UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

# A Docking System for Battery Exchange

Paulo Tiago de Sousa Neves

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronautica
(Ciclo integrado de estudos)

Orientador: Prof. Doutor Kouamana Bousson

Covilhã, Outubro de 2014

# Resumo

Na agricultura de precisão é necessária uma plataforma de aquisição de dados ágil que pode navegar terreno vasto e complexo.
Os Quad-rotores têm grande capacidade de manobra mas geralmente têm autonomia curta, limitando a sua utilidade em vários cenários.
Como a autonomia é um problema de armazenamento de energia e as soluções são normalmente evolutivas, ao invés de revolucionárias, um sistema que pode restaurar a energia armazenada de forma autónoma é uma melhoria na operação de um avião não-tripulado autónoma.
Como o trabalho atual é de natureza concetual, uma configuração de simulação foi desenvolvida para fazer validação básica. O objetivo deste trabalho é integrar tecnologias já existentes. Assim um aspecto importante, é a interface entre componentes do simulador que nunca foram projetados para trabalhar juntos.
Exemplos dessa integração são o processamento de gráficos de uma cena 3D com os dados do motor de física e a exportação de modelos 3D do programa CAD Solidworks. Então, um fluxo de trabalho foi concebido para agregar diferentes tipos de dados. O fluxo de trabalho também foi desenvolvido para criar um ambiente no qual cada componente é o mais dissociado possivel de outros outros.
Todos os dados simulados alimentados ao sensor vêm do simulador de cinemática, exceto para a câmera. Para o caso da câmera, os dados precisam ser processados para um ambiente sintético 3D.
Exceto também para a câmera, os dados são usados pelo sistema de controle de dinâmica de vôo (FDCS), especificamente Arducopter. O módulo Arducopter executa voos no ambiente sintético como faria num real.
Um módulo adicional é ligado ao FDCS que aciona um plano de voo especial para aterrar visualmente em posições predeterminadas. Estes locais têm uma estação que permitem seguir pistas visuais e a troca de bateria.
É mostrado que um controlador PI modificado é suficiente fazer uma aterragem visual. Também é mostrado que as tecnologias off-the-shelf podem ser usadas em conjunto com outras especialmente desenvolvidas para criar plataformas de teste estaveis, permitindo a substituição seletiva de funcionalidade. Desta forma o trabalho pôde focar-se nos problemas a serem resolvidos de forma mais abstrata, em vez de implementar recursos que são detalhes práticos. A biblioteca de software MAVLink desenvolvida foi especialmente útil porque criou a ponte para integrar o planeamento de vôo especial desenvolvido e, controlar um módulo de piloto automático de quad-rotor em vôo.

## Palavras Chave

Atracagem, quad-rotor, troca de bateria, voo autonomo, Arducopter, aproximação visual.

# Abstract

In precision agriculture there is a need for an agile data acquisition platform that can navigate vast and complex terrain.
Small sized electrical rotor-craft have great maneuverability but generally have short endurance, limiting their roles in several scenarios.
As endurance is a power storage problem, and solutions are normally evolutionary rather than revolutionary, a system that can restore it's relatively low power capacity while remaining autonomous is an improvement in the operation of an autonomous unmanned airplane.
As the current work is of a conceptual nature, a simulation setup was developed to do basic validation. The goal of this work is to integrate already existing technologies thus, an important aspect, is the interface between simulator components that were never designed to work together. Examples of this integration are the graphics rendering of a 3D scene with data from the physics engine and the exporting of 3D models from Solidworks CAD program. A workflow was then devised to aggregate different types of data. The workflow was also developed to create an environment in which each component is the most decoupled from each other.
All the simulated data fed to the sensor models come from the kinematics simulator except for the camera. For the case of the camera, data needs to be rendered to a 3D scene.
Except for the camera, data is used by the Flight Dynamics Control System module(FDCS), specifically Arducopter. The Arducopter module executes flights in the synthetic environment as it would in a real one.
An additional module is connected to the FDCS that triggers a special flight plan to land visually in predetermined locations. These locations have a station that allows for the tracking of visual cues and battery switching.
It is shown that a modified PI controller is enough make a visual landing. It is also shown that off-the shelf technologies can be used in conjunction with custom developed ones to create stable testing platforms, allowing for selective replacement of functionality. This way work can focus on the problems to be solved in a more abstract way instead of having to implement features which are practical details. The developed MAVLink library was specially useful because it created the bridge to integrate the developed landing flight planning and control into a general quad-rotor autopilot module in flight.

## Keywords

Docking, quad-rotor, battery switching, autonomous, Arducopter, visual approach.

# Acknowledgements

This thesis is dedicated to the people who have been with me throughout all my academic path. I would be lying if I said all of them were as important, and a special heartfelt thanks goes to my parents, who did not merely support me financially but also carried the burden of keeping a posture of calmness and stability whenever my thoughts wandered off University. I also would like to thank Magdalena Maciejewska for all the things she has showed me both here in Covilhã but also in Poland.
Last but not least I would like to thank my friends for the unforgettable experience I have enjoyed in this city. I have learned so much about life from you I could write a book.

# Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| $\alpha$ | Proportionality constant between pixel and real world measurements |
| $\alpha$ | Thrust Scale Factor |
| $\alpha$ | Thrust scale factor |
| $\ddot{x}_T$ | Position second time derivative |
| $\delta_{hover}$ | Hover Accelerator Fraction |
| $\delta_{servo}$ | Motor Accelerator Fraction |
| $\Delta T$ | Time step |
| $\eta$ | Euler Angles |
| | |
| $\hat{x}_k$ | $k_{th}$ system estimate |
| $\lambda$ | Wavelength [m] |
| $\omega$ | Rotation Velocity |
| $\phi$ | Pitch |
| $\psi$ | Yaw |
| $\theta$ | Roll |
| $\theta_{nb,ni}$ | Angle between the earth fixed $n_{th}$ axis and the body fixed $n_{th}$ |
| $C$ | Camera Matrix |
| $d_bm$ | Distance of the motor from the origin of the body frame |
| $d_f$ | Propeller Drag Factor |
| $d_{bm}$ | Distance of the motor from the origin of the body frame |
| $det(R)$ | Determinant of the rotation matrix |
| $e$ | Nepper number |
| $f$ | Focal length |
| $F_i$ | ith Motor Thrust |
| $F_k$ | State transition matrix |
| $f_V$ | Field of view |
| $F_{aero}$ | Linear Aerodynamic Friction |
| $F_{grav}$ | Gravity Vector |
| $g$ | Gravity scalar |

$h$ Computed height from camera data

$H_k$ Observation Matrix

$I$ Identity matrix

$I_t$ Inertia Tensor

$K_k$ Kalman gain

$m$ Quadcopter mass

$n$ Number of motors

$P_k$ Covariance matrix

$R$ Rotation matrix

$R^T$ Transpose of a Rotation matrix

$S(\lambda)$ Perceptual color channel weighting function as a function of wavelength

$S(\omega)$ Skew Symmetric Matrix

$T_{aero}$ Aerodynamic Friction Torque

$V_b$ Body linear velocity

$V_i$ Inertial Linear Velocity

$v_k$ Measurement noise covariance matrix

$W_c$ World Coordinates

$w_k$ Process Noise Covariance Matrix

$Y$ Transformation Matrix

$z_k$ Observation Model

CIE Commission Internationale de l'Eclairage

DCM Director Cosine Matrix

FDCS Flight Dynamics Control System

GPS Global Positioning System

HVS Human Visual System

PID Proportional Integrator Derivative

POSIX Portable Operating System Interface

PPM Pulse Position Modulation

RC Radio Control

RGB Red Green and Blue

SITL Software In the Loop

SPD Spectral Power Density

# Chapter 1

# Introduction

## 1.1 Problem statement

The broader mission of this work was to develop and integrate a system that would autonomously land a quad-rotor in a static platform to perform switching of batteries. This static platform must have means of storing a magazine of charged batteries and swapping them with the discharged batteries of the landed rotor-craft. All this must be done autonomously, without human intervention.
This mission is relevant because it provides one solution to the problem that, small sized electrical rotor-craft generally have short endurance, limiting their roles in several scenarios. The scenario providing the context for the mission is precision agriculture.
In precision agriculture there is a need for an agile data acquisition platform that can navigate vast and complex terrain. Currently, precision agriculture is mostly done through airplanes to take aerial photography. The higher the airplane altitude, the bigger the area covered and less accurate the measurements. Conversely, to obtain better detailed imagery, lower altitude is necessary.
The problem with airplanes getting into lower altitudes is that they must fly slower so that that image artifacts like motion blur do not destroy the validity of the gathered data. Multiple passes at different angles also take more time due to the maneuverability constraints of an airplane. Also, an airplane cannot target trajectories following in between cultivation lines. This trajectories allow for acquisition of data related to the lower parts of agricultural flora. Currently only land robots are used for this kind of work, even though in a very limited scale due to low terrain flexibility and high operational costs compared to an individual person making an assessment on site.
Better agility is currently only possible in rotor-crafts but their range and endurance is severely limited by the power system. As endurance is a power storage problem, and solutions are normally evolutionary rather than revolutionary, a system that can restore it's relatively low power capacity while remaining autonomous is an improvement in the operation of an autonomous unmanned airplane.

## 1.2 Mission

The mission is such that, given a network of refueling stations in a given field geography, the rotor-craft can autonomously plan and find the correct station in which it makes the battery swapping. The processing and planning is done on the rotor-craft itself, allowing the stations to be "dumb" and isolated of any power and data as long as there is information about the absolute position of stations with charged batteries magazines. This magazines stations can either be charged manually or, if connected to a power system(solar power or mains power), act as a conventional charger.
For this work it was decided that the processing would happen on-board the quad-rotor. There were several guidance systems to consider, among those, radio and visual, but it was decided that a radio guidance system would not be cheap to design, and instead, a visual system was preferred. During the work requirements phase it was also decided that a full implementation of a Flight Dynamics Control System(FDCS) would be secondary, and an off the shelf system should be would

be used. The component chosen was the Arducopter from 3D Robotics. While it is manufactured and supported by a company, the source code is open which allows for deeper documentation of the FDCS behavior[1]. Arducopter allows for several autonomous navigation features, namely geographic waypoint following and position hold. It also has a communication interface which allows for attitude overriding, fly by wire. These are essential to execute the proposed mission:

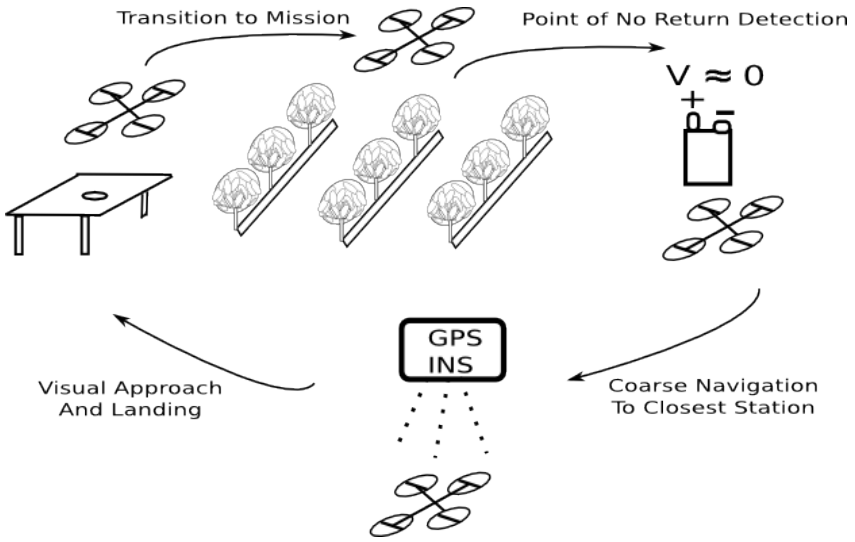



Figure 1.1: Mission Sequence

1. The rotor-craft exits from a station
2. The rotor-craft executes a mission.
3. The rotor-craft determines it needs to go to a certain battery switching station.
   (a) By receiving an external order.
   (b) By realizing it is reaching a Point of No Return.
4. The rotor-craft moves to a desired station(way-point) based on coarse absolute position sensors, e.g. Global Positioning System(GPS).
5. When inside the tolerance of the coarse absolute position, it switches navigation mode off to auto landing mode.
6. Attempts to recognize visual references of the station.
7. Based on recognized references progressively home in to the station.
8. Upon landing, swap batteries if possible, and proceed with the next instructions if any.

In the researching phase of this work it was hypothesized that the stations should have some kind of artificial lighting to facilitate the recognition of the station relative position to the rotor-craft, but it became clear that given the defined mission, open environments would greatly diminish the effectiveness of such visual cues, specially sunlight. It also does not fit well with the idea of a passive magazine station, even though low powered visual aids may still be of use in the later stages of the landing or at night.

[1]Discovered Bug in https://github.com/diydrones/Arducopter/issues/1026

As a conventional GPS system has coarse capabilities, it is possible that the conditions of lighting, references available, and relative position towards the designated station are not ideal when the switch to visual guidance happens. The conditions are then highly variable, making the reference recognition a special task, and the main problem to solve. It is also not expected that the system autonomously handles obstacles, meaning the stations must be placed in an area cleared of them.

## 1.3 Developed solution

As the current work is of a conceptual nature, a simulation setup was developed to do basic validation. The goal of this work is to integrate already existing technologies thus, an important aspect, is the interface between simulator components that were never designed to work together. Examples of this integration are the graphics rendering of a 3D scene with data from the physics engine and the exporting of 3D models from Solidworks CAD program. A workflow was then devised to aggregate different types of data. The workflow was also developed to create an environment in which each component is the most decoupled from each other.
To be sure if the workflow was capable of executing the goal of this work, a kinematics simulator was developed to provide data to simulated sensors. This simulated sensors include:

- Gyroscope.
- Accelerometer.
- Global Positioning System(GPS).
- Camera.

All the simulated data fed to the sensor models come from the kinematics simulator except for the camera. For the case of the camera, data needs to be rendered to a 3D scene.
Except for the camera, data is used by the Flight Dynamics Control System module(FDCS), specifically Arducopter. The Arducopter module executes flights in the synthetic environment as it would in a real one.
An additional module is connected to the FDCS that triggers a special flight plan to land visually in predetermined locations. These locations have a station that allows for the tracking of visual cues and battery switching.
It is shown that a modified PI controller is enough make a visual landing. It is also shown that off-the shelf technologies can be used in conjunction with custom developed ones to create stable testing platforms, allowing for selective replacement of functionality. This way work can focus on the problems to be solved in a more abstract way instead of having to implement features which are practical details. The developed MAVLink library was specially useful because it created the bridge to integrate the developed landing flight planning and control into a general quad-rotor autopilot module in flight.
In sum, the following technologies were developed:

- A simulation workflow.
- A physics and sensor simulator.
- 3D conceptual models of station for automatic landing.
- A visual pattern to be set on the station.
- A C++ MAVLink communication library.

- A PI flight controller.
- A visual pattern tracking algorithm.
- A gyro stabilized image projection correction system, software based.
- A low battery flight plan.

All of these technologies have been integrated together to make a validation of the proposed solution to the mission statement.

# Chapter 2

# Modeling and Theory

## 2.1 North-East-Down Referential

North-East-Down is a referential commonly used in aeronautics to described the world based on the perspective of the flying airplane. One of the drawbacks is that it is not related to the more common conventions of right hand rule. A representation of this referential can be found in figure 2.3.

## 2.2 Referential Rotations

Before proceeding to the kinematics description of the physics simulator an introduction to the referential rotation methodology is done. Following the notation found in [3] the relevant definitions are:

- The subscript $b$ denotes a coordinate which is measured from an origin of a body fixed referential. The origin of the body fixed referential is fixed on the center of gravity of the object to simulate.
- The subscript $i$ denotes a coordinate which is fixed to an arbitrary point in the world being simulated. For the specific case of this work this point is the point where the simulation begins and is immutable.

According to [3]:

> A rotation matrix is a matrix whose multiplication with a vector rotates the vector while preserving its length.

This preservation of length is a characteristic of orthonormal matrices, which rotation matrices are. In matrix form this property is formalized by considering $R$ as a rotation matrix:

$$\begin{cases} RR^T & = I \\ det(R) & = \pm 1 \\ R^{-1} & = R^T \end{cases} \quad (2.1)$$

These properties are very useful not only in terms of preserving the length of the vector, but numerically also allow for the inverse matrix to be obtained in a very cheap way, through its transpose. E.g, given a rotation matrix which rotates from body to inertial coordinates, to obtain the inverse operation from inertial to body coordinates, the multiplication only has to use the transpose matrix.
A rotation matrix that translates one referential to another is usually a multiplication of 3 rotation matrices, each rotating an axis by an angle. The rotation is related to the cosine of the angle between the axis. The name of each individual matrix is then appropriately called Director Cosine

Matrix, (DCM). To illustrate this concept consider a general DCM, with $\theta_{nb,ni}$ denoting the angle between the earth fixed $n_{th}$ axis and the body fixed $n_{th}$ axis[3].

$$R = \begin{bmatrix} \cos(\theta_{xb,xi}) & \cos(\theta_{xb,yi}) & \cos(\theta_{xb,zi}) \\ \cos(\theta_{yb,xi}) & \cos(\theta_{yb,yi}) & \cos(\theta_{yb,zi}) \\ \cos(\theta_{zb,xi}) & \cos(\theta_{zb,yi}) & \cos(\theta_{zb,zi}) \end{bmatrix} \quad (2.2)$$

If we consider each row and column to be an augmented matrix of basis's, then:

- The rows of R are the basis's of the body coordinates in terms of world coordinates.
- The columns of R are the basis's of the earth fixed coordinates in body fixed coordinates.

The reason the rotation matrices rarely comprise only cosine functions is due to the fact that, also often, the cosine of unsigned angles out of phase by $\frac{\pi}{2}$ can be better described by the sine of an angle. This can be visualized by a rotation of a referential through its origin in figure 2.1 and equation 2.3:

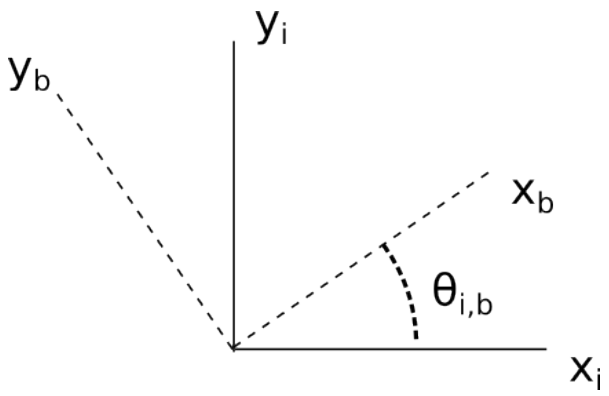



Figure 2.1: 2D Referential Rotation

$$\begin{aligned} R &= \begin{bmatrix} \cos(\theta_{xb,xi}) & \cos(\theta_{xb,yi}) & 0 \\ \cos(\theta_{yb,xi}) & \cos(\theta_{yb,yi}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta_{xb,xi}) & \cos(\frac{\pi}{2} - \theta_{xb,xi}) & 0 \\ \cos(\frac{\pi}{2} + \theta_{xb,xi}) & \cos(\theta_{xb,xi}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta_{xb,xi}) & \sin(\theta_{xb,xi}) & 0 \\ -\sin(\theta_{xb,xi}) & \cos(\theta_{xb,xi}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \cos(\tfrac{\pi}{2} - \theta_{xb,xi}) &= \sin(\theta_{xb,xi}) \end{aligned} \quad (2.3)$$

As mentioned before a 3D referential rotation can be composed by the multiplication of rotation matrices. Let the angles around $[x_i, y_i, z_i]^T$ be respectively $\eta = [\theta, \phi, \psi]$ roll, pitch and yawr, the Euler Angles then:

$$R_{xyz}(\theta,\phi,\psi) = R_x(\theta)R_y(\phi)R_z(\psi) \quad (2.4)$$

$$[x_i, y_i, z_i]^T = R_{xyz}(\theta,\phi,\psi)[x_b, y_b, z_b]^T \quad (2.5)$$

$$[x_b, y_b, z_b]^T = R_{xyz}(\theta,\phi,\psi)^T[x_i, y_i, z_i]^T \quad (2.6)$$

One of the problems of describing rotations through Euler angles is that there are angles that cause singularities, commonly called Gimbal Lock. Intuitively the singularities happen when the second axis of the rotation reaches a critical value, giving rise to a loss of a degree of freedom. The angles that cause gimbal lock are then related to the rotation order. Gimbal lock cannot be avoided for this rotation method but the rotation sequence can be defined in a way that makes the problem unlikely to surface. In the aeronautics field it is common to compose the rotation matrix as the sequence described in equation 2.4. This rotation sequence only produces a singularity when the pitch is $\pm\frac{\pi}{2}$ radians. For the implemented simulator this was the rotation sequence chosen as for this work it is not expected that a high pitch flight is necessary. The singularity and loss of a degree of freedom is visualized by taking the equation 2.15 and applying a pitch of $\phi = \frac{\pi}{2}$ radians as in equations 2.7 and 2.8:

$$R_y(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \quad (2.7)$$

$$R_y(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \quad (2.8)$$

There are other ways rotation methods that avoid the singularity but they are not without drawbacks[3]:

1. By obtaining rotation matrices through other means different than Euler angles, for example through rotation matrix integration. The disadvantage is that through integration, cumulative errors may make the matrix non-orthogonal. As rotation matrices must be orthogonal, the resulting matrices must normalized to be used.

