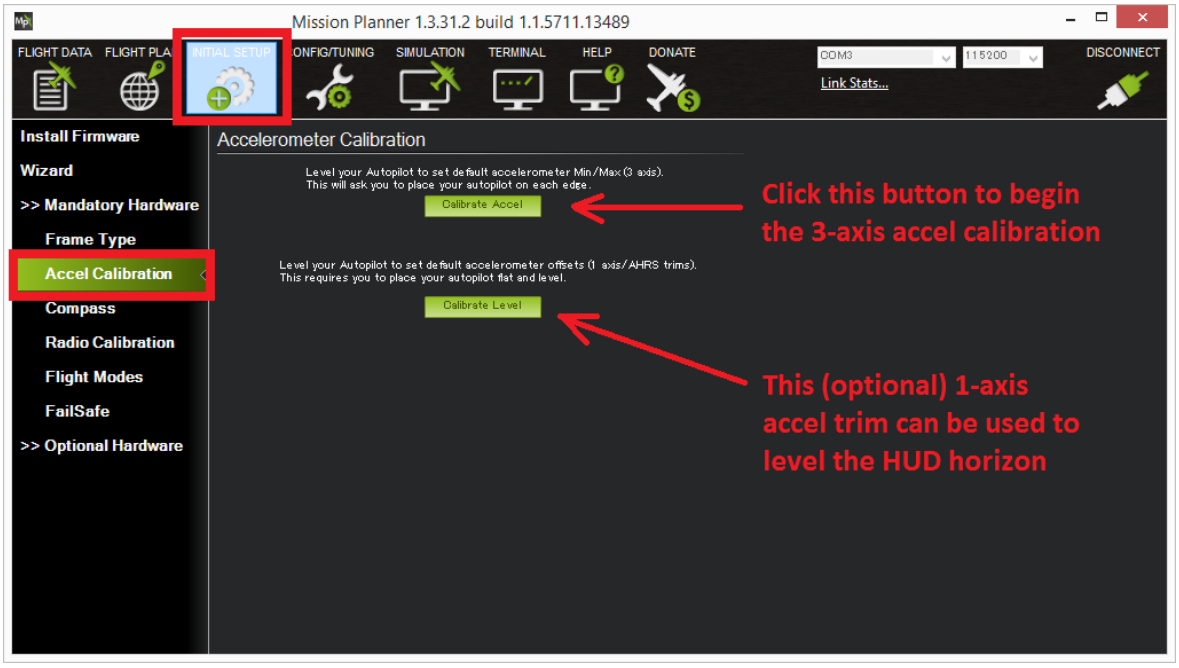


Figure A.13: Accelerometer calibration screen.

Source: `http://ardupilot.org/copter/docs/`

Figure A.14: Accelerometer calibration positions for an hexarotor.

Source: `http://ardupilot.org/copter/docs/`

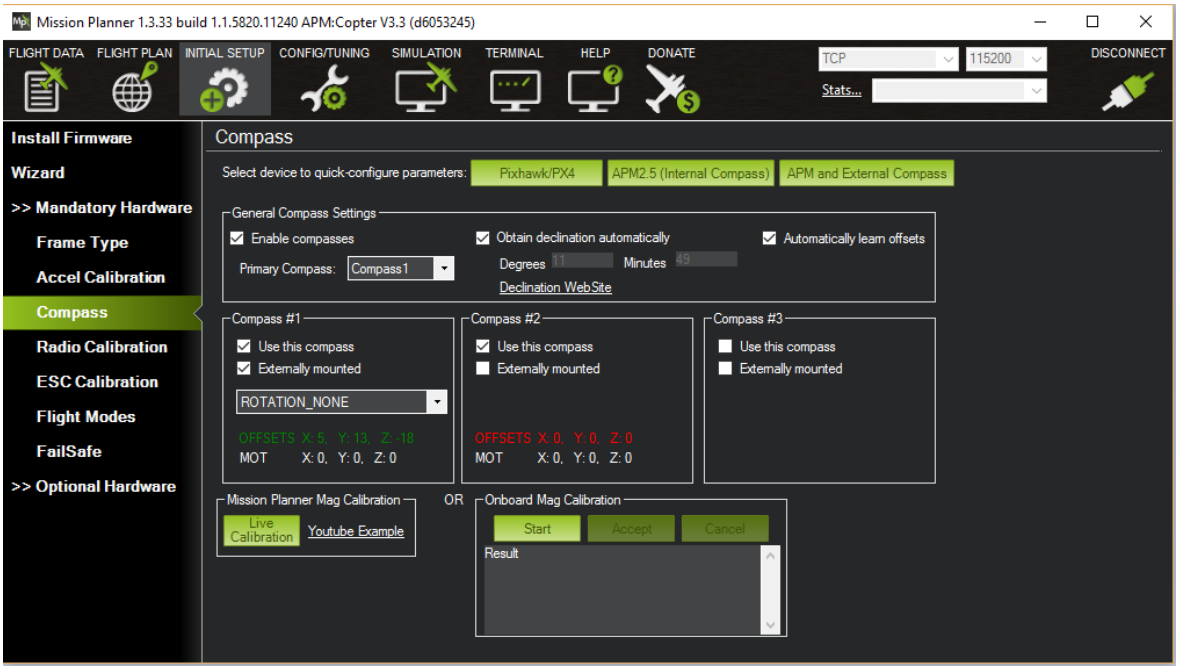


Figure A.15: Accelerometer calibration screen.