2. By using quaternions, no singularities exist and their integration is numerically more stable. On the other hand their meaning and mathematical formalism is not physically so intuitive as Euler angles. The resulting integrated vectors also need to be normalized to unit quaternions to represent pure rotations. Also, for optimization the nature of the normalization may not be easy to deal with.

## 2.3 Physics simulator

The physics flight simulator was developed and integrated with the visual part. All units are in SI system although the models described are general for other units. In this section the equations used will be described, starting by the forces of the motors in the body frame arranged for our case in the '+' frame represented in figure 2.2:

$$\ddot{x}_T = \frac{F}{m} \quad (2.9)$$

$$F = \sum_{i}^{n=4} \begin{bmatrix} 0 \\ 0 \\ F_i \end{bmatrix} \quad (2.10)$$

$$\alpha = \frac{mg2\delta_{hover}}{n} \quad (2.11)$$

$$F_i = \alpha\delta_{servo} \quad (2.12)$$

As can be seen from equation 2.12 the individual motor thrust is not given by electrical constants or by motor models but from a scaling factor $\alpha$. This way the model is simplified and thrust becomes relative to quad-rotor configuration, namely the defined hover accelerator $\delta_{hover}$, the number of motors $n$ and body mass. $\delta_{servo}$ is the accelerator that goes from 0 to 1. There are better models, specifically ones that include blade element theory coupled with empirical results for vortex ring state where the rotors are barely producing thrust[4], but they are more complex.

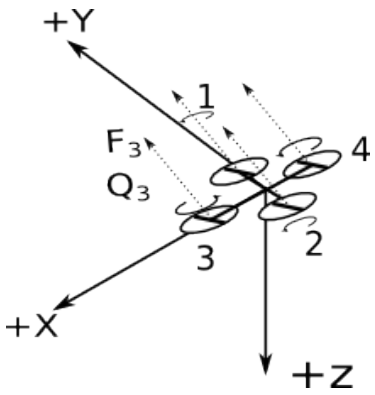



Figure 2.2: Body Frame

The motors are not in the center of gravity and the propellers, spin clockwise or counter clockwise in pairs. This is reflected on equation 2.18. As shown in the inertial frame illustrated by figure 2.3 the body frame can rotate and translate all its axis, thus having 6 inertial degrees of freedom. One important aspect for the control of a quad-rotor is that motors only produce thrust in one direction in the body frame thus it is under actuated. Only through other means to achieve body frame rotation it is possible to rotate the thrust vector to produce linear acceleration in the inertial frame. This is a controllability problem.

The final equation is given by 2.20, but the steps to arrive to it were very important to achieve the final implementation.

Starting with equation 2.13 a relation between the earth fixed referential and body fixed referential is presented. $V_i$, the velocity of the quad-rotor in the earth fixed coordinates is obtained through a transformation of the body velocity $V_b = [V_{bx}, V_{by}, V_{bz}]^T$. This transformation is achieved through a matrix composed of Euler Angles called $R_t$. $R_t$ was obtained as described in section 2.2. The notations used in the rotation matrices represent $\cos(\alpha)$ and $\sin(\alpha)$ as $c_a$ and $s_a$ respectively to make shorter representation.

$\omega_b$ represents the rotation velocity of the given body, this is, the vector along which the body rotates, not to be confused with the time derivative of roll, pitch and yaw, $\dot{\eta}_i$. As such, only a

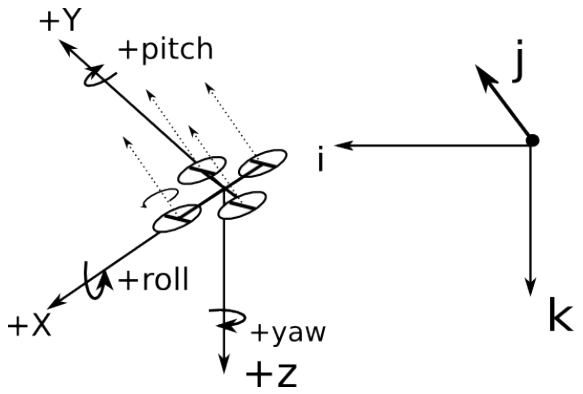



Figure 2.3: Body frame in Inertial Frame

rotation matrix is necessary to relate the attitude rate vector to the angular velocity vector, here denoted by $R_r$.
Through both angular and translational velocities, motions are defined for both referentials in equation 2.13.

$$\begin{cases} V_i &= R_t V_b \\ \omega_b &= R_r \dot{\eta_i} \end{cases} \tag{2.13}$$

$$R_r = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\phi C_\theta \end{bmatrix} \tag{2.14}$$

$$R_t = \begin{bmatrix} C_\phi C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\phi & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix} \tag{2.15}$$

To obtain accelerations, which can be related to torque and thrust, the time derivative of 2.13 is taken with the result in 2.16.
For the linear acceleration the chain rule is used and a property of the time derivative of a rotation matrix is applied: $\dot{R}_t = S(\omega)R_t$ where $S(\omega)$ is a skew symmetric matrix. A skew symmetric matrix can be thought as an infinitesimal rotation[5][6].
Given that for any vector $v$, $S(\omega)v = \omega \times v$, then with matrix associativity the last term for the linear acceleration is obtained in equation 2.16.

$$\begin{cases} \dot{V_i} &= R_t \dot{V_b} + R_t S(\omega_b) V_b = R_t \dot{V_b} + R_t S(\omega_b) V_b = R_t(\dot{V_b} + \omega_b \times V_b) \\ \dot{\omega_b} &= R_r \ddot{\eta} + \nabla R_r(\dot{\phi}, \dot{\theta}, \dot{\psi}) \dot{\eta_i} = R_r \ddot{\eta} + \nabla R_r(\dot{\phi}, \dot{\theta}, \dot{\psi}) \dot{\eta_i} \end{cases} \tag{2.16}$$

According to Newton's laws, linear and angular accelerations due to frame rotations can be balanced against external forces through the mass, $m$ and inertia tensor $I_t$ respectively.
Before introducing what external forces and torques are considered for the dynamics model a very important aspect of the following equations has to be mentioned: The orientation of the moments and direction of forces, due to the fact that the simulator must communicate sensor measurements

to Arducopter in a NED referential, as mentioned in section 2.1. Because of this, the expected order of the motors around the body frame for the Arducopter must match. The propeller rotation orientation must also match so that yaw rotations are in a coherent referential. A balance of forces in the body is then like in equation 2.17, where $F$, $F_{grav}$ and $F_{aero}$ represent, respectively, motor thrust $[0, 0, \sum F_i]^T$ the gravity vector $[0, 0, g]^T$and $K_v[V_b]^T$ the linear aerodynamic friction. In a similar way $T$ and $T_{aero}$ represent the motor torque and aerodynamic friction torque[7].

$$\begin{cases} \sum F_{ext} & = -F + F_{aero} + F_{grav} \\ \sum T_{ext} & = T - T_{aero} \end{cases} \tag{2.17}$$

For a quad-rotor, the rotors are in a predefined orientation regarding the body frame, specifically they produce trust along the negative part of it's z axis.

The torque produced by the asymmetries of thrust is what allows the change of attitude. In equation 2.18 $d_{bm}$ is the distance of the motor from the origin of the body frame and $d_f$ is a propeller drag factor.

$$\begin{cases} F_b & = \begin{bmatrix} 0 \\ 0 \\ \sum_{i-1}^{4} \end{bmatrix} \\ T_b & = \begin{bmatrix} d_{bm}\left(F_2 - F_1\right) \\ d_{bm}\left(F_3 - F_4\right) \\ d_f(-F_1 - F_2 + F_3 + F_4) \end{bmatrix} \end{cases} \tag{2.18}$$

Gathering equations 2.18, 2.17 and 2.16 applying the necessary rotations to the body forces and torques the result is angular and linear accelerations in the earth fixed coordinates shown in equation 2.20:

$$\begin{cases} T_{pb} & = I_t R_r \ddot{\eta} + I_t \left( \frac{\partial R_r}{\partial \phi}\dot{\phi} + \frac{\partial R_r}{\partial \theta}\dot{\theta} + \frac{\partial R_r}{\partial \psi}\dot{\psi} \right) \dot{\eta} + T_{aero} R_r \dot{\eta} + (R_r \dot{\eta}) \times (I_t R_r \dot{\eta}) \\ F_{pb} & = m R_t^T \ddot{V}_i + K_t R_t^T \dot{V}_i + m R_t^T G \end{cases} \tag{2.19}$$

$$\begin{cases} \ddot{\eta}_i & = (-I_t R_r)^{-1} \left( -T_{pb} + I_t \left( \frac{\partial R_r}{\partial \phi}\dot{\phi} + \frac{\partial R_r}{\partial \theta}\dot{\theta} + \frac{\partial R_r}{\partial \psi}\dot{\psi} \right) \dot{\eta} + K_{r\_aero} R_r \dot{\eta} + (R_r \dot{\eta}) \times (I_t R_r \dot{\eta}) \right) \\ \ddot{V}_i & = \left( m R_t^T \right)^{-1} \left( K_t R_t^T \dot{V}_i + m R_t^T G - F_{pb} \right) \end{cases} \tag{2.20}$$

Instead of matrix inversions transposes can be used, with all the numeric advantage it brings. For the linear velocity no constraints exist and the actual simulator implements a transpose. For the angular velocity the matrix is only orthogonal if the inertia moment $I_t$ matrix is also. Considering $I_t$ is a physical property of a body that cannot always be guaranteed to be orthonormal the simulator does not implement a transpose operation.

## 2.4 Color Models

There are many color models for the representation of color. Their purpose is to provide a standard reference for their representation. The importance of color models in machine vision is tantamount because the representation of the data given by most available capture devices is given in a human perceptual color model and not in a direct physical one. E.g, an image and it's elements(pixels) are not described by quantity of photons and wavelength(joules), but by subjective standardized quantities, related or not, to physical phenomenae. These standard references are important due to color being mainly a subjective property when evaluated by a person, i.e. same color may be perceived differently.

> Color is the perceptual result of light in the visible region of the spectrum, having wavelengths in the region of 400 nm to 700 nm, incident upon the retina. Physical power (or radiance) is expressed in a spectral power distribution (SPD), often in 31 components each representing a 10 nm band[8].

Even though this definition appears to conclusively model the human perception of light, it leaves room to an important detail: The Spectral Power Distribution of the 31 bands available is weighted differently from person to person. It follows that colors models can give weights to some bands arbitrarily. Thus, colors models can be classified in 3 types[9]:

- Human Visual System(HVS)
  - RGB
  - HSV
- Application Specific
  - YCrCb
  - CMY
- CIE
  - CIE XYZ
  - LAB
  - Luv

The spectral power distribution is the irradiance of a body or received by a surface:
It can also be called intensity. Normally, capture and display devices are not proportional to this linear light measure, due to non linearities throughout the bandwidth of wave lengths.
Luminance is a perception correction factor for the radiant power. This correction weights the spectral sensitivity characteristic to the eyes. The correction formulae is normally taken from CIE(Commission Internationale de l'éclairage), an international authority on illumination. Luminance maps the Y parameter of CIE color models.
Some of the relevant systems to this work will be detailed below:

### 2.4.1 RGB

In the Human Visual systems the most known model is the RGB. RGB stands for Red, Green and Blue. The RGB model is related to the HVS because the eye has 3 types of photo-receptors that closely represent Red Green and Blue. In fact the red component can be mapped to a long band width receptor, green as middle and blue as short[10].

To match the description of color with the element components of the human vision system, RGB has been widely used for capturing images. The RGB components are obtained with a weighting function to the wavelength $S(\lambda)$ and to the spectral power density $R(\lambda)$, $G(\lambda)$ and $B(\lambda)$. The conversion from spectral power density to each component is done through integration because it is a meaningful physical and mathematical operation.

$$\begin{aligned} R &= \int_{300}^{700} S(\lambda)\, R(\lambda) d\lambda \\ G &= \int_{300}^{700} S(\lambda)\, G(\lambda) d\lambda \\ B &= \int_{300}^{700} S(\lambda)\, B(\lambda) d\lambda \end{aligned} \tag{2.21}$$

Additive color systems are specified by the chromaticities of the primaries (red, green, blue) and the white point. There are no physically meaningful chromaticities or white points. The only way they can be meaningful is through artificial standards, like the television and CRT industry standards. This makes the RGB color model device dependent.

The problem is in most cases compounded by the fact that for an image to reach it's destination it must be first captured by one device and displayed by another. In a world where a big variety of displays and capture devices exist and where RGB is the preferred way, device manufacturers rely on calibrations that transform, back or forward, into a linear perceptually uniform color space. The normalization is usually denoted as a gamma correction and reflects a transfer function from the sensed stimulus to actual desired color values .

Even when the RGB color space can be correlated to actual physical phenomena there are other hindrances in its reliability:

According to literature[9] there is a high correlation between red and blue(0.78) and between blue and green(0.94).

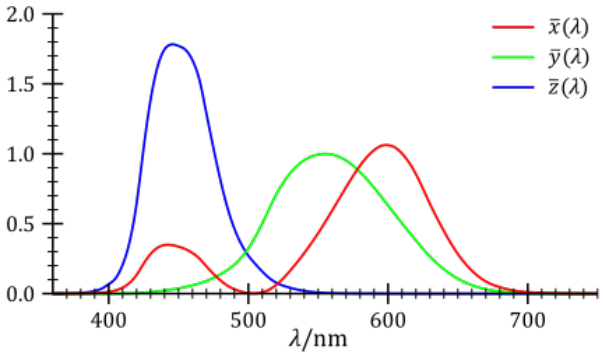
Another problem is that the euclidean distance of model coordinates in not correlated with perceptually different colors. It follows that image segmentation techniques like the ones used on this work become less robust in RGB color space.

### 2.4.2 CIE spaces

To create a context first it is important to talk about an outdated color standard CIE XYZ. The CIE XYZ was the first color space which considered standard observer data. This data and the underlying observer model does not change over time, thus much of this older model was taken to the current models. Worthy of mention is what the XYZ components represent:

- Y maps the luminance as previously stated.
- X and Z are standardized by the CIE based on empirical data from human perception.

Below is a picture with relation between spectral power distribution and x,y,z mapping[11]:




XYZ component magnitudes are not proportional to physical energy but spectral composition is related to characteristics in human vision.
Currently there are 2 widely used CIE standard color spaces. CIE Luv and CIE Lab. They are both valid and the reason there is not a single standard is because disagreements on the subject of chromatic adaptation. Both models have a clear constraint in their definition:

- Chromatic Adaptation
- Non Linear Visual Response

Chromatic adaptation relates to false perception of colors under different observing conditions, such as lighting. An example being a different color perceived the same as another, or the reverse, the same color being perceived as different on different environmental factors.
Non linear visual response is transformed into a linear numeric representation of colors. A color perceived to be similar to another should map closely in a chromaticity projection.
Both of this constraints allow for a color space where perceptual differences between colors can be represented in euclidean distances, numerically and graphically. In equation 2.22 an example is shown where L, a and b are components of a CIE Lab color vector and L, u and v are components of a CIE Luv color vector.

$$\begin{aligned} \Delta E_{ab} &= \sqrt{(\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2} \\ \Delta E_{ub} &= \sqrt{(\Delta L)^2 + (\Delta u)^2 + (\Delta v)^2} \end{aligned} \tag{2.22}$$

CIE color spaces are also able to represent every 3D vector of stimulus perceived by the average human observer.
Another important characteristic of CIE color spaces is that they are device independent and map the spectral power distribution to perceived colors through standardized and validated models[10]. Both variants of the CIE color space have the disadvantage that formulae for conversion, back and forth, RGB space are computationally expensive in real time. As such, industry standards for color encoding in video try to offer some of the qualities of a CIE color space but with a 1:1 mapping with RGB color space[8]. Such an example is the YCbCr color space.

### 2.4.3 YCrCb

YCrCb has its origins in analog television signals. The Y component represents the luma as mentioned before and the Cr and Cb represent components of chrominance. It is a format that maps directly to RGB, and has the same color space gamut. The conversion formulas from RGB to YCrCb are taken from [8]:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 298.082 & 0 & 408.583 \\ 298.082 & -100.291 & -208.120 \\ 298.082 & 516.411 & 0 \end{bmatrix} \begin{bmatrix} R_{255} \\ G_{255} \\ B_{255} \end{bmatrix} \quad (2.23)$$

Different formulas may be applicable depending on the numeric format in which the channels are represented.
The most intuitive way to understand the YCrCb map is to visualize it. The the concept of luma becomes clear in figure 2.4:

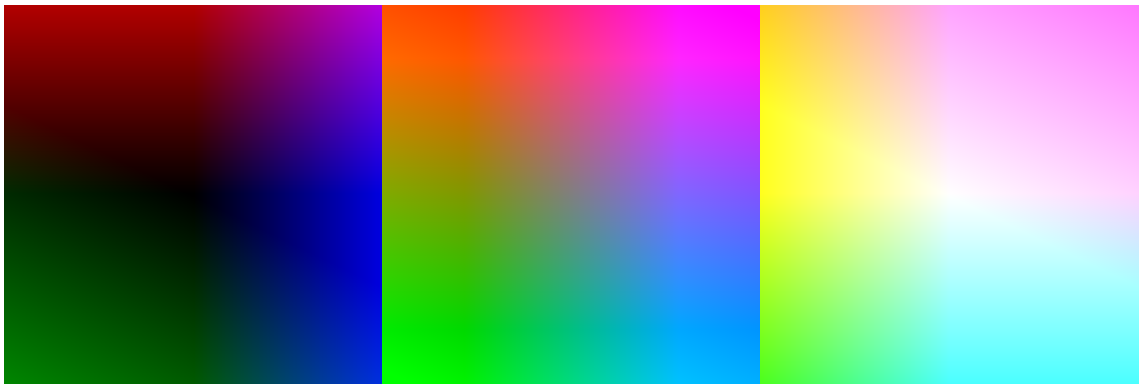


Figure 2.4: Illustration of YCrCb color space with varying Luma(Y)[1]

## 2.5 Camera Model and Perspective Projection

A camera model is a mathematical representation of the transformation of a 3D scene information into a 2D plane. In this 2D plan the image is formed. This transformation can be done through an orthographic transformation or perspective transformation[12]. An orthographic transformation is of limited use in uncontrolled environments because it assumes the 3D objects represented in the image to be orthogonally projected on the image plane. The perspective transformation/projection on the other hand can model the pose and distortion properties of a real camera with greater accuracy.

### 2.5.1 Homogeneous Coordinates

To represent projection transformations it is necessary to use a homogeneous coordinate system. Homogeneous coordinates are a system of coordinates like the Cartesian is a system of coordinates of Euclidean geometry. Homogeneous coordinates are useful because they allow a representation of an $n$ dimensional system in scale invariant $n+1$ coordinates[13]. The following examples are in 2 dimensions to keep the reasoning simple. Cartesian coordinates $[x, y]$ are represented in homogeneous like $[x : y : w]^T$ where the mapping is:

$$[x, y] \rightarrow [x/w : y/w : w'], \forall w \in R \backslash \{0\} \quad (2.24)$$

It can be seen that if $w = 1$ then the previous 2D point can be represented in 3D as in the original 2D plane.
In euclidean space rotation and scaling $\mathbf{R}$ is a different operation than translation $\mathbf{T}$:

$$[p_x, p_y] = [p_x, p_y]R + T \quad (2.25)$$

However in homogeneous coordinates $[p'_x : p'_y : 1]$ the previous operation can be described in matrix form in a single operation as:

$$[p_x, p_y] = [p'_x : p'_y : w'] \begin{bmatrix} s_x r_{00} & r_{01} & 0 \\ r_{10} & s_y r_{11} & 0 \\ t_0 & t_1 & 1 \end{bmatrix} \quad (2.26)$$

This is possible because $[p_x, p_y]$ is unchanged, but useful properties are now available as described in[13]:

- The resulting (homogeneous) vector has the same dimension as the argument (homogeneous) vector
- The matrix is invertible (assuming a non-zero determinant),
- Two or more transformations may be concatenated.

For $nD$ conversion of euclidean space to projective space the properties for matrix multiplication can also be achieved. In compact form:

$$[\boldsymbol{p}, 1] = [\boldsymbol{p} : 1] \begin{bmatrix} RI & 0 \\ T & 1 \end{bmatrix} \quad (2.27)$$

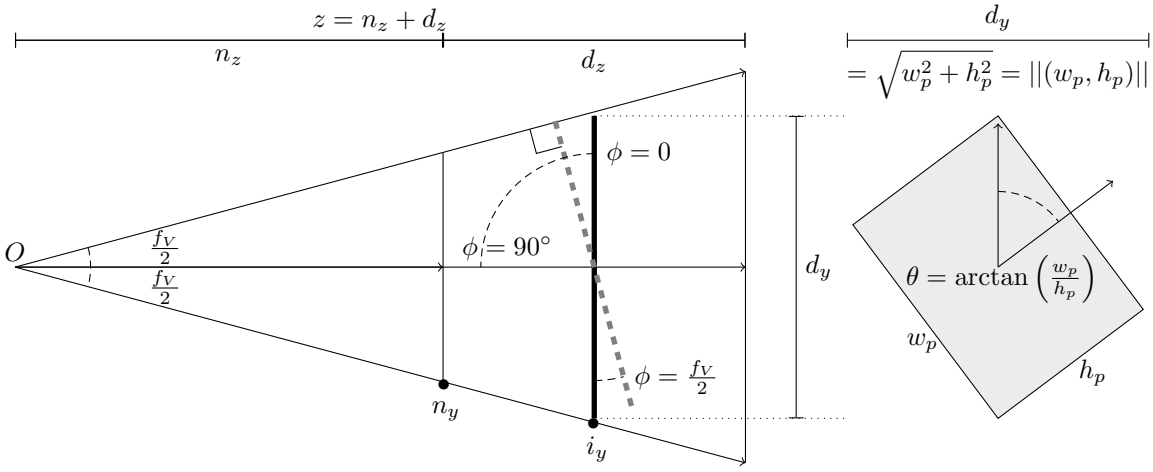



Figure 2.5: Representation of the image transformations in the camera referential.[2]

Until now the homogeneous coordinate $w$ has been fixed to 1 but if it is proportional to the distance of the origin $O$ a perspective projection can be obtained. In Figure 2.5 on page 15 consider $n = [x, y, z]$ to be a point and $i = [i_x, i_y, i_z]$ the image point projection contained inside the viewing pyramid. With triangle similarities it can be seen that if $\frac{i_y}{y} = \frac{i_x}{x}$ then $\frac{i_z}{z} = \frac{i_y}{y} = \frac{i_x}{x}$. This allows us to obtain projected coordinates of $n$ to different z planes:

$$[x, y, z] \to \left[\frac{x}{w} : \frac{y}{w} : \frac{z}{w} : w'\right] = \left[x\frac{i_z}{z} : y\frac{i_z}{z} : z\frac{i_z}{z} : \frac{z}{i_z}\right] = \left[x' : y' : z' : \frac{z}{i_z}\right] \quad (2.28)$$

The (orthographic) projection can be described in a transformation matrix form:

$$\left[x' : y' : z' : \frac{z}{i_z}\right] = [x' : y' : z' : 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/i_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.29)$$

The above projection causes the loss of depth information characteristic to the orthographic projection. A correction to make the $z' = i_z - i_z/z$ compresses the Euclidean space. The general perspective transformation in homogeneous coordinates is then:

$$\left[x' : y' : z' : \frac{z}{i_z}\right] = [x' : y' : z' : 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/i_z \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (2.30)$$

### 2.5.2 Projection

Most viewing frustums are not defined by unitary planes. They are defined by parameters of the camera namely the field of view, near and far field.
Considering Figure 2.5 on page 15 it is possible to adjust the perspective transformation's scale and the depth of field so as to match the characteristics of a real camera:

$$P = \begin{bmatrix} \frac{n}{n\tan\left(\frac{f_V}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{n}{n\tan\left(\frac{f_V}{2}\right)} & 0 & 0 \\ 0 & 0 & -\frac{(f+n)}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (2.31)$$

Let O be the origin of the camera frame, the image be given by $K = [u, v, 1]$ in homogeneous coordinates and the world space in the camera referential be defined by $W_c = [x, y, z]^T$ then there is a transformation given by $Y$ such that:

$$Y = CPTR_\phi R_\theta R_\gamma \quad (2.32)$$

As found in equation 2.33, $C$ is called the camera matrix because it contains intrinsic parameters of the camera, such as $f$ which is the focal length and U and V which represent the offset of the principal axis of the camera in pixels. Other higher order parameters are not represented but can be obtained from camera calibration tools. These high order coefficients reflect radial or tangential distortions from the lens. The camera matrix C is independent of the scene view so once the camera is calibrated it can be used with all images.

$$C = \begin{bmatrix} f & 0 & U & 0 \\ 0 & f & V & 0 \\ 0 & 0 & f & 1 \end{bmatrix} \quad (2.33)$$

$[R|T]$is a rotation and translation matrix which converts the camera referential to the world referential, through Euler angles. If Euler angles $[\theta, \phi, \gamma]$ are known then a rotation matrix can be constructed according to XYZ angle rotations:

$$R = R_\theta R_\phi R_\gamma = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\gamma) & 0 & \sin(\gamma) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\gamma) & 0 & \cos(\gamma) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.34)$$

For this work the Euler angles are obtained by the accelerometers of Arducopter. One of the shortcomings of this approach is that the refresh time of this angles can be too low. To increase this refresh time a penalty in utilization of the communication interface can be felt.
Based on Figure 2.5 on page 15a translation in the Z axis is necessary so that the image is never clipped outside the frustum.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{d}{2\sin(0.5fov)} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.35)$$

$$I_m = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = YW_c \quad (2.36)$$

$I_m$ denotes the final image in 2D corrected for the factors of transformation F.

### 2.5.3 Visual Height Measurement

Taking figure 2.5 as a reference it is simple to see that if a camera is guaranteed to be parallel to a plane and that plane has inscribed in it known features, simple trigonometry is sufficient to obtain the the altitude. The guarantee that the camera is parallel to a plane cannot be physically maintained without stabilization of the camera by a gimbaled mechanism. As the current work philosophy is one of simplicity, the stabilization is provided by using Euler angled measured by the Arducopter. On the other hand the visual features on a plane are given by color patches with known positions.
Taking a measured $d_y$ as the distance of 2 recognizable features, the altitude in pixels is then given by equation 2.37

$$h = \frac{d_y}{2\tan(fov/2)} \quad (2.37)$$

Assuming the lens of the camera does not introduce extraneous distortions and a square pixel, a one time calibration can be performed. The method used by this work is described in algorithm 1:
With $\alpha$ as a constant to convert pixel distances to millimeter:

$$d_{mm} = \alpha d_{pixel} \quad (2.38)$$

**Algorithm 1** Altitude calibration Algorithm

Require: Place camera in a way that the target fits one of the image dimensions
Require: Measure the target side that fits in the dimension in millimeter
1: $\alpha \leftarrow \frac{S_{mm}}{S_p}$

Thus it follows that height above the plane is given by:

$$h = \frac{\alpha d_y}{2\tan(fov/2)} \quad (2.39)$$

## 2.6 Kalman Filter

The Kalman filter can be used for other uses than filtering. Generally it is an estimator, and it's role is to provide an estimate $\hat{x}(t+\tau)$ on a previous time $\tau < 0$, on current time $\tau = 0$, or in the future $\tau > 0$. Respectively it can smooth, filter, or predict[14]. Throughout this work the Kalman estimator or smoother will be referenced as Kalman filter because most literature does so.
The canonic discrete Kalman filter state transition model is of the form:

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + w_k \quad (2.40)$$

$F_k$ is the state transition matrix and $\hat{x}_{k|k-1}$ is the estimated state in time knowing the previous state, as the Bayes probability notation. $w_k$ is the process noise covariance matrix. $w_k$ is assumed to represent a matrix following a normal distribution $w_k \sim N(0, Q_k)$. The complete transition model accounts for model forcing of a control input, but for the usage of Kalman filter in this work it will not be necessary and thus was not included.
Similarly for the observation model:

$$z_k = H_k x_{k|k} + v_k \quad (2.41)$$

$H_k$ is the observation matrix and relates in linear equation form the measured inputs with the the current state. Analogously $v_k$ represents the measurement noise covariance matrix and is assumed a normal distribution $v_k \sim N(0, R_k)$.
This model does not account for how the filter updates and runs recursively. The Kalman filter works by propagating the both the state $x_{k|k}$ and its covariance $P_k$ forward in time[15].

### 2.6.1 Kalman Prediction Algorithm

1. Predicted state estimate: $\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1}$
2. Predicted covariance estimate: $P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + w_k$

### 2.6.2 Kalman Correction Algorithm

1. Innovation with measurement: $y = z_k - H_k \hat{x}_{k|k-1}$
2. Kalman Gain with measurement: $K_k = P_{k|k-1} H_k^T [H_k P_{k|k-1} H_k^T + v_k]^{-1}$
3. State estimate Update: $\hat{x}_k = \hat{x}_{k|k-1} + K_k[z_k - H_k \hat{x}_{k-1}]$

4. Covariance Update: $P_k = (I - K_k H_k) P_{k|k-1}$

With the algorithm laid, some observations are important:
According to Reid et. al $w_k$ and $v_k$ represent uncorrelated, zero mean white noise processes with known positive semi-definite covariance matrices.
An intuitive way of understanding the Kalman Gain is to view it as a weighting gain of the sum of the estimated state and observed state.

### 2.6.3 System Model

The system model used in this work is one of constant acceleration $\ddot{x} = [0]$. It gives a high order description of the motion measured by the system. [14] provides a derivation of the covariance matrices for constant acceleration.
The state vector for a single coordinate is:

$$x = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \quad (2.42)$$

The continuous state space system equivalent to equation 2.40 is described like:

$$\begin{aligned} \tfrac{\partial x}{\partial t} &= Ax(t) + Bw(t) \\ A &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ B &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned} \quad (2.43)$$

To solve the partial the differential equation the Laplace transform is used against equation 2.43 and the discrete result for constant acceleration is:

$$F_k = e^{A\Delta t} = \begin{bmatrix} 1 & \Delta t & \Delta t^2/2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \quad (2.44)$$

As can be seen $F_k$ is the matrix representing the familiar Newton motion equations. The corresponding process noise covariance matrix is obtained from the Taylor expansion of the system:

$$w_k = q \begin{bmatrix} \Delta t^5/20 & \Delta t^4/8 & \Delta t^3/6 \\ \Delta t^4/8 & \Delta t^3/3 & \Delta t^2/2 \\ \Delta t^3/6 & \Delta t^2/2 & \Delta t \end{bmatrix} \quad (2.45)$$

Upon experimentation it was found that the process covariance matrix severely lagged behind the true and measured signal. A possible reason is that the process model is one of a linear system while in truth that is not the case, as such the form given for the process noise model may overestimate the random error.

### 2.6.4 Observation Model

The system model is not sufficient to ensure that estimation and the actual state do not diverge over a significant time period. It is clear that the system model needs initial information and corrections to provide an innovation close to 0. This information is integrated in the Kalman estimator through an observation model.
The observation model is described mathematically through a measurement matrix. This matrix relates a set of measurements or observations to the system model. In the current work the measured quantities are pixel points of the centroids of a given region of interest, and altitude. The pixel points are relative to the camera referential while the altitude is in a unit related to the camera model. Ideally the visual position measurement should be in the same referential as other position measurements available, for sensor fusion to be possible. Sensor fusion would enable a better position estimate and a more general system model. One of the potentialities would be online system identification, essential for more advanced control mechanisms with augmented states. Unfortunately the scope of this work is already very wide and this aspect was left out.
In this work it was assumed that X, Y and Z position are independent. This may not be true and can only be evaluated with a dynamics model of the rotor craft. For a quad-rotor the model is described in [7]. The independence assumption further helps in the computation of the Kalman gains with measurements because the matrix inversion operation is applied on a smaller matrix, improving the numerical stability and computational performance[15]. The observation model used, $H$, is analogous to the system model, a position measurement, with a conversion to velocity and acceleration.

$$H = [\ 1, \quad 1, \quad 0.5\ ]^T \quad (2.46)$$

The measurement noise matrix is more complicated to obtain, as it depends the complex camera model[16]. In this work it was decided to manually make adjustments to the gains so as to find a balance between noise and signal lag. The observation noise matrix $R$ is as in equation 2.47 where $I$ is a square identity matrix with dimension given by the observation model matrix:

$$v_k = 20I \quad (2.47)$$

Due to the incomplete observation model it is not possible to evaluate the observability of the system to be estimated.

## 2.7 Machine Vision

### 2.7.1 SLAM (Simultaneous Vision and Mapping)

An unmanned vehicle needs a representation of reality - perception - to be capable of , cognition and action[17]. SLAM treats the perception part of the problem, with data gathered from video cameras. The gathering of data, whether from monocular or stereo cameras almost always serves the purpose of self-localization, an essential one, as it is required at various levels in the whole system, from mission supervision to fine trajectory execution control[?].
SLAM is akin to a dead reckoning technique, meaning that the current system estimated state does not have a corresponding known ground truth. Instead it relies upon tracking the changes of

the successive states. As such position estimate errors are unbounded in time or state transition. The only way to set a boundary to these errors is to update the system with known ground truths. Machine vision SLAM generally relies on local invariant descriptors of the terrain or the environment for the construction of global maps.
Invariant features descriptors are extracted through knowledge of indirect parameters of the camera (intrinsic and extrinsic parameters) and selected feature type (geometric properties). This knowledge allows for the normalization of the extracted features and makes them invariant.

### 2.7.2 Local Feature Definition

[18], clearly defines what a local feature is in the context of machine vision:

> A local feature is an image pattern which differs from its immediate neighborhood. It is usually associated with a change of an image property or several properties simultaneously, although it is not necessarily localized exactly on this change. Local features can be points, but also edges or small image patches. Typically, some measurements are taken from a region centered on a local feature and converted into descriptors. The descriptors can then be used for various applications.

The feature type chosen for the present work is related to the image patches and will be detailed below, but there are other available algorithms that take extract different feature descriptors. These algorithms can be found in one of the most widely used computer vision libraries, both commercially and academically[19][20][18].

### 2.7.3 Image Thresholding

There are many image segmentation techniques, from complex to simple. In this work one of the simplest is tried, image thresholding. The reason this algorithm was chosen is because it fits well with the feature detection algorithm employed on the segmented image, is very fast [21] and benefits from processing parallelism.
Image thresholding consists of converting pixel value to True or False values based on a rule set. For instance considering an YCbCr image:

$$\begin{cases} T_{ij} = 1 & , I_{ij} \subset \{[Y_m, Y_M], [Cb_m, Cb_M], [Cr_m, Cb_M]\} \\ T_{ij} = 0 \end{cases} \tag{2.48}$$

This simple technique can be very effective with a color system like the YCbCr [22] because the group of the Y component - luminance - can be defined as a wide band, while the Cr and Cb components - chrominance - can ideally be narrowly defined, thus making the segmentation effective on different light environments with short computational resources.
In figure 2.6 it can be seen that the chosen patch colors are distributed in 5 of the 6 vertexes of the YCbCr color space. It follows that the 5 colors are the most separated possible, enabling each color to be defined with a wider group range in chrominance without fear of overlap. As can be seen next, a real camera is not able to have the color dynamic range necessary to capture colors with so extreme chrominance values.
In figure 2.7 some information may be inferred:

- Yellow has a very wide Cb band and overlaps with Green in the chrominance plane. The real distinction is in Y component, which is not visible because it is in the depth axis of the plot.

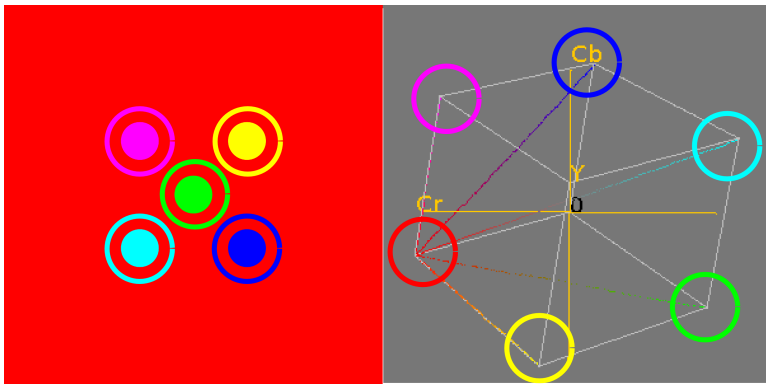



Figure 2.6: Original Target pattern in CrCb referential

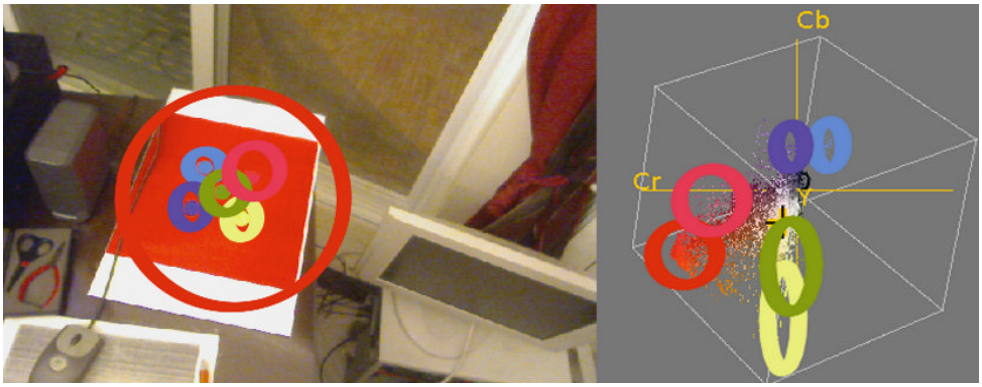



Figure 2.7: Plot of captured relevant colors in YCrCb color map

- Purple is perceptually very similar to Red but in the plot the 2 color ranges are distinctly spaced.
- Both blues are distinguishable both in the visible plot but also in the Y plot so they can be can be represented by distinct ranges.
- The right side of the cube is severely under represented in the image. Comparing with figure 2.6the green band chrominance was severely clipped
- A better distribution of colors can be obtained with a better crafted target.
- The faithful reproduction of colors in the printed target is very important.
- The red color is the most distinct color.