Source: `http://ardupilot.org/copter/docs/`

Figure A.16: Compass calibration procedure.

Source: `https://www.youtube.com/watch?v=DmsueBS0J3E`

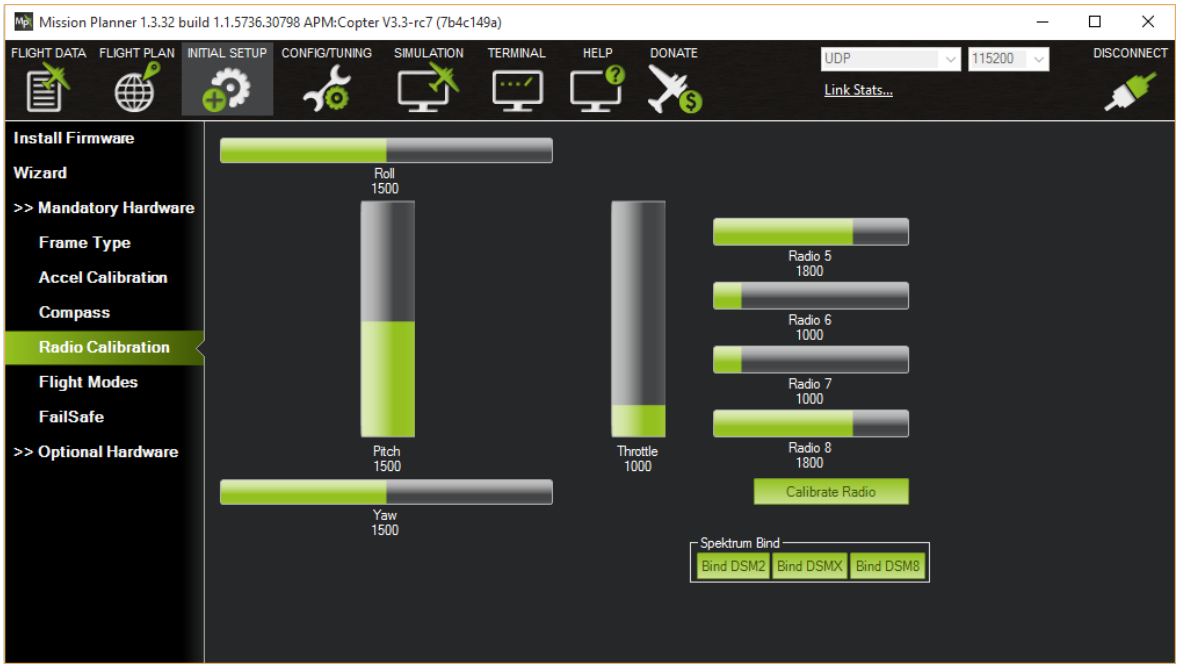


Figure A.17: Radio calibration screen.

Source: `http://ardupilot.org/copter/docs/`

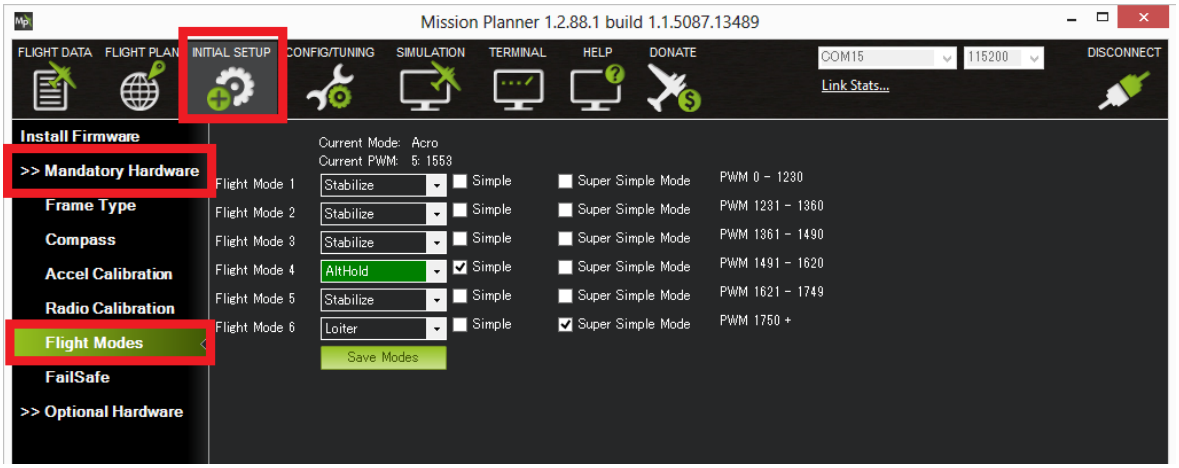


Figure A.18: Flight modes screen.