There are other ways of making the thresholding technique more resilient to lighting changes and camera noise. An important one is applying a Gaussian a motion blur effect, as can be seen in 2.8: The important thing to note in figure 2.8 is that the plot has a much narrower spread of points in the color map. This fact directly allows for a narrower definition of the color to detect, thus reducing the possibility of false positives. As treated in the probabilistic thresholding, blurring can also reduce the possibility of a single real region being split because of a single line of noise. The use of a blurring filter is dependent on the camera performance because it adds overhead on the image analysis. A way to perform an effect similar to blurring is by averaging areas of smaller size than the image. Blurring and averaging act as a low pass filter to features smaller than the mesh area.
The algorithmic implementation of equation 2.48is simple but slightly more complex.

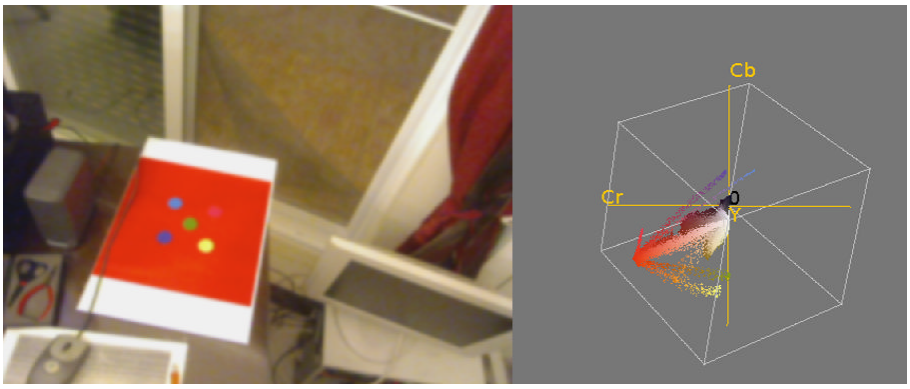



Figure 2.8: Plot of blurred($\sigma = 2px$) captured colors in YCrCb

Algorithm 2 Range Thresholding[23]

if $Ch_1 \geq Ch_{1lower}$ AND $Ch_1 \leq Ch_{1upper}$ AND
$Ch_2 \geq Ch_{2lower}$ AND $Ch_2 \leq Ch_{2upper}$ AND
$Ch_3 \geq Ch_{3lower}$ AND $Ch_3 \leq Ch_{3upper}$ then
    $pixel\_color \leftarrow color\_class$
else
    $pixel\_color \leftarrow 0$
end if

The algorithm exposes a weakness on such a naive implementation. 6 logical AND operations are required for each pixel. Considering a standard definition 3 channel image (640x640) there are $2,457,600$ required operations per frame. To further elaborate, on a high definition image (1920x1280) there are $14,745,600$ operations. This count of operations are hard to obtain in real time on a small scale machine. To improve the number of operations required per pixel, [12] described an algorithm that uses a precomputed look up table and only 3 logical AND operators per pixel. Considering a 3 channel image with 8 bits each:

Algorithm 4 Thresholding Lookup Table Generation

Require: $I\,(rows, collumns)$
$M \leftarrow M(I_{rows}, I_{collumns})$
$b \leftarrow$ channel bit depth
$n \leftarrow 2^b$
$C(channels, n) \leftarrow 0$
for $i < n$ do
    for $c \leq channels$ do
        if $i \geq T^c_{lower}$ AND $i \leq T^c_{upper}$ then
            $C(c, i) \leftarrow 1 << ch$
        end if
    end for
end for

In words, the algorithm creates a matrix representing all the valid color space, each element corresponding to a point in the color space. If a point is inside the required range a bit is set on that element. The bit shift operation allows various independent color ranges to be represented in a single color space. This way the real time operation can check for multiple colors with a single operation.
Algorithm 5 represents the real time operation when segmenting a captured image from video. To check whether a pixel is inside the required range, the channel values themselves work as matrix indexes to obtain the binary value of each channel of the color space matrix. The AND

Algorithm 5 Improved Range Thresholding

```
1: function isPixelInsideRange(pixel)
2:     logic_state ← 1
3:     for c ≤ channels do
4:         logic_state ← logic_state AND pixel(channel)
5:     end for
6:     return logic_state
7: end function
```

operation of all the selected channels returns if the pixel is valid or not, regardless of which color. This works because the bitwise AND returns 0 an all bits of different colors except on the only possible one. To further identify the color which was returned as valid, a simple operation to get the position of the highest bit is employed. This operation is supported in most consumer-grade processor architectures with a single instruction to count the number or leading or trailing zeros, making this operation very fast, around a clock cycle. This performance can be tapped in most architectures in the compiler used through the use of a compiler intrinsic __builtin_clz[24].
An intuitive way to visualize the algorithm is by taking the graphic in 2.8 and checking if at a given color coordinates there is a color point.
Although this technique is very fast it suffers from the drawback of the difficulty of finding the correct range of values for a wide variety of light conditions. A way to improve the robustness is to specify the channel ranges in a color space which separates the color description from the light in which the images were captured. As described in 2.4, a suitable color space would be either Cie or YCbCr color spaces. Considering most consumer grade cameras return image in a RGB related color space, YCbCr has several advantages against a Cie space:

- YCrCb is related to RGB color space.
- Cie space is described in Real coordinates while YCbCr is in Natural coordinates
- Cie space has a wider gamut but it of little use because the underlying range of most captured devices is related to RGB color space.

The software function used in this work always returns captured images in RGBa format, therefore if a different color specification is to be used in a different color space, a conversion has to take place. [22] also obtained segmented images with channel ranges described in another color space(HSV) to attempt to separate the color description from lighting conditions, but they implemented the conversion from RGB in the real time phase which imposed a significant performance penalty. In this work the conversion from YCbCr conversion happens during the look up table generation: The YCbCr indexes are converted to RGB indexes, this way the real time algorithm is independent of the color space in which the threshold was described, as long as there is corresponding RGB conversion. This independence allows for non linear, discrete, threshold color descriptions.
Even with an improved color description the resulting image segmentation is very fragile as can be seen in 2.9. These images were generated with the algorithms described and can show that what should be only one region per color can, unpredictably, be any number depending on the internal camera color correction algorithms.
A noteworthy detail is that the purple color on the upper right circle is misidentified and creates false positives. The same can be said about a small blob near the upper right corner of the image where a red feature was correctly identified but created a separate region. This is an important aspect of the false positive problem. The captured scene is not under control therefore it can contain multiple regions with the colors to be identified.

Figure 2.9: Target Ground Truth, Captured Image, Captured Image Thresholded

The goal of image segmentation according to a predefined color is to obtain a position descriptor of where the color region is. With this method, upon positive identification of the pixel color class, the positions of the pixels are averaged along the image, where $n$ is a vector with the number of pixels in the rows and columns direction.

$$[\overline{x}, \overline{y}] = \frac{1}{n} \sum [x_i, y_i] \quad (2.49)$$

As simple as this method sounds, it is very robust to white noise. White noise is assumed to be a random variation on the intensities of the pixel channels. Furthermore it is assumed to be Gaussian distributed with zero mean defined variance. It follows that noise will not affect the values calculated in 2.49. This assumption does not hold if lighting changes along the span of the image, introducing biases which will affect the estimated position.

### 2.7.4 Probabilistic Thresholding

One of the problems with regular binary decision thresholding was that, given a color range, all its values were equally likely to belong to the class of pixels to segment. This is not true. The landing target selected for image segmentation have clearly defined colors and a range is selected only to deal with environmental changes that still produce acceptable results. The farther from the initial value the range goes, less likelier is the pixel belonging to the class. Thus it is natural to describe the ranges in terms of probability distribution.
An accurate probability distribution of the pixel intensities(each for each channel) is very complex[16]: it requires the computation of the camera response function. This function can be affected by several factors and usually is not directly available by the camera manufacturer. The OpenCV library contains methods to obtain this function but its use was deemed outside the scope of this work.
The basic layout of the formulation changes slightly from equation 2.48 to 2.50:

$$\begin{cases} P(I_{ij} \subset C) > p & , T = 1 \\ T_{ij} & = 0 \end{cases} \quad (2.50)$$

This means that the threshold now is a probability $p \in [0, 1]$ and not a pixel intensity value. This probabilistic approach requires preprocessing of each pixel channel intensity to convert it to a probability, which can be a very expensive operation operation on this level of the system. The only way to obtain acceptable performance of the whole system with this approach was to

micro optimize the element operations to take advantage of vectorization functionalities of modern processors. Several distributions were tried but the one that returned the most accurate results was a linear distribution.
The linear distribution can be defined as found in 2.51:

$$P(I_{ijc}) = -\frac{1}{2^n}|i_{ijc} - i_{cp=1}| + 1 \quad (2.51)$$

In 2.51 $n$ is the full scale of the channel representation, and $i_{p=1}$is the pixel channel value desired. The difference relates to the relative error of the pixel to the desired value. Then, when the error is null the probability is 1 and when the error is maximum the probability is null.
To make the segmentation on multiple channels it is assumed that all the measurements are independent, an assumption that may not be true also depending on the camera model and even the color model. To integrate this dependences a naive bayesian filter was created but not tested. For the results obtained the assumption of independence was very satisfactory. The probability of independent measurements of different channels belonging to a certain color class is then:

$$P(I_{ij|c=0} \cap ... \cap I_{ij|c=C}) = \prod_{c=0}^{C} P_{ijc} \quad (2.52)$$

The probabilistic approach to pixel thresholding is very flexible because while the representation model may change from a naive linear distribution to one based on the camera response function the higher level algorithm does not need changes.
Another modification to the original algorithm is that probabilities allow for the image to be divided in a grid of probabilities, naturally generating regions of interest of a unit larger than an elemental pixel. This will be shown to be important because the region merging algorithm will have less units to operate on.
This division of the image in a grid is done by taking the average of the pixel probabilities in that grid area, thus creating a resilience to noisy images related to the grid scale and not the whole image like the simple thresholding algorithm described before. The grid scale chosen was obtained through the characteristics of the camera. The size of the features to be recognized is known, as is the altitude. Then, the lowest grid scale is going to be the smallest expected feature size plus a tolerance

## 2.8 Segmentation Labeling

Segmentation labeling or region labeling is related to disjoint sets graph theory, in the sense that it requires the grouping of $n$ distinct elements into a collection of disjoint sets[25]. In context, the elements correspond to pixels and the disjoint sets to regions of related pixels. The relation is gathered from the image segmentation.
Generally to work with disjoint sets only 3 operators are necessary:

- Make set: Creates a new set with one element of a certain segmentation class $C$.
- Merge sets: Merges sets of same class and which are neighbors.
- Find set: Returns a pointer to the current valid set.

Neighborhood in the context of the current problem is defined by the a set which has an element adjacent to the current position. This adjacency is not diagonal.
The result of the operators return a set graph with rules stated in [22]. Given $K$ sets:

1. $R_i \cap R_j = 0 \forall i, , j = 1, 2, ..., K \wedge i \neq j$
2. $P(R_i) \geq \alpha, \forall i$
3. $P(R_i \cup R_j) = 0, i \neq j$

The first condition states that disjoint sets do not share elements and thus not intersect. In practice this means the merge operator never merges sets if the pixels are from different classes according to the color segmentation.
The second condition states that there will only be regions which were subject to segmentation. In effect the background is a region but will be ignored for region labeling purposes.
The third condition states that the reunion of 2 regions is not a valid operation if $R_i \neq R_j$. This is true because after the region labeling, different regions are necessarily disjoint otherwise they would have been merged into a single set.
In algorithm 6 the problem of region labeling is solved. It is important to note that image boundaries require a special case of the algorithm but it is very similar to the general description. It is also clear that it only uses the 3 operations on the sets while scanning the image, thus it is general enough for sets that contain additional information about the region, as is the case of this work.

Algorithm 6 General Region Labeling

```
1: function Region Label(image)
2:     for y ≤ I_rows do
3:         for x ≤ I_collumns do
4:             if P(R_i) ≥ α then
5:                 current_region ← Call MakeSet(P)
6:                 if P_{x−1,y} ⇔ P_{x,y} then
7:                     left_region ← Find_Set(P_{x−1,y})
8:                     above_region ← Find_Set(P_{x,y−1})
9:                     Merge(left_region, current_region)
10:                end if
11:                if P_{x,y−1} ⇔ P_{x,y} then
12:                    Merge(above_region, current_region)
13:                end if
14:            end if
15:        end for
16:    end for
17:    for all Regions do
18:        curret_root ← GetRoot(region)
19:        if current_root ∃ K then
20:            continue
21:        else
22:            Add root to final region list
23:        end if
24:    end for
25: end function
```

Algorithm 7 is very simple but illustrates the data structure that each region holds. This information is useful for both image analysis and for region merging.
The sets resultant from the 3 operators are regions which contain:

- An identifier index.

- A class index (color).
- A rank (number of pixels/area).
- A centroid (region center of gravity in pixels).

Algorithm 7 Make Set

```
1: function Make Set(pixel)
2:     region.centroid = pixel
3:     region.parent = parent
4:     region.area = 1
5:     region.color = pixel.color
6:     Add a new region to the global list of regions
7: end function
```

The merging operator is very important because certain heuristics are proved to make the computational complexity proportional[25] to the number of merge operations. Two heuristics make this improvement possible:

- Merge to highest rank
- Graph tree compression by keeping a pointer to the highest ranking node, or as called in the algorithm, 8 to the root of the set tree.

Algorithm 8 Region Merging Operator

```
1: function Merge(region1, region2)
2:     if region1.color ⇔ region2.color) then
3:         root1 ← GetRoot(region1)
4:         root2 ← GetRoot(region2)
5:         if root1.area ≥ root2.area then
6:             root2.parent = root1.parent
7:             root1.area = root1.area +root2.area
8:             root1.centroid MergeCentroids(root1.centroid, root2.centroid
9:         else
10:            root2 assimilates root1 like above
11:        end if
12:    end if
13: end function
```

Region labeling can be visualized as a forest of graph trees where, each region merged is a tree (merge by path compression) added to the branch of the highest level of a bigger tree (merge by rank). Thus the tree tends to grow in width rather than in depth. The closer a node is to the root the least recursions are needed in the second pass over all the created regions.

## 2.9 Controller

The final validation comprises a simulated flight with all the integrated components, both external and developed. The simulated flight is controlled by a PI controller. A PI controller was chosen because it is very simple and found to be capable of performing the necessary control to automatically land.
The error signal for horizontal control is given by the distance in pixels from the center of the camera and recognized region of interest. For the vertical loop the error is given by the difference between the known height of a given feature and the measured height.

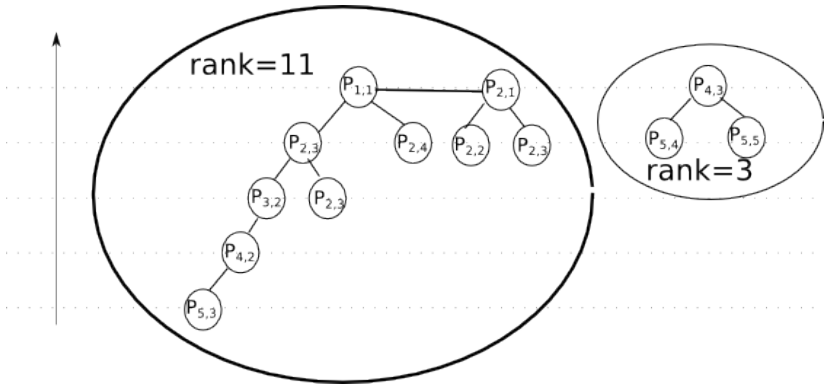



Figure 2.10: Path Compression and Rank merge

The control signal outputs an attitude(yaw, roll, pitch) PPM encoded value limited to a predefined value so that the quad-rotor does not fly in very aggressive attitudes. The altitude and body stability of the quad-rotor is done by the Arducopter module in stabilized mode, therefore assuring that the quad-rotor assumes a necessary attitude and does not enter in continuous angular motion. The basic equation of a continuous PI controller is given by 2.53 where $u(t)$, $e(t)$, $K_p$, $K_i$ represent respectively: forced control input, error function at an instant, the proportional constant and the integral constant.

$$u(t) = K_p e(t) + K_i e(t) \quad (2.53)$$

The constants are tuning parameters that are related to the system dynamics. With the analysis of the equations in 2.3 theoretically the tunning gains could be obtained. In reality, as the attitude control is done on top of a stabilization loop the dynamics of the system are not easily available without deep understanding of the Arducopter code. This deep understanding would be such an endeavor that an autopilot module could be developed in the mean time with the developer expertise gleaned from this work and from the specialty classes lectured in Universidade da Beira Interior.

With this, it is important to know what are the meaning of the constants in the context of a PI controller.

The proportional constant $K_p$ represents how important the instantaneous error should be in the system control. Effectively, it constraints how fast or slowly the error should be corrected. A high $K_p$ leads to control outputs that may produce a system forcing that overshoots the desired system state. This can lead to oscillation. On the other hand a low $K_p$ can, in extremis, not be able to control the system and on a more moderate mis-tuning, lead to a system that takes too long to converge to the desired state.

The integral constant $K_i$ represents the sum of errors over time. Doing the same analysis as in $K_p$, a high integration constant can also introduce system instability due to consistent desired state overshooting. On the other hand a very low $K_i$ negates the benefits in convergence acceleration and steady state error correction, that cannot be obtained with only the proportional part. One of the problems found with the horizontal error function is that it can lead to problems of integral term windup when descending, that is, the integral has a good influence while at higher altitudes but becomes a problem at lower altitudes because the high accumulated values are unreasonable as the motion becomes finer grained. This problem was solved by bounding the maximum integrated error component and clipping it as altitude was lowered.

A derivative term was not included due to the complexity of tuning for derivative error gain. Also derivative error gain is highly dependent on the noise of the error signal, which was not studied.
In this work the PI controller was considered a satisfactory but not adequate controller because in the traditional form the gains are constant for all operating conditions. As stated before, special system requirements exist as the quad-rotor approaches the station.
The time constant of the control loop has the lower bound in the time it takes to compute a control signal from the Kalman filter estimator and a higher bound when the Kalman filter is used as a filter to continuously processed image data.

# Chapter 3

# Hardware Concept

## 3.1 Quad-rotor Frame

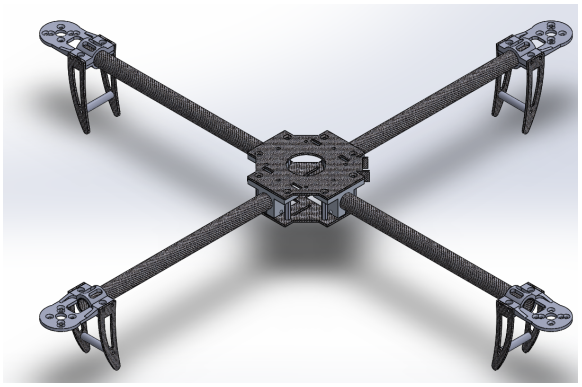


Figure 3.1: quad-rotor Frame[Grabcad]

The modeled frame is an off the shelve part. This frame model was chosen out of the others available because it was an off the shelve frame that is for sale in HobbyKing. This way it is easy to source it cheaply and in whatever volume.
This work work did not execute a physical test and because it only has 550mm of total width this is a possible problem. This forces the landing positioning to be more precise as will shown in the landing choreography description. Even so the frame is modular and it is trivial to make custom arms if necessary.

## 3.2 Battery Pack

For the conceptual work described here only the battery dimensions are represented so it is not relevant to post images.
The batteries chosen for this project are the Panasonic 18650. This batteries offer good specific power in a small form factor. The main characteristics are not very important for the current stage of development of the battery pack, with the exception of the number of batteries connected in series that make up the standard voltage of 11.1V. Each battery supports a voltage of 3.7V. This means a battery pack must have at least 3 batteries in parallel.
The batteries are packed in an enclosure suitable for the mechanism of battery switching. When the rotor-craft is airborne the battery sits horizontally on the lower plate of the quad-rotor frame. It is centered in the center of gravity, even though some deviations may be tolerated.
On the interface between the battery pack and the rotor-craft, 4 electrical power leads exist. On the frame side 2 spring loaded leads extend until they touch 2 pads on the battery pack, one for each battery polarity. A diode rectifying bridge can be installed so the polarity is corrected for the polarity sensitive circuits. The spring loaded leads may exploit the natural spring behavior of deformed metals, simplifying the construction and reducing the number of components.

Also on the interface between the battery pack and the rotor-craft is a small patch of Velcro. The role of Velcro is to create a reusable fastener to attach the battery pack to the rotor-craft. It also allows for its simple removal. The size of the Velcro patch was not determined and has to be defined empirically with the the servo and arm mounted. The Velcro patch must be strong enough to keep the battery attached during normal flight but must not require excessive force from the servo. The Velcro patch should not interfere with the power connections and should maintain them during flight.

## 3.3 Servo

The servo chosen was a generic hobby servo, SG90, and controls the position of the arm. This servo was chosen because it is light, cheap and low powered. The servo may be changed in the future if the arm needs to be actuated with more power. The position of the servo, as currently in the design, is off the center of gravity. This must be taken in to account when positioning other attached components.

## 3.4 Servo Arm

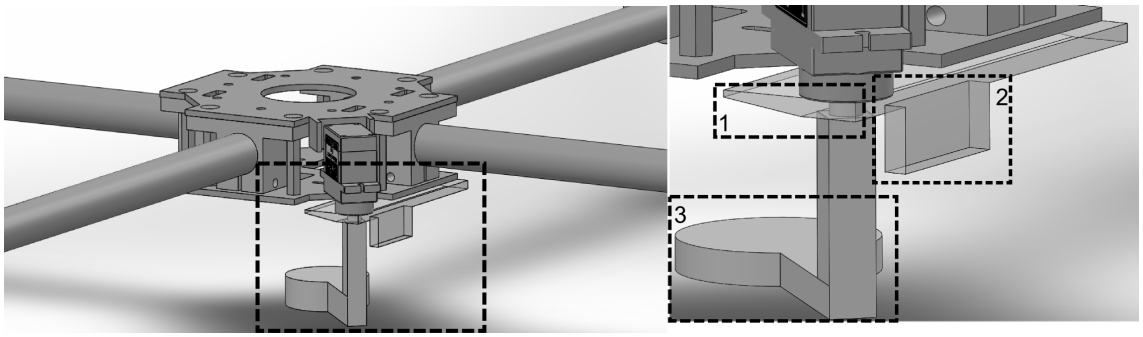



Figure 3.2: Battery switching mechanism

The arm modeled in figure 3.2 is simple and does not show the pin to actuate the switch that gets the the battery from the magazine. When committing to a final design the requirements are:

- Can be produced by a normal 3D printing machine.
- Executes its function with efficient use of the force provided by the servo.
- Low overall weight.
- Low profile when stowed during flight.

To better understand the parts included in figure 3.2, the following description applies. Do note that the arm is the component linked with the servo:

1. A wedge shaped blade that peels the Velcro link between the quad-rotor frame and the battery pack(Velcro not represented). The best way to break the connection of a Velcro fastener is through peeling[26].
2. A tab that provides additional shearing force for the unfastening of the battery. It also serves as a guide for the trajectory of the battery before falling. This tab is not essential and may be eliminated to reduce weight if it does not significantly improve the removal of the battery.

3. The camera is positioned so that when the battery is being ejected the arm rotates it out of the path of the new battery when the switching happens. When in flight the arm rotate the camera back to the center of gravity position.

While the quad-rotor is flying the arm must not be protuberant, meaning it must not have any part outside the bottom plate of the frame. This is important so that it interferes the least possible with the aerodynamics and stability of the quad-rotor.

## 3.5 Station

The main design criteria of the station is related to the mission. According to the constraints of the mission it was found that a large area flat surface would be a good support for an imprinted machine recognizable pattern. In figure 3.3 the pattern was not included because it would make the geometric characteristics of the station less clear.

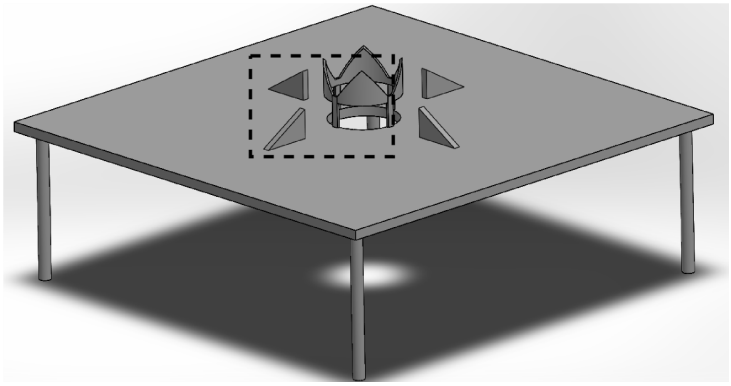


Figure 3.3: Perspective and region of interest of the Station Model

In the center of the flat plate there is a geometric feature resembling 4 ridges, that position the arms of the rotor-craft as it approaches the resting position for battery switch. This feature is better understood when observing the full choreography of battery switching. Also through the center of the station there is a hole where the battery magazine is located.

## 3.6 Battery Pack Magazine

The battery pack magazine is detachable from the station but it is intimately related to it.
The battery pack magazine was inspired in ammunition magazines. Looking at figure 3.5 the concept can be understood:

- Stacking of batteries (A) and dispensing one object at a time is done with only passive mechanisms. They are also refillable and reusable. Specifically there is a casing where the battery packs are stacked.

- On the end where the battery pack is dispensed there is a shutter(not represented) which unlocks when a switch toggles. The shutter opens enough time for one battery pack to pass, locking itself until another toggle activation. The shutter is spring loaded. The spring is loaded manually when the battery pack is loaded by an operator(not represented). The toggle button and spring loading mechanism is analogous to the ones found on photographic cameras.

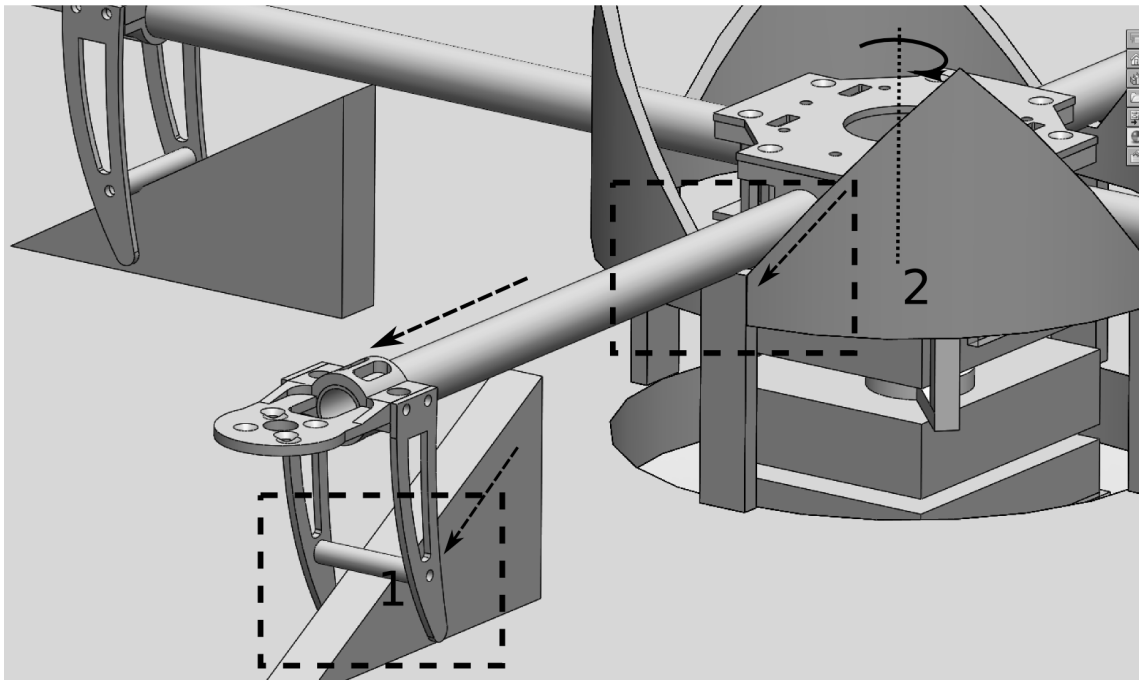



Figure 3.4: Aligning Mechanism

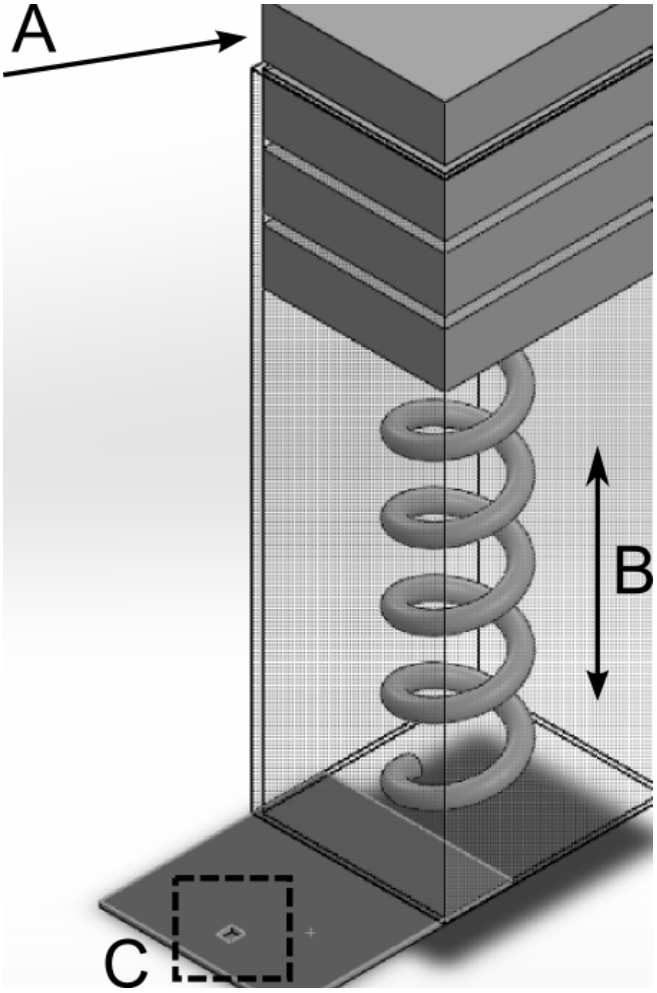



Figure 3.5: Battery Pack Magazine

- On the end where batteries are loaded there is a cap. The cap is where the spring that pushes the battery packs is supported. Like automatic rifle magazines it works as sliding window with a braking clip (C).

- The batteries are always under force from the spring (B) mounted on top of the sliding cap. That spring is responsible for the ejection of the battery. Upon ejection, the Velcro on the battery pack will secure it in place once it contacts the quad-rotor frame.

## 3.7 Battery Switching Choreography

In the approach phase, but close to touchdown, it is possible that some disturbances make the orientation and positioning of the rotor-craft less than ideal. As such, the static platform has a convolved ridge pattern that guides the arms to a more precise position. This can be seen in figure 3.4, region of interest(ROI) 1 and 2.

1. The angular positioning is ensured by the fact the the landing leg which first touches the platform, slides tangent to the the edges of a ridge(ROI 1). The edges will then push the leg and rotate the whole quad-rotor body. This can happen on any leg depending on the way the quad-rotor is misaligned. The interface between the leg and the edge should provide low friction so that the sliding movement is possible.

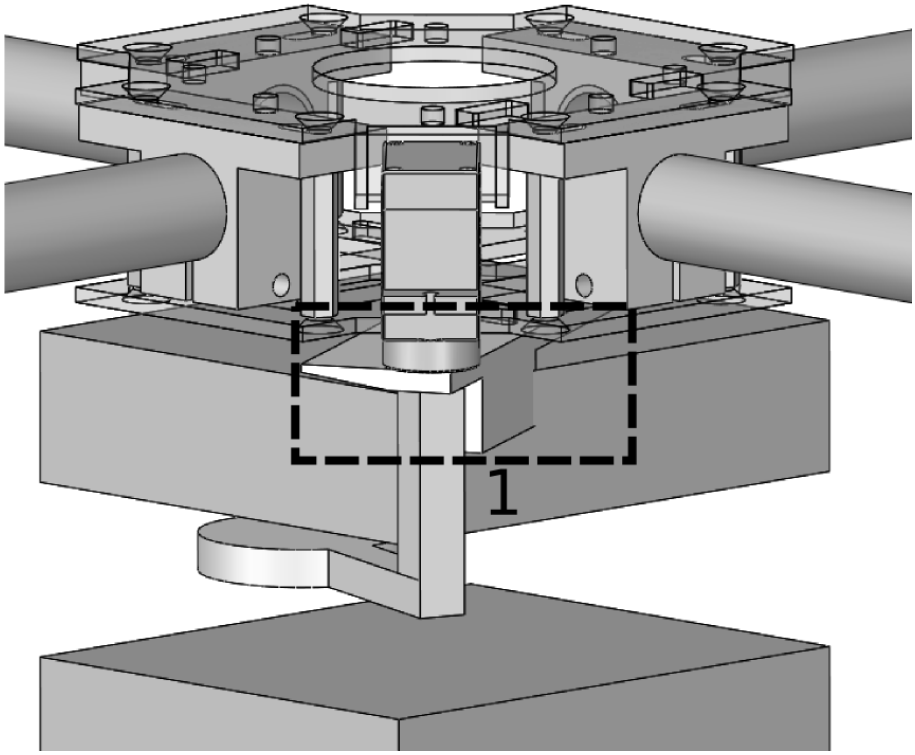



Figure 3.6: Tab ejecting the battery

1. The linear positioning is achieved through the sliding of the landing gear legs on a slope. The slope is a thin triangular prism thus it is only usable if the quad-rotor is already positioned in the correct direction. This is possible because a mis-positioned leg will touch the prism and exert weight on it.

2. The final touchdown happens when the arms can't slide further. At this point the quad-rotor is aligned within the tolerance. The only point of failure is the existence of some friction which may leave one arm slightly raised. A relevant feature in the choreography is the initial position of the servo arm.

3. When the control system decides the landing has been successful, the servo arm is rotated to peel the battery away. As described before: A tab pushes the battery imposing a shearing force, a wedge peels the Velcro apart and the camera support uncovers the battery pack. With the depleted battery, the wedge, the tab, and camera support out of the way, in their final position the new battery is ready to be pushed into the vacant space.

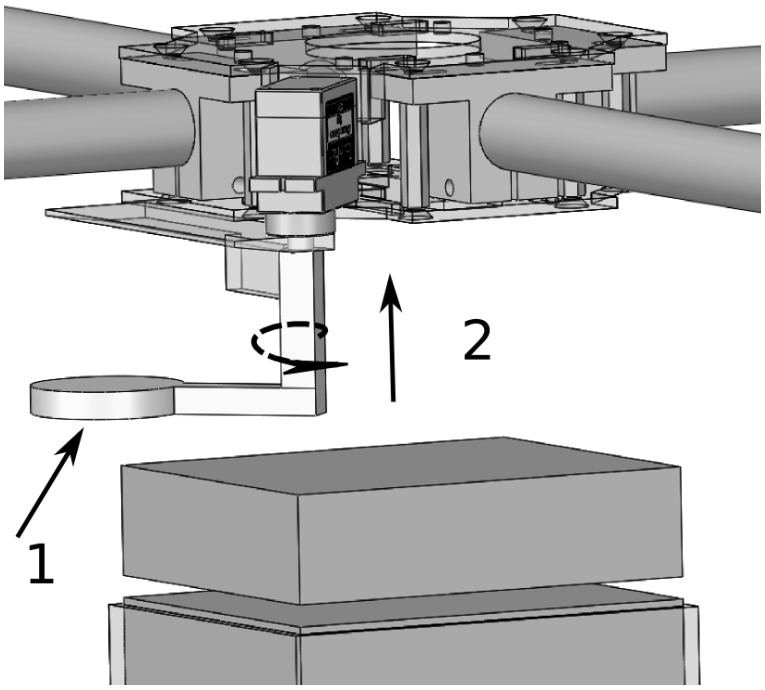



Figure 3.7: Magazine Battery Popping

1. When the trigger of the magazine is hit a new battery is pushed. As the camera fixture is at the most outward position of the arm rotation it is a suitable candidate to be the trigger of the battery pushing mechanism.

2. On the the final step the camera goes back to the correct position making a full rotation. This is a requirement because the tab would take out the battery if it reversed the course made. The servo proposed does not rotate the full 360 degrees by default but it can be adapted to do so. Upon repositioning of the camera fixture the general mission can proceed with the normal flight planning routines.

## 3.8 Target design

While reviewing the available literature the most developed machine vision algorithms were based on general feature detection for positioning and orientation. Unfortunately this algorithms are complex and computational demanding[20, 27, 18, 28]. It became clear that by controlling the pattern of the target, much simpler and effective machine vision primitives could be used, specifically the application of binary image segmentation. This operations allow the detection of shapes

expending low computational power. The low computational power requirement is crucial for low latency, fast response positioning.
Fast and effective marker recognition is highly valued in robotic football and as such the main inspiration for the target design was drawn from scientific papers related to the subject.
In [21] a study of targets is done. In it is evaluated the shape and the pattern of the target:

> For detection, the simplest approach to take for a single patch is to use a simple regular geometric shape of a single color. Detection in the vision system can be carried out on a binary or multi-class threshold image from which connected regions of common color class can be extracted.
>
> [...]The next variable to determine for is patch shape. We chose circles, because they guarantee rotational invariance, and analytical corrections for the projective distortions of their image centroids are known. In addition, they are compact, minimizing the length of the border with other regions, where thresholding is most difficult.

These 2 paragraphs justify most of the computer vision techniques used in this work. The only difference in the realization of this work, in respect to machine vision, to the above sentences is that the projective distortions were not corrected analytically but through additional heuristics.

## 3.9 Target layout

As shown before in [21] patch size is studied. There, it is concluded that patch sizes do not interfere with position accuracy, and are only important for reliable detection of the whole region. For this work, it means that the patch size should be only enough so the visual references can be detected in the transition from coarse navigation to visual navigation. Thus 2 levels of visual accuracy were conceived:

- One level for the whole station visual detection.
- A smaller scale level for visual marker redundancy.

The higher level feature is the station itself marked in a non natural occurring color. The larger the station face area the higher the distance at which it can be detected.
The smaller scale color features are not detectable from high distances and may even be unreliable because environmental color noise may introduce false positives. On the other hand when the distance from the camera to the target gets small(how small depends on the camera focal length) this colored features are seen completely embedded in the station background thus almost eliminating the possibility of false positives from environmental features. A way to understand this color noise would be to imagine how from a afar, in a continental landscape, most of the image would contain vegetation features which exist in a wide gamut of green(this is why the eye is most sensitive to green gamut). Tracking a green feature would be useless, and most likely would give an image centroid in the center. On the other hand if this green color patch is located straight in the middle of the target pattern, an inherent high likelihood of a correct color recognition arises. A synthesized image like the one in figure 3.8 illustrates this reasoning. In it, the image from the higher altitude has an histogram with green values approximately in the middle of the channel range. The pure green color of the target is not even visible at the overall scale. As the image gets close the background green of the vegetation almost completely dominates the green channel histogram. Finally directly above the station the histogram only has the target's color and the null intensity values.