Source: `http://ardupilot.org/copter/docs/`

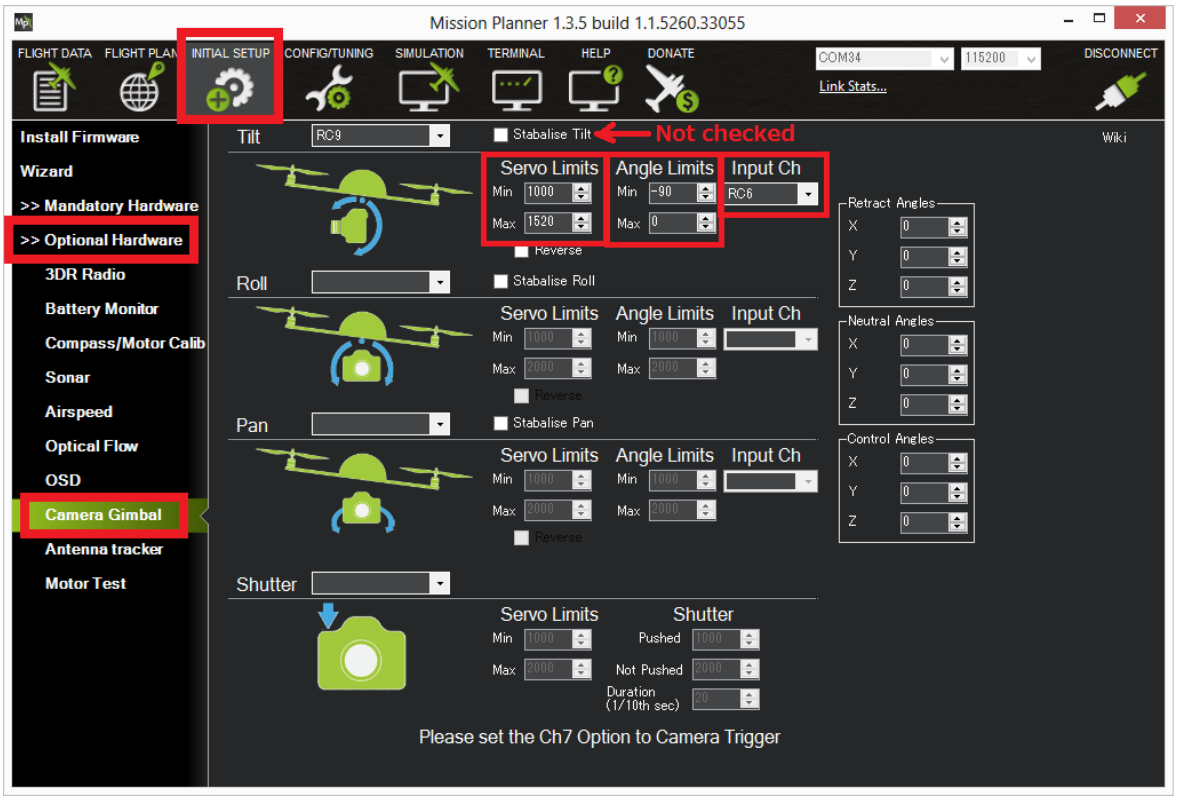


Figure A.19: Camera gimbal setup screen.

Source: `http://ardupilot.org/copter/docs/`

Figure A.20: ESC calibration procedure.

Source: `https://www.youtube.com/watch?v=gYoknRObfOg`

A.5 Mission Planing

We setup the remote controller failsafe and flight modes so that if the UAV loses connection with the remote controller it goes into auto mode, which executes the mission on the quadrotor memory, the reason we did that is because sometimes a mission takes the vehicle too far and the signal might drop, but we wanted the vehicle to keep executing the mission, this however requires well planned missions and must be done very carefully. Always set up a return to launch waypoint at the end of your missions and the vehicle will return to the place it took off from. In this section we present a brief introduction on mission planning. To plan a mission one must follow these steps:

1. connect the telemetry radio to the computer;
2. power on the quadrotor;
3. open MP and connect to the quadrotor;
4. select flight plan at the top bar (Figure A.21);
5. click the add below button and set the first waypoint to the takeoff option;
6. click on the waypoints you want the vehicle to go;
7. click on the add below button and set the last waypoint to return to launch;
8. press the write WPs button and wait for the complete transfer;
9. press the read WPs button and confirm that the desired waypoints are now saved at the flight controller memory;
10. go to the takeofff location;
11. turn on the remote controller;
12. set the flight mode switch to the top position;
13. connect the battery to the quadrotor;
14. wait a couple minutes for the quadrotor to get a GPS fix;
15. arm the quadrotor by moving the throttle stick to the bottom-right position;
16. move the flight mode switch to the bottom position;
17. watch the mission being executed;
18. if anything goes wrong move the flight mode switch to the center position, the vehicle will remain in the same place with the GPS help;

19. if the GPS signal is lost and the vehicle is drifting move the flight mode switch to the top position and control the vehicle to a safe landing.
20. disconnect the battery from the quadrotor;
21. turn off the remote controller.

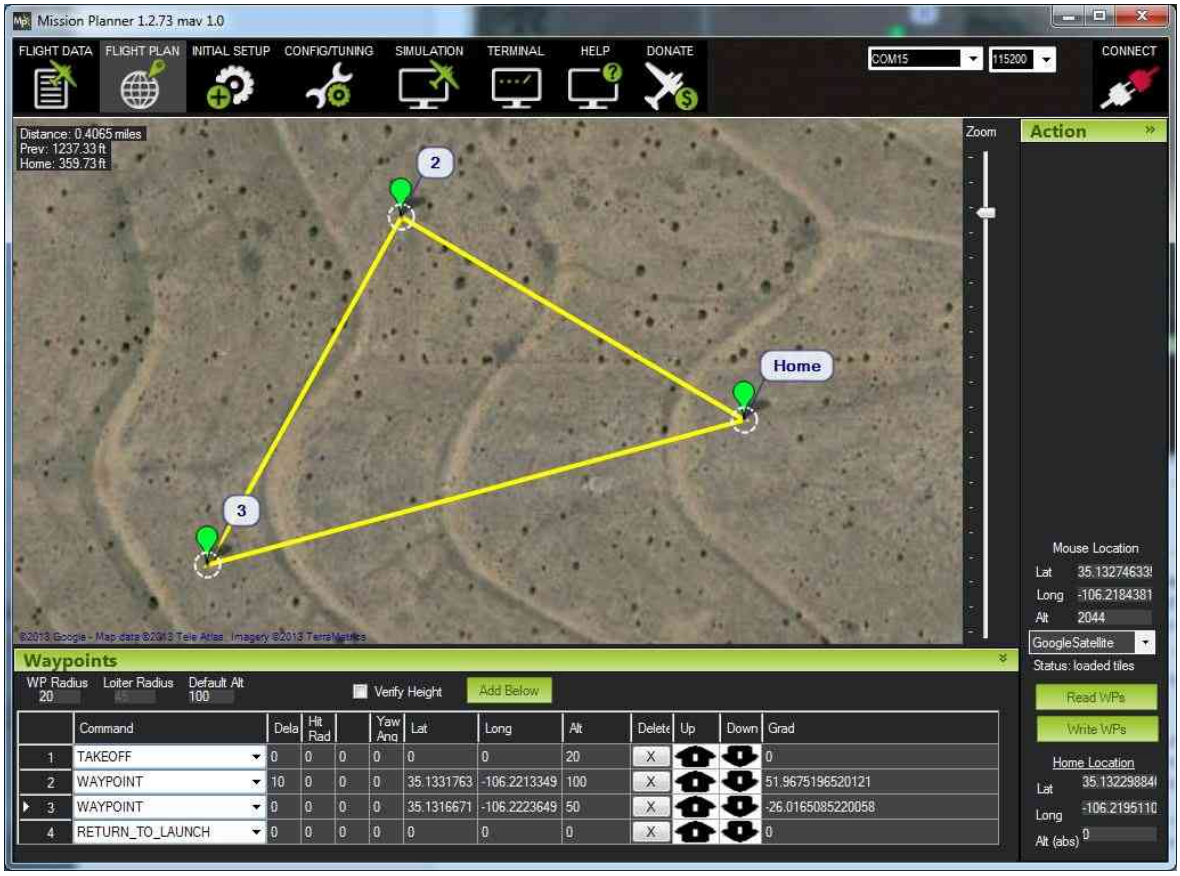


Figure A.21: Mission planing screen.

Source: `http://ardupilot.org/copter/docs/`

For further information about mission planing please visit `http://ardupilot.org/copter/docs/common-planning-a-mission-with-waypoints-and-events.html`. For other information about the ArduCopter firmware please visit `http://ardupilot.org/copter/docs/introduction.html`.