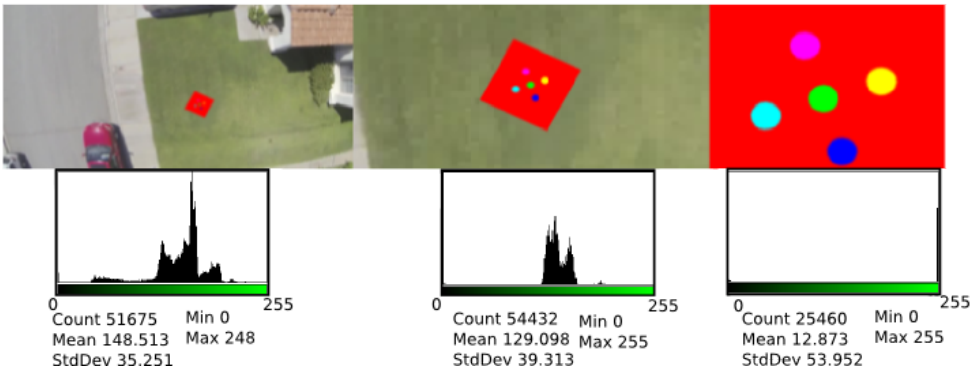



Figure 3.8: Green channel histogram

The pattern selected was one similar to the dice representation of 5. This differs from the butterfly pattern described in [21] because for our case there is not a special need for target compactness, plus the reconstruction of the centroid given the visibility of only one color becomes easier.

## 3.10 Target Colors

The colors selected for the target were obtained by additive pure RGB colors. E.g, purple's RGB coordinates are given by:

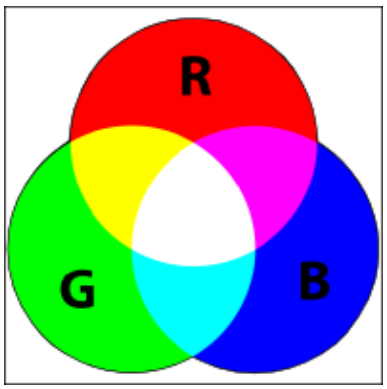



Figure 3.9: Additive RGB color primaries

$$RGB_{purple} = \begin{bmatrix} 255 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 255 \end{bmatrix} = \begin{bmatrix} 255 \\ 0 \\ 255 \end{bmatrix}$$

The intent is that these colors have greatest distance between them and thus allowed for a greater tolerance without overlapping values. Unfortunately to get pixel values on this range the exposure must be almost perfect and in practice, the channel values will have intensities which are much closer to halfway the range than the maximum as was proposed. An intuitive way to think about a single channel intensity not on each extreme is a gray scale of an image channel. In figure 3.11 a real camera image red histogram where there are 2 peaks and none are on the 255 range illustrates this low dynamic range. The right narrower peak is actually not from the red of the target but of the white of the paper, which also includes a high intensity red channel. It is narrower because the camera automatically calibrates itself for correct white.

An argument found against using primary additive colors was that these colors are actually per-

ceptually similar. Considering that for the camera used, the manufacturer says it calibrates the cameras against a pattern called Gretag Macbeth[1] it makes sense choosing the patches colors from this calibration model. This was discovered too late in the development of this work but provides a very good possible improvement, because Gretag Macbeth calibration patterns provide mappings between Cie XY color space colors and manufacturer gamma corrected RGB. Thus, choosing colors from this pattern would theoretically allow for reading calibrated colors from an off the shelve camera.

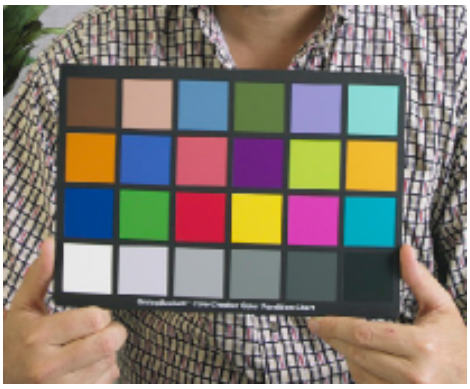


Figure 3.10: Gretag MacBeth calibration pattern

Red was chosen as the target's color because it is a color that doesn't naturally occur so frequently in nature in great areas, thus making a good coarse scale visual marker.

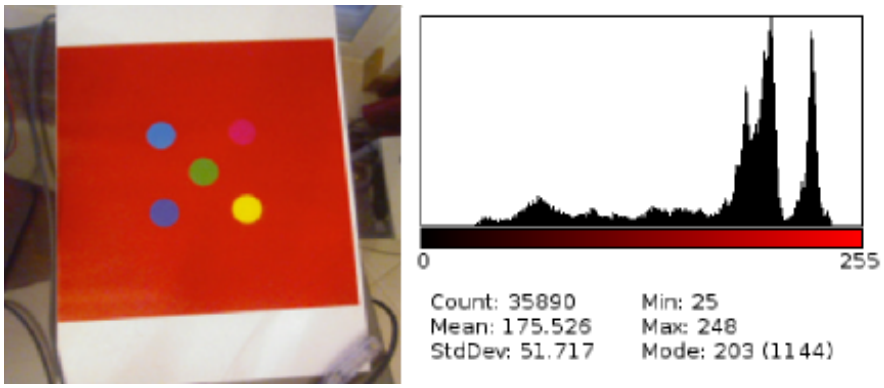



Figure 3.11: Red histogram of camera image of the target

## 3.11 Serial Connection - Physical Layer

The communication between the Arducopter component can be achieved either through USB physical interface or through a simple Serial interface. The Serial interface comprises 4 pins as in Figure 3.12:

The component developed for this work communicates through the TX and RX pins. The elegance of a serial connection is that it is a very simple widely used interface. As such the developed software component in the current stage of the project does not need to be airborne, as there is an electronic component called XBee that can provide a wireless carrier for a serial signal.

[1] According to RightLight technology description which the camera C500 claims to have: http://www.logitech.com/lang/pdf/ib-rightlight_EN.pdf

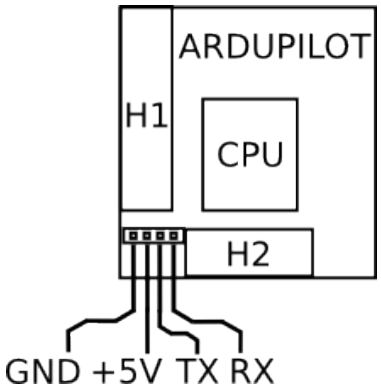



Figure 3.12: Arducopter Serial Pins

The component was programmed to work on a Portable Operating System Interface(POSIX) operating system. As such to enable successful communication with Arducopter, certain information needs to be given to the operating system regarding the type of data expected from such a low level interface. Most of the settings necessary are orders to make the operating system disable all possible pre-processing of data before it is handled to the requesting application. The other 2 important settings are Baud rate, which is a unit of electrical impulses per second, and port device name. Both are different if the Serial connection is passed through USB or RS232. The used settings in this work are in table 3.1:

| | USB | RS232 |
|---|---|---|
| Baud Rate | 115200 | variable |
| Port | /dev/ttyACM0 | /dev/ttyS0 |

Table 3.1: Serial Connection Basic Settings

Serial functionality is accessible through a C++ abstract class. This way different physical interfaces can be developed while imposing no constraint on software processing of the MAVLink messages.

# Chapter 4

# Implementation and Validation

## 4.1 Target Generator Software

To easily iterate through pattern designs a command line tool was produced that generates vector graphics of the station pattern. This vectorial representation in real world units of the target pattern can be as accurate as a printer can reproduce it. The position of the color patches are mathematically defined to match the "5 in a dice" pattern. The colors can also be defined in any color system, provided the printer software can interpret it. To test the colors of the Gretag MacBeth calibration only a new pattern would have to generated with the new colors and printed with no manual work involved. The way the visual station tracking software was designed also allows for the change of patch colors by only changing a compile time constants.

## 4.2 Component Communication

To be able to execute the landing there must be means to obtain sensor information and capability to direct a portion of the flight. This means are provided by the Arducopter FDCS through a protocol called MAVLink. Camera information and processing is done by a developed target recognition software library. Additional control is sent to the FDCS to execute the landing.

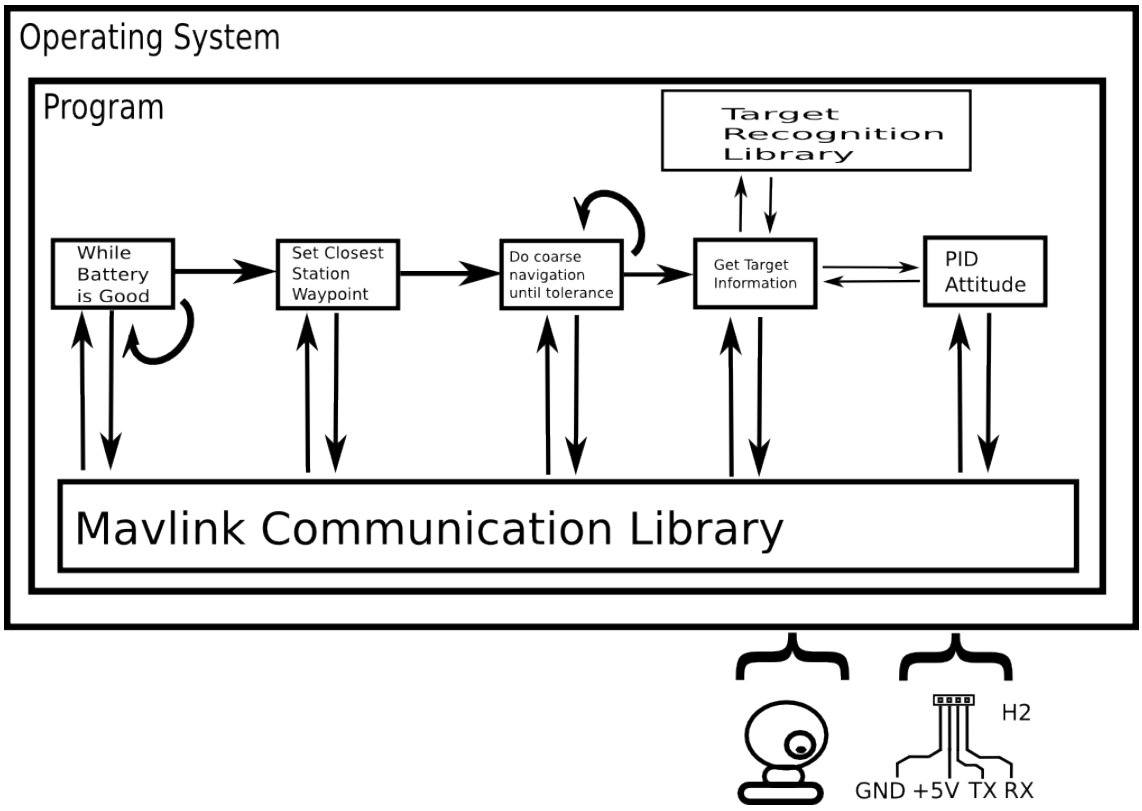



Figure 4.1: Software Layout

## 4.3 Design by Contract

One of the most important software development patterns used throughout the execution of this work was "Design By Contract". This pattern establishes that C++ classes that execute certain functionality must expose it through a standard, or as implied by the name, a contract. This contract is set by a parent class with nothing specified but the methods that a class must implement to be able to be a child class. An example from the Serial Communication C++ class is that other parts of the software only expect to receive and send data through methods called SendMessage and ReceiveMessage. The rest of the software is then isolated of the exact implementation of how the messages are sent. This allows for the replacement of a serial connection with e.g a TCP connection without changing any code besides that of the implementation of the TCP connection specification.

## 4.4 MAVLink Communication Library

MAVLink is a communication protocol for small unmanned vehicles. According to the official website[sic]:

> MAVLink is a very lightweight, header-only message marshalling library for micro air vehicles.
>
> It can pack C-structs over serial channels with high effiency and send these packets to the ground control station. It is extensively tested on the PX4, PIXHAWK, APM and Parrot AR.Drone platforms and serves there as communication backbone for the MCU/IMU communication as well as for Linux interprocess and ground link communication.
>
> MAVLink was first released early 2009 by Lorenz Meier under LGPL license.

MAVLink is a very important piece of the developed work because it provides the means of communication with Arducopter. In the website quotation it says it is a lightweight library. The offered library comprises the very basic data packing blocks to communicate single messages, but to achieve meaningful communication often several messages defined by this protocol need to be exchanged with an interlocutor.
The object of this section is then to describe the library produced to achieve the actions necessary to control the quad-rotor.

### 4.4.1 Accessible functionality

One way to understand the basic actions is looking at figure 4.1. There it is seen that almost every part of the landing orchestration needs to interface with the MAVLink library. In more detail the public methods are:

- Set Flight Mode
- Override RC commands
- Read and Write EEPROM system constants
- Read and Write Waypoints
- Request sensor calibration and sanity checks

- Arm motors
- Request Real Time telemetry

There is a very mature library that achieves most of this work but is written in Python, an interpreted language. As such the integration with a C++ program would be not very elegant nor flexible. It is reckoned that the message handling would not be the bottleneck of the operation but given communication is such a critical part of this work it could not be left outside control. Besides in an embedded environment a complex interpreter like python would consume resources which are dear in the air. This work also aims at creating intellectual property developed independently and controlled by this author.

Generally MAVLink communication with an interlocutor can be thought of as transactional, meaning that several steps are needed for an action to be executed. If all the steps are not completed, including acknowledgement by the intervening parties the communications are without effect.

The message exchange and processing occurs in another thread of execution, this way the user of the library does not need to block the execution of the main program while messages are being received. Thus the developed library runs asynchronously until data needs to be exchanged with the main execution thread.

### 4.4.2 General MAVLink library layout

The library is designed to use a communication interface that offers the methods described in section 3.11. To start a connection with Arducopter a special message must be sent periodically. This is called the heartbeat message. All parties in a communication must issue heartbeat messages. If these messages are not received after some period of time the communication and any transaction is dropped. The heartbeat message has the role of not only signaling a party in the communication but also includes vital information about the current flight mode set and general identification data.

The library organizes individual messages, both received and sent, in state-machines. This is required because multiple steps are needed to commit an action. This multiple steps can formalized as state machines of the transaction being communicated. Algorithm 9 describes the implementation

Algorithm 9 MAVLink Library Event Loop

```
1: while Communication Interface is Valid do
2:     Receive Raw Data
3:     Process Recognized Messages
4:     Send Necessary Messages
5: end while
```

The state machines are updated according to the expectancy or reception of specific messages. They can also be updated if the user of a library requests one of the public actions. The user sets the initial state and may wait for a reset of the state by the library. When the action is reset either the action was fully executed or failed. Either way the library user, with the exception of a Radio Control(RC) override, has information about the success of the requested action. The detailed description of the developed library would be too technical and tedious considering it is over 1500 lines of code.

As such only one example of the of waypoint setting algorithm is described in Algorithm 10:

Algorithm 10 Send New Waypoint List

1: Wait for any pending actions to finish.
2: Put $W_n$ new waypoints in a buffer.
3: Action state machine ← $SET\_NEW\_WAYPOINT$.
4: Main Thread may wait for reset.
5: Event loop detects a new requested action.
6: Waypoint state machine ← START.
7: Send Waypoint Count
8: Waypoint state machine ← SENT_WAYPOINT_COUNT.
9: while $W_i \leq W_n$ OR abnormal do
10: Parse new waypoint request
11: Waypoint state machine ← WAYPOINT_REQUEST
12: Process Waypoint and send.
13: Waypoint state machine ← SENDING_WAYPOINT
14: end while
15: if $successful$ then
16: Waypoint state machine ← WAYPOINT_SENT_ACK
17: else
18: Waypoint state machine ← WAYPOINT_SENT_FAILED
19: end if
20: Action state machine ← RESET.

### 4.4.3 MAVLink Library Testing

An important characteristic of the developed library is that, it too was validated. The framework used was googletest. The automated tests generally consist in sending information to Arducopter and reading them back. As the methods to do both are independent, if the read results are the same that were sent, there is a high confidence the library is working as intended. This way if some future change in the library happens and causes the the response to not be the expected, a validation error is generated and has to be fixed. The most advanced features were tested against a simulator of Arducopter as described in section 4.5. The features that didn't require a quad-rotor to be airborne were tested against a real Arducopter board connected by USB.

## 4.5 Flight Simulator

For this work a visual flight simulator was developed. Although the development itself was brief and consisted mostly of editing and joining pre-existing code, it enabled a new workflow to allow 3D objects created in Solidworks to be used in a flight simulator. A similar functionality could be obtained by using another available flight simulator called Gazebo. Gazebo is a very advanced robotics simulator used by other works for simulation of virtual environment[28]. Unfortunately the author is not aware of interfaces between the Arducopter Software in the Loop functionality and the Gazebo software.
The developed simulator workflow was designed so as to provide the least intrusive integration with the visual control software. That is, the system should not be able to distinguish between a real environment and a synthetic one. There were several data bridges that enabled this claim.

### 4.5.1 Simulation Layout

As can be seen in figure 4.2 the simulation layout shows a similar setup to the normal system layout. The only differences happen at the operating system level, with 2 data bridges: socat, and gst-v4l2loopback.

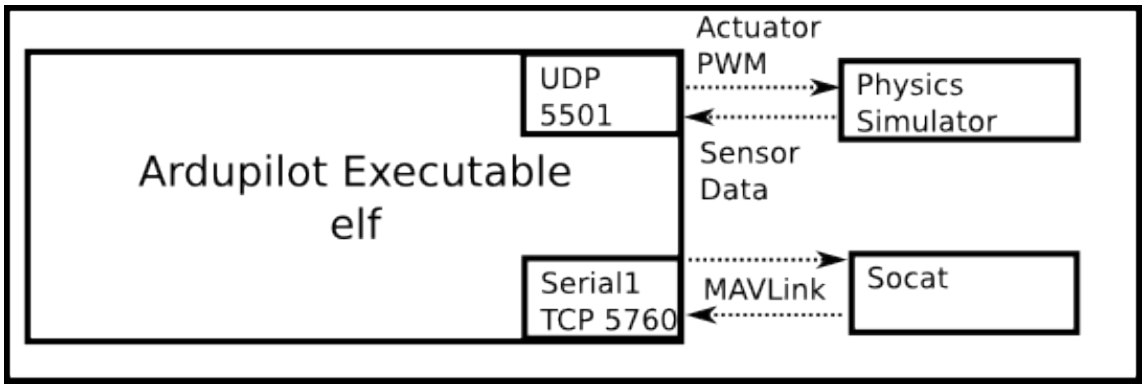



Figure 4.2: Simulation Experimental Setup

The flight simulator used was developed from code of 2 parts: The visual simulation and the physics simulation. Both parts are integrated into one single code in the Python language.

### 4.5.2 3D environment

In this work a method for simulation of 3D objects designed in Solidworks was developed. Because texture is important for this work the model has to include one. The texture applied to the 3D model was the one of the target. The visual simulator developed accepts a file of OBJ type. As far as the author is aware this is the only open 3D format with texture support that Solidworks exports to. Others are binary files or do not support texturing, e.g the STL format.
OBJ files simply store the vertices's position along with corresponding texture coordinates in a human readable format. This way it can be edited manually. Although manually editing mostly numerical data is not useful, it provides access to change the texture files or materials of the model if needed.
One of the problems encountered with the OBJ file format, and even with the 3D composer, OpenGL, is that there is no concept of absolute units. All the space is described in homogeneous coordinates like the ones described in section 2.5.1. This forces manual adjustments in the viewing frustum of the camera so that the homogeneous units would match the correct scale. This manual adjustments were done considering the known dimensions of the CAD model.
Another problem in the exported model workflow is that the texture coordinates do not always exactly match the ones set on Solidworks. In figure 4.3 it can be seen that the green patch is in the top face corners instead of in the middle, inside the landing supports. Because of this the green color patch is ignored throughout the validations of this work.
Although it can be hard to perceive in figure 4.3, the colors rendered on both perspectives are not exactly the same. This is due to the fact that the lighting is also simulated, and the position and type of the light spots affect the rendered objects and textures in different ways. Thus the model and accompanying texture can be used to simulate lighting effects due to changes of the relative position of the quad-rotor to the target. One thing that is not simulated is the shadow cast by the quad-rotor, because the quad-rotor 3D model was not included in the simulation.
The light source setup impinges on the scene 2 types of lighting coming from a single point also described in homogeneous coordinates: diffuse and ambient.
Diffuse lighting defines how much of the intensity of the light source radiates in all directions when reflected by a surface.
Ambient lighting represents a base line lighting illuminating every object the same way.
There is no actual defined relation between the the parameters and physical units of intensity,

but only a perceptual relationship. This is because the OpenGL graphics library does not aim to control the output so as to be a physically correct representation. This lack of physical accuracy in the color and light rendering was not deemed crucial because the simulation is not intended to replace future real world tests. Also, correct simulation of lighting would have to be done with Ray-tracing technology which is not adequate for real time.

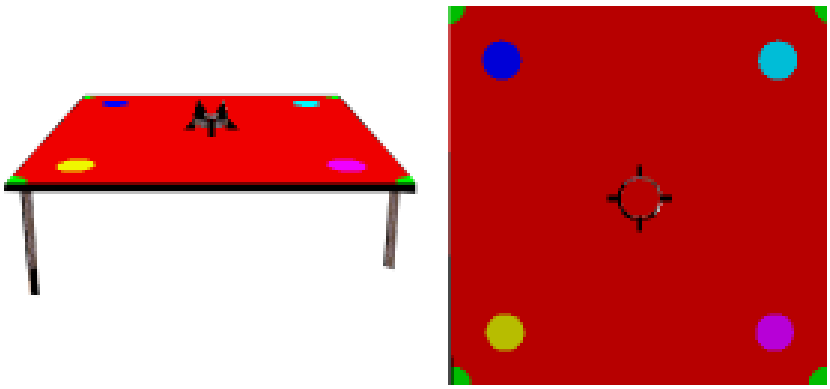


Figure 4.3: 2 Views From the Visual Simulator

The rendered scene is shown in a window. This window is then captured by gstreamer to simulate a camera.

### 4.5.3 Camera Simulation

In the Linux operating system there is a suite of utilities called gstreamer. The basic logic consists of media elements. Each element has pads which can be either inputs or outputs of the element. A string of elements connected to each other through their pads creates a pipeline. The beginning of a pipeline is called a source and the end is a sink. The analogy with the electronics signal pipelines is intended.

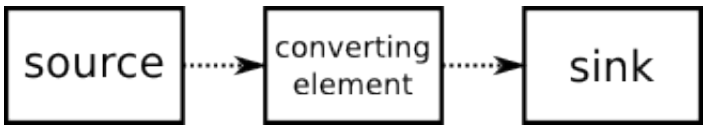



Figure 4.4: Generic gstreamer pipeline

Two types of validations were made that required different camera sources. Therefore, the sink element will be described first because it is an element that never changes regardless of the validation being done.

#### 4.5.3.1 sink

The sink element used is called v4l2sink and allows media coming from a gstreamer pipeline to be offered as a "fake" camera device. This device is created at the operating system level. This way, when the operating system is requested with images from a camera, it returns the pipelined media just as if it came from a physically connected device. This is what enables the same component software to be run either on a synthetic or real visual environment.

#### 4.5.3.2 source

The gstreamer media suite includes several default sources and elements. The 2 sources used were:

- filesrc which enables the loading of any file to the media pipeline. This was used to load target pictures described in the subsection 4.7.

- ximagesrc is used to capture the images from the flight simulator window. This is not very elegant but consistently works. It is also independent of the actual rendering taking place in the window, also ensuring independence from the view and the actual simulation.

The source elements normally have to be passed through intermediate elements that formats the media to be compatible with v4l2sink.

### 4.5.4 Serial Port Simulation

The developed component expects a connection to Arducopter through a serial interface. The problem is that Arducopter running in Software-In-The-Loop(SITL) accepts a connection through the TCP protocol. This mismatch was bridged in a completely transparent way for both the component and Arducopter SITL. The solution found was to use a POSIX utility called socat. According to the manual:

> Socat is a command line based utility that establishes two bidirectional byte streams and transfers data between them. Because the streams can be constructed from a large set of different types of data sinks and sources (see address types), and because lots of address options may be applied to the streams, socat can be used for many different purposes.

It is then clear that socat has a similar architecture to gstreamer, where the concept of sinks and sources of data streams is also used. For the simulation, the concept of source and sink was stretched because the data is streamed bidirectionally as shown in figures 4.2 and 4.5.

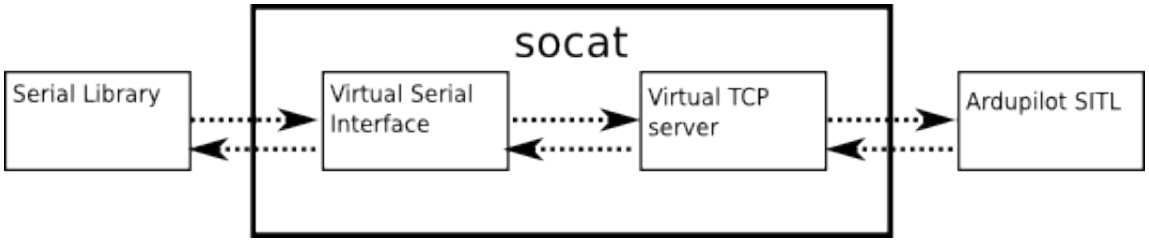



Figure 4.5: Socat bridging

### 4.5.5 Arducopter Software in the Loop

As mentioned, the Arducopter source code is freely available and distributed. As such technical utilities besides the flying software are available. One of those utilities is a method to build this source code targeting a normal personal computer instead of the Arducopter physical board. The name given to Arducopter built this way is Software In the Loop(SITL). The important thing to note is that the code base is the same only the build process is different, specifically the tool chain and the compiler switches that enable code paths that allow for the sensors and actuators to be simulated.
This build mode is so important and reproduces so faithfully the behavior of the real code that Arducopter's continuous code integration and update is tested first in this mode, only afterwards moving to the real board.

The behavior of Arducopter SITL resembles so much to the real one that before achieving flight, pre-flight configuration commands must be transmitted with the parameters of the simulated quadrotor. This parameters are then saved in a a binary file that has the same representation as the on board EEPROM memory. This way, as in the real board, the following settings are stored from flight to flight:

- RC channels minimum, maximum and trim offset values.
- Frame type.
- Compass offset.
- Compass, GPS and barometer enabling.
- Compass, GPS and barometer noise.

Additionally the sequence of preflight commands to achieve autonomous station departure are:

1. Sensor calibration before each flight. Without this step Arducopter cannot set reference values for the sensor data it is receiving.
2. As no actual RC remote is used in the simulation all the radio channels must be overridden with neutral values so that undefined ones are not used when the flight starts.
3. With the controls neutral it is safe to set the Arducopter in Stabilized mode. In stabilized mode Arducopter assures the attitude stability of the Arducopter and only leaves a neutral attitude when an RC attitude command is inputted.
4. Having all this steps done, the motors can be armed and the throttle channel overridden.
5. A fully autonomous attitude stabilized take-off takes place.

An important technical detail about the RC command overrides is that they are encoded in a binary format called Pulse Position Modulation(PPM). PPM encodes a PWM signal through the predefined time between different channels. The timing difference between each pulse acts like a duty cycle for that channel. In Arducopter values range from 1000 to 2000 with neutral in 1500. Although, in the tested simulator, the throttle trim of 1500 roughly corresponds to constant height hovering as explained in equation 2.10.

## 4.6 Kalman Filter Implementation Validation

To validate the Kalman filter a simplified test case was created. One part of the test case consisted in creating a ground truth set of data. This ground truth data was generated with a logarithmic spiral so as to simulate a non linear position variation of a given measurement. A 2D case was used to simplify the test case.

$$f(\alpha) = \left\{ \begin{array}{c} x = a\cos(\alpha)e^{b\alpha} \\ y = a\cos(\alpha)e^{b\alpha} \\ z = h \end{array} \right\}, \alpha \in [0, 4\pi] \quad (4.1)$$

To guarantee that the Kalman filter implementation would be exactly the same validated, the routine that returns the visually analyzed data was transparently overridden through an interface

method. This interface method generates random additional points reflecting the positions of the various color patches. The random number for these positions follow the distribution:

$$\mathcal{U}(P_i - \frac{R_x}{30}, P_i + \frac{R_y}{30}) \quad (4.2)$$

The uniform distributed noise is slightly inadequate for the Kalman filter because Kalman filter assumes normally distributed noise, but in a real case it is not possible for a pixel value to be outside the screen. $R_x$ and $R_y$ represent the number of pixels in each direction. 30 was an arbitrarily chosen value and in total represents a range of 60 pixels.

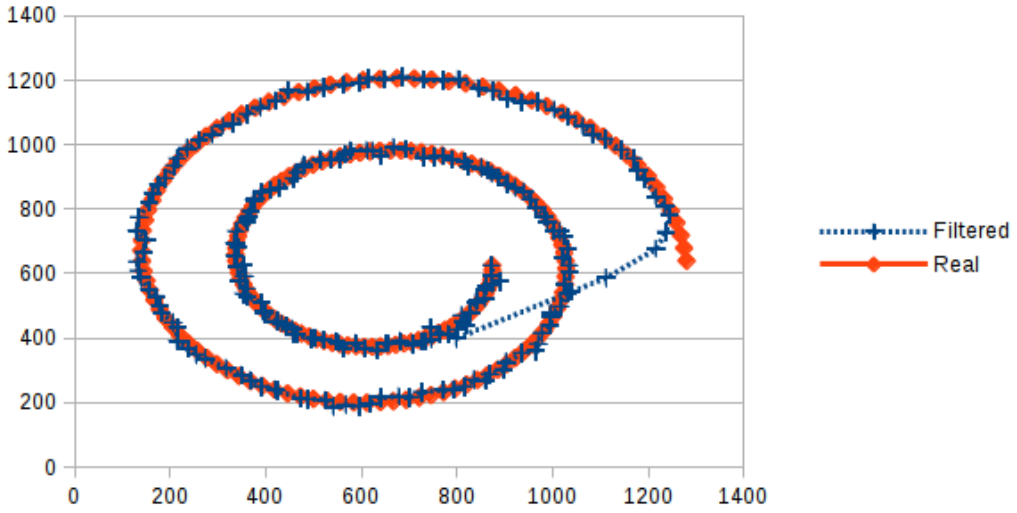



Figure 4.6: Ground Truth and Filtered Result

The initial data of the Kalman Filter assumes the minimum error possible to start the convergence, namely half the width and height position. In 4.6 and 4.7 it is possible to see that the start of the filtered results come from the center of the image and converge to the ground truth in about 5 predictions. The signal passed to the filter is too noisy to be included. It is interesting to note that even though the signal is very noisy the discovered covariance matrices allow for a resulting filtered signal with small lag (small baseline error and oscillation) $\mu = 15, \sigma = 7.30$. For the tested simulation the pixel space had a dimension of 1280x1280.
Considering that a perfect system and observation model would originate null innovation (white noise is suppressed) the base line noise average is possibly due to wrong system model and the standard deviation for wrong covariance matrices. It is also remarkable that even though each correction had uniform error with very high deviation there was no need for a low pass filter to clean outliers.

## 4.7 Target Recognition Validation

The target recognition validation is based on getting the centroids of the various color patches in the expected positions. Again the the Googletest test framework was employed.
The test sets the most simple task to the target recognition library, which is to present to the camera the original image of the target pattern shown in figure 4.8. This setup is useful to assess

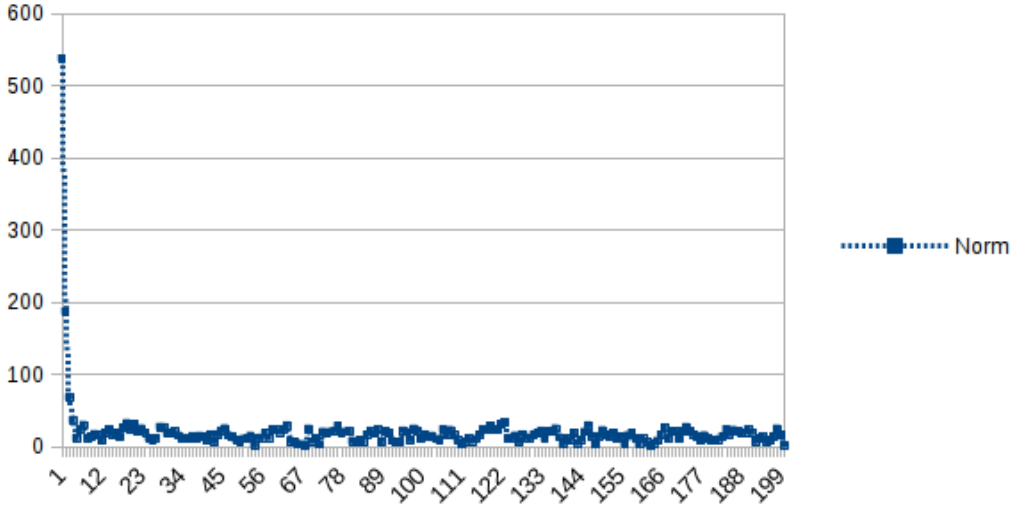



Figure 4.7: Kalman Filter Error

the raw performance of the algorithms without any interference of lighting or perspective distortion.

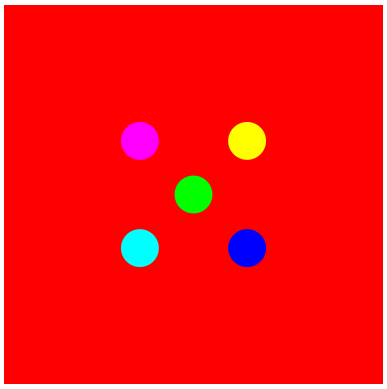


Figure 4.8: Target Original Image

The validated parameters are displayed in table 4.1:
The recognized values are always off by one pixel. The reason for this behavior was not verified but likely 2 factors originated this result:

- The original target was a vectorized image with units in millimeters. When scaling the image to raster pixels a rounding error may occur.
- The algorithm to calculate centroids has an integer division operation. Integer division in C++ returns truncated results.

In total the error in each of the markers for each axis was 1 pixel. Considering the image fed through the camera has 640x640 resolution, that gives a relative error of $1,5625E-3$. This value is insignificant considering the noise and uncontrolled factors of the real world image signal.

## 4.8 Validation Results

The final validation comprises a simulated flight with all the integrated components, both external and developed. The simulated flight is controlled by a PI controller. A PI controller was chosen

| Test | Measured | Expected |
|---|---|---|
| Region Count | 5 | 5 |
| Number of Regions Per Color | 1 | 1 |
| Station Background (Red) Position (X,Y) | 319,319 | 320,320 |
| Upper Left Patch (Purple) Position (X,Y) | 228,228 | 229,229 |
| Lower Left Patch (Cyan) Position (X,Y) | 228,409 | 229,410 |
| Upper Right Patch (Yellow) Position (X,Y) | 409,228 | 410,229 |
| Lower Right Patch Position (Blue) Position (X,Y) | 409,409 | 410,410 |

Table 4.1: Target Recognition Validation

because it is very simple and capable of performing the necessary control to automatically land. In figure 4.9 a graphic of the position error as measured by the target recognition library is presented:

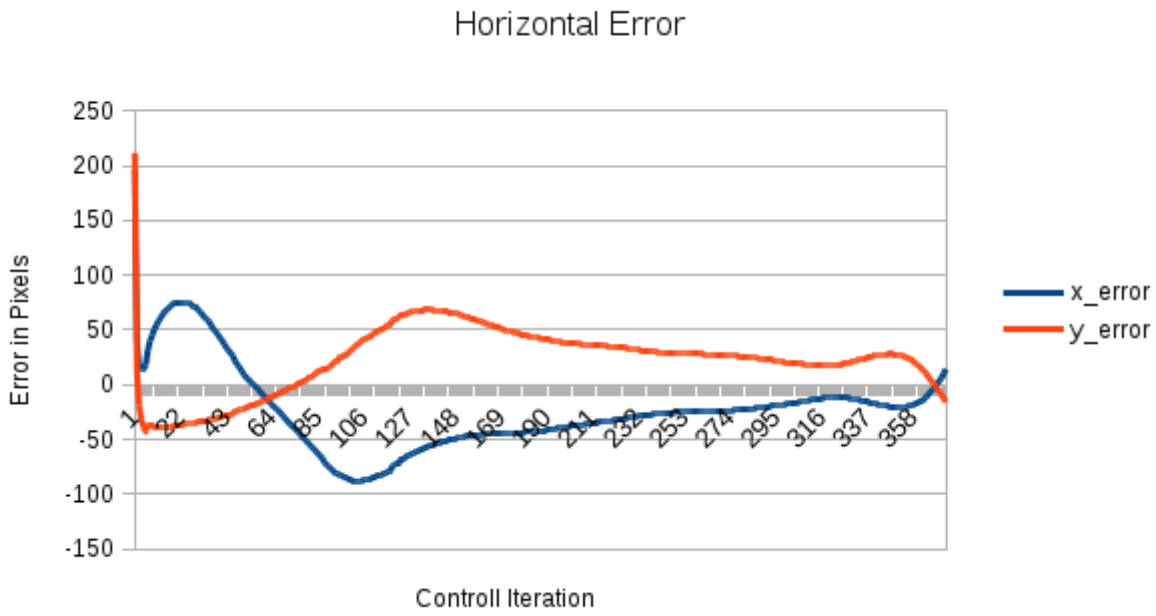



Figure 4.9: Horizontal position plot

- The error is in pixels and as the quad-rotor approaches the target the real precision of the system improves. This happens because the area of a square pixel in real world coordinates increases as $(\tan(f_V/2)z)^2$, where z is the altitude and $f_V$ is a constant field of view. The precision improves because the image centroids are scale invariant and the relative error decreases.
- The effect of error amplification by perspective distortions is such that without the gyroscopic image stabilization the controller cannot finish the final stages of landing. Small rotations relate to big centroid displacements.
- The errors are measured from the output of a Kalman filter which takes a finite number of iterations to converge to the real position, as such a minor glitch shows for Y axis error.
- The simulator introduces a minor constant drift, which physically would be translated to wind. Given both roll and pitch axis were given the same step input this explains the different error progressions on the X and Y axis.

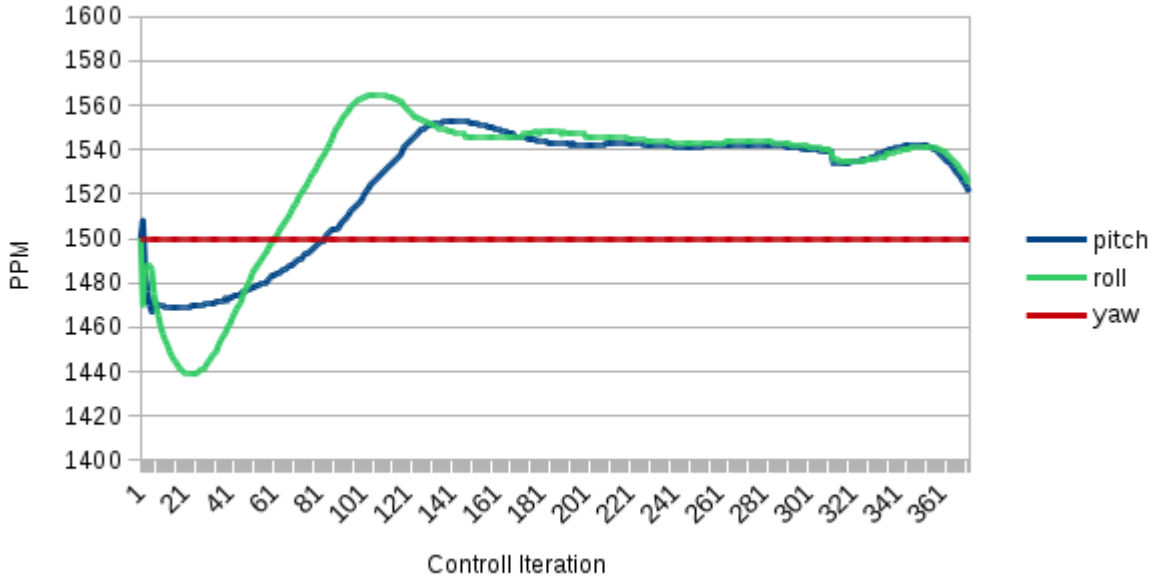



Figure 4.10: Attitude PPM over Time

- At approximately the 130th iteration both errors decrease approximately linearly. This is due to the fact that the system is mostly controlled by the integral gain.
- Yaw was ignored because it has much simpler dynamics.
- On the final iterations there is a very small error (15 pixel) consistent with the baseline error of the Kalman filter.

Algorithm 11 Descent Rule set

1: if $\sqrt{E_x^2 + E_y^2} \geq \varepsilon$ then
2: $\quad Throttle \leftarrow 1420$
3: else
4: $\quad Throttle \leftarrow 1150$
5: end if

For descent, a rule set was created as a way to force the approach from above and not sideways. A sideways approach to the station will possibly create a partial occlusion of some of the targets visual features. Also, a sideways approach is incompatible with the geometry of the station and intended quad-rotor-station meshing. While it is true that the rule set for throttle control signal could be filtered by an integrator filter this aspect has not shown itself relevant for the success of the landing.
The altitude is negative because it follows the North-East-Up referential convention instead of the North-East-Down.
An important aspect to note is that while the quad-rotor was maneuvering and the camera was rotating around the attitude axis the visual altitude measurement maintained a consistent error signal, meaning that the stabilization of image worked as intended.
Another aspect to track during the final validation was the number color patches recognized during the whole visual landing maneuver. In figure 4.12 on the left there is a graphic about the ratio of

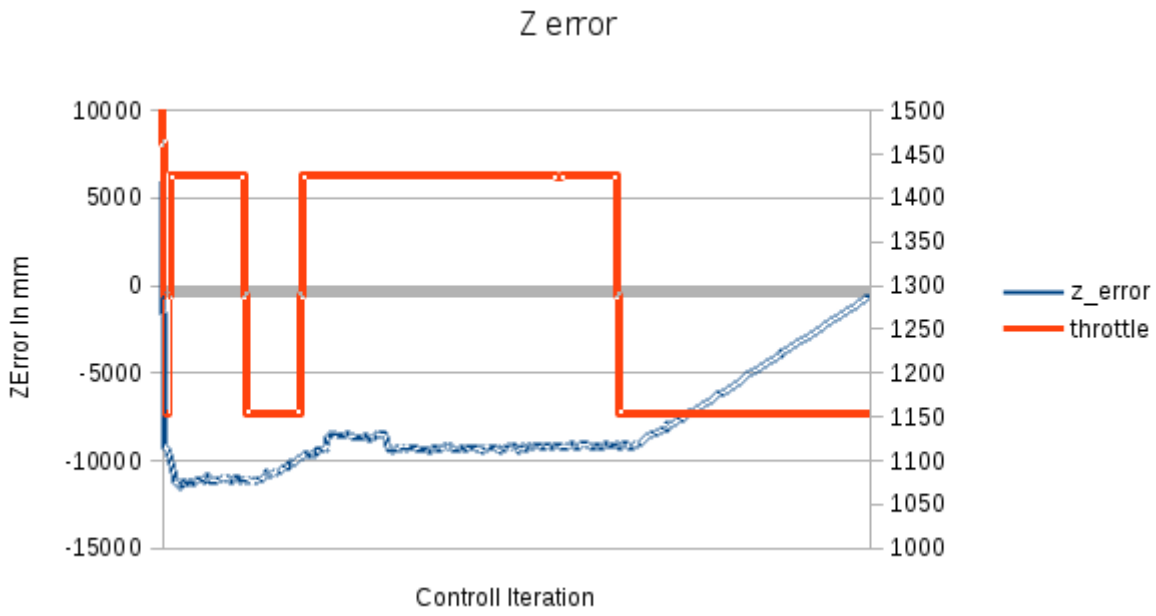



Figure 4.11: Vertical Position Plot

valid recognitions to the total of flight iterations. On the right a graphic about the frequency of detection for all regions is shown In these graphic some observations can be made:

- All regions except the purple one, were recognized higher than 95% of the time.
- The purple color was shown inadequate not being recognized almost 20% of the time.
- The maximum recognition rate was of 99.7% by Red, Cyan and Yellow.
- Due to the red color having the biggest area it suffered from image fragmentation. 30% of the time it had greater than 2 regions detected or, in the context of the problem, false positives.
- All colors had false positives along the procedure.
- The high number of false positives did not impede the successful landing. This means the whole system is satisfactorily resilient to noisy data.

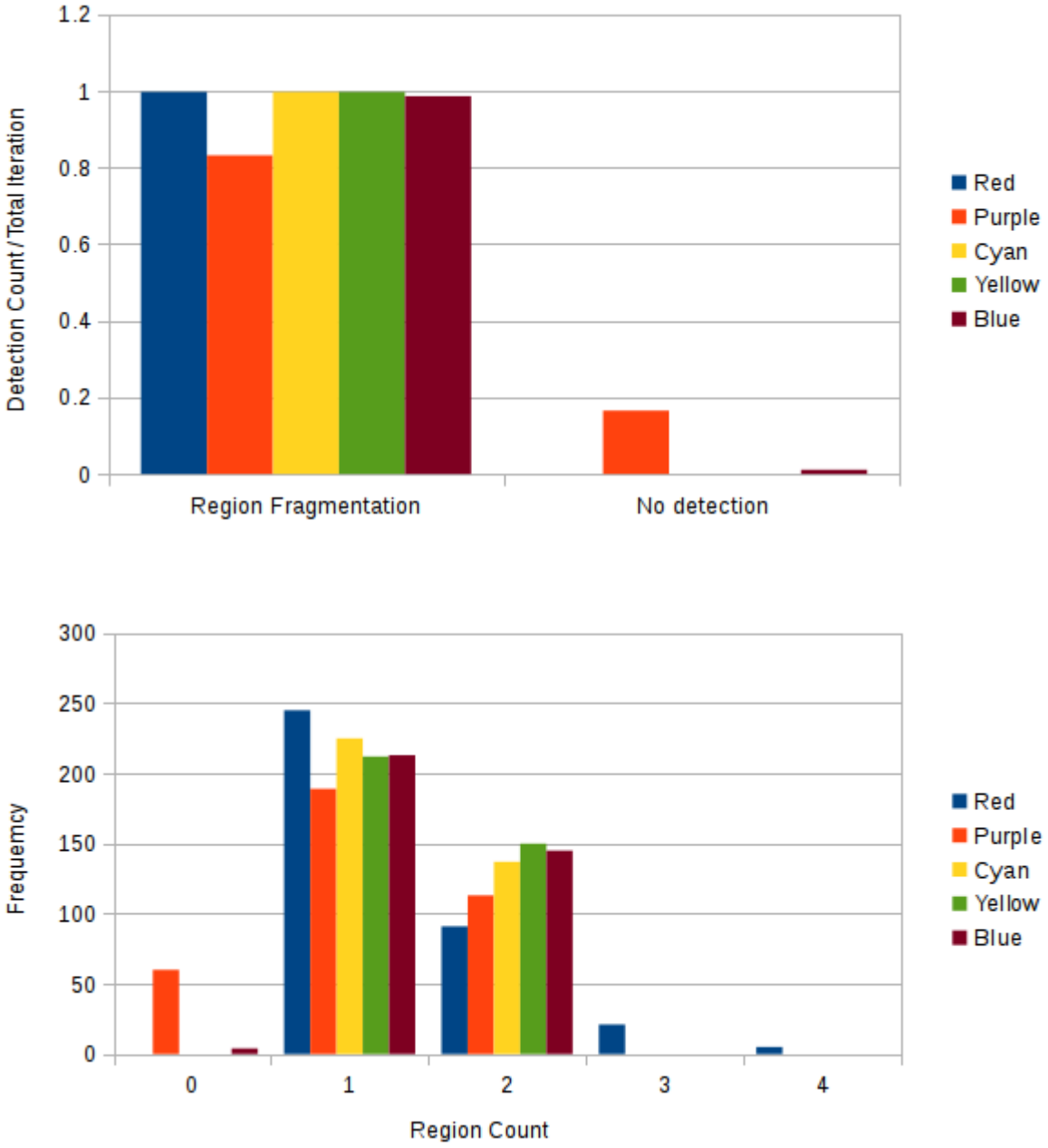



Figure 4.12: Region Data from Landing Operation

# Chapter 5

# Conclusion

In this work it has been demonstrated that several areas of knowledge are necessary to achieve the goal of vision based docking. It also shows that sometimes getting a technology developed with less advanced scientific principles can give adequate results and serve as a stable stepping stone for continuous improvement instead of a revolutionary complex implementation. Alas, the most important lesson that may even not be clear in this work, is that when working with multi-disciplinary technology, careful system architecture is critical to project success.

Conceptually, or in the domain of strategy, the mission is very simple and can be described as in figure 1.1 in 4 steps. Tactically, or in the domain of scientific tools to employ, there is a huge body of bibliography that is applicable to solve the proposed problems, much of which was not included in this work. But operationally, several unaccounted-for factors make themselves obvious. These factors are not only unanticipated but also require very specialized understanding to take them into account. The specialized nature makes it very hard to obtain guidance and huge amounts of time can be wasted for conceptually very small parts of the overall work. Also, the author feels most practical programming of projects beyond proof of concept, require great knowledge of software development best practices, not only in code, but in project management.

An example of a requirement for the very specialized work described, was found when writing the MAVLink communication library. Due to constraints on real time processing of the Arducopter, the calculation of geodesic coordinates of GPS waypoints are converted to meters through an approximate formula, and not through WGS-84 recommended formulas. This fact led to the searching of the whole Arducopter code base for where this conversions were made. Also finding out the correct messages sequences that override Arducopter physical initialization was only possible with a mixture of search in both the Arducopter source code and a Python utility called MAVProxy.

Another example, but of the tactical difficulty in transitioning from the algorithmic perspective of a problem to the operational implementation on a computer, was found when implementing the image labeling algorithm. Specifically the general algorithm does not handle upper and left borders without special cases.

The same can be said about the calculation of rotation matrices. Rotation matrices should always return an identity matrix when multiplied by its transpose but in practice they rarely do. Mathematically they should, but the numeric constraints of floating point operations and the precision of the calculation of transcendental functions destroy this ideal concept. When this ideal concepts are not valid additional work unrelated to the problem is necessary.

Finally, this work is intended to be continued beyond academic interest and that is one of the reasons the technological aspects of the work were given so much emphasis, in a way, sacrificing state of the art science in favor of functional science. An example of a work with very similar goals and scientifically more developed than this is found in [29]. The experience the author has gotten is also almost as gratifying as the the hypothetical degree of a master degree.

## 5.1 Future Work

In the final validation it was demonstrated that the technologies developed allowed for the successful achievement of this works goals, even if only on a synthetic environment. Due to the way the technology was set up, there is a level of confidence that the system architecture developed would be capable of supporting real environments and components with comparable results. In the event of a transition to the real environment the likely problems would come from the inadequacy of some of the technical solutions, specifically the type of controller(PI), image recognition(image segmentation) and Kalman Filter model.

For the controller, an investigated solution was the creation of control laws through the Backstepping method. The reason it was not implemented was that it required the identification of physical characteristics of the body and propeller system. This identification would need to be done experimentally or online. For the experimental identification, the Universidade of Beira Interior will soon have a working propulsion validation system, which could be used to identify the propulsion model. Unfortunately for the measurement of other characteristics, like the moment of inertia there is not any implemented experimental protocol in place, although CAD tools could be of great assistance. In [29] an experimental method is proposed. On the other hand this properties are constant in time and a model could be developed for an augmented Kalman filter to identify this properties online, based on the regular sensors onboard a quad-rotor. Optimal control techniques would always require some form of system identification.

Another hindrance in implementing the Backstepping method control laws with the current work is that, as far as the author is aware, there is no way to directly override the PWM signal to send to the motor drivers, because this would defeat most of the autopilot functionality of Arducopter. In this mode Arducopter would essentially work as a data acquisition and general purpose input output device. Not being able to control directly the PWM signal of the motors means the landing control algorithm runs cascaded with other control loop, which complicates analysis unnecessarily. In a future work the author has confidence it is feasible to add a mode allowing complete motor override and gradually transitioning other navigation functions to custom developed technology. This way advantage can be taken of the hardware and its already developed drivers. The drivers are one of the most critical parts of using advanced digital components and having their data available without concern for their internal protocols helps the focusing of the development of the navigation task itself.

The Kalman filter implemented does not integrate a more realistic model with online target and airplane identification. Tracking robustness is then sacrificed. A Kalman filter that would track characteristics of the image is one aspect which was not explored by this work at all.

The physics simulator also has shortcomings in the propulsion model, specifically the electrical, low power operation and ground effect model. The most important shortcoming is the use of Euler angles and gimbal lock. Integration of rotation vectors would preserve the numerical stability and mathematical intuition of the system while avoiding gimbal lock[3].

The visual environment set up is very rudimentary consisting in the rendering of only the target model, while it would be more realistic to model the terrain along with its texture. The development of these features is slightly outside the scope of knowledge of Aeronautical engineering, with limited advantages.

Regarding the image segmentation problem, the implementation of a camera response function coupled with the probabilistic approach to image segmentation would likely yield very good results. While this work proposed and implemented a probabilistic approach it was not tested with due validation.

Even without extending this work as proposed before, full flight validation can already be tried. With a fully functioning quad-rotor with Arducopter autopilot, an XBee and a camera broadcasting system to a computer on the ground, this work could be physically validated. To do this, an experimental protocol must be developed to ensure the safety and validity of the results of a whole landing and battery swapping mechanism.
The target pattern design was found adequate for a successful validation although in reality other colors might be more appropriate. Specifically the purple color would have to be replaced. This task is trivial because for this work a target pattern generator was already developed.
The communication library does not implement every possible functionality but has proved to be very stable throughout this project. While during it's use no problem has surfaced, the most likely point of failure is the concurrency functionality of the library. After the support for C++ 2011 standard becomes more mature some features of the library can also be re-factored to modularize the code further.
To finish, the author has confidence that the experience gathered from this work may be very useful for the continuation of other technological projects.

# Bibliography

[1] Simon A. Eugster, "CbCr plane of the YCbCr color space, with a Y value of 1, Cb on the horizontal x axis going from -1 to 1, Cr on the y axis going from -1 to 1 as well. Created with kdenlive's export function in the Vectorscope (development version)." 2010.

[2] of Ottawa VIVA Lab, O. B. U., "How to calculate perspective transform for OpenCV from rotation angles?" http://stackoverflow.com/questions/17087446/how-to-calculate-perspective-transform-for-opencv-from-rotation-angles/20089412?noredirect=1#comment38883579_20089412.

[3] Diebel, J., "Representing attitude: Euler angles, unit quaternions, and rotation vectors," 2006.

[4] Hoffmann, G. M., Huang, H., Waslander, S. L., and Tomlin, C. J., "Quadrotor helicopter flight dynamics and control: Theory and experiment," Proc. of the AIAA Guidance, Navigation, and Control Conference, Vol. 2, 2007.

[5] Gallier, J., "Basics of Classical Lie Groups: The Exponential Map, Lie Groups, and Lie Algebras," Geometric Methods and Applications, Springer, 2001, pp. 106–107.

[6] Lerner, D., "Symmetric and skew-symmetric matrices," Lecture notes on linear algebra, Springer, 2007, pp. 367–414.

[7] Madani, T. and Benallegue, A., "Backstepping control for a quadrotor helicopter," Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, IEEE, 2006, pp. 3255–3260.

[8] Charles Poyton, "Frequently Asked Questions about Color," 1997.

[9] Tkalcic, M., Tasic, J. F., et al., "Colour spaces: perceptual, historical and applicational background," Eurocon, 2003.

[10] Wyszecki, G. and Stiles, W. S., Color science, Vol. 8, Wiley New York, 1982.

[11] CIE, "CIE 1931 XYZ color matching functions," http://cvrl.ioo.ucl.ac.uk/database/data/cie/Illuminanta.csv.

[12] Heikkila, J., "Geometric camera calibration using circular control points," Pattern Analysis and Machine Intelligence, IEEE Transactions on, Vol. 22, No. 10, 2000, pp. 1066–1077.

[13] Li, H., Hestenes, D., and Rockwood, A., "Generalized homogeneous coordinates for computational geometry," Geometric Computing with Clifford Algebras, Springer, 2001, pp. 27–59.

[14] Reid, Term, U. o. O., "Estimation II, Lecture Notes 2," www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf, 2001.

[15] Kelly, A., "A 3D state space formulation of a navigation Kalman filter for autonomous vehicles," Tech. rep., DTIC Document, 1994.

[16] Goossens, B., Luong, H., Aelterman, J., PiÅŸurica, A., and Philips, W., "Realistic camera noise modeling with application to improved HDR synthesis," EURASIP Journal on Advances in Signal Processing, Vol. 2012, No. 1, 2012.

[17] Arkin, R. C., An Behavior-based Robotics, MIT Press, Cambridge, MA, USA, 1st ed., 1998.

[18] Tuytelaars, T. and Mikolajczyk, K., "Local Invariant Feature Detectors: A Survey," Found. Trends. Comput. Graph. Vis., Vol. 3, No. 3, July 2008, pp. 177–280.

[19] Bradski, G., "OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

[20] Eugene Khvedchenya, "Comparison of the OpenCV's feature detection algorithms," 2011.

[21] Bruce, J. and Veloso, M., "Fast and accurate vision-based pattern detection and identification," Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on, Vol. 1, IEEE, 2003, pp. 1277–1282.

[22] Klančar, G., Brezak, M., Matko, D., and Petrović, I., "Mobile robots tracking using computer vision," AUTOMATIKA: časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije, Vol. 46, No. 3-4, 2006, pp. 155–163.

[23] Bruce, J., Balch, T., and Veloso, M., "Fast and inexpensive color image segmentation for interactive robots," Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on, Vol. 3, IEEE, 2000, pp. 2061–2066.

[24] Free Software Foundation, I., "A GNU Manual," `https://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Other-Builtins.html`, 2013.

[25] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., et al., Introduction to algorithms, Vol. 2, MIT press Cambridge, 2001.

[26] Bader, D. and Pearcy, M., "Material properties of Velcro fastenings," Prosthetics and orthotics international, Vol. 6, No. 2, 1982, pp. 93–96.

[27] Lemaire, T., Berger, C., Jung, I.-K., and Lacroix, S., "Vision-Based SLAM: Stereo and Monocular Approaches," Int. J. Comput. Vision, Vol. 74, No. 3, Sept. 2007, pp. 343–364.

[28] Carreira, T. G., "Quadcopter Automatic Landing on a Docking Station," 2013.

[29] Mendes, A., "Vision-based automatic landing of a quadrotor UAV on a floating platform," 2012.