UNIVERSIDADE D
COIMBRA

João Pedro Jacinto Sousa

Fiducial Markers For Autonomous UAV Landing

Dissertação no âmbito do Mestrado em Engenharia Eletrotécnica e de Computadores, do Ramo de Robótica, Controlo e Inteligência Artificial, orientada pelo Professor Doutor Lino José Forte Marques e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Fevereiro de 2023

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Fiducial Markers For Autonomous UAV Landing

João Pedro Jacinto Sousa

Coimbra, February 2023

Fiducial Markers For Autonomous UAV Landing

Supervisor:

Prof. Lino José Forte Marques

Co-Supervisor:

Master Rui Pedro Rodrigues Baptista

Jury:

Prof. Hélder de Jesus Araújo

Prof. Nuno Miguel Mendonça da Silva Gonçalves

Prof. Lino Marques

Dissertation submitted in partial fulfilment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, February 2023

Acknowledgements

I would like to thank my supervisors Professor Lino Marques and Master Rui Baptista for all the guidance and patience to lead me in the right way, as well as all the knowledge taught to me, which is invaluable.

I would like to thank the Embedded Systems Laboratory of the Institute of Systems and Robotics (LSE-ISR) of the Univerisity of Coimbra for providing the space and the materials to develop this dissertation, as well as the real world robotics skills which will be a major help in my future career.

I would like to express my gratitude to Sedat Dogru, who went out of his way to guide me and for the time and patience he dedicated to help me finish this work.

I would like to thank my friends, namely André, Chaluça, Joca and Sgotti for their unending support and motivation, brilliant idea and their help in developing my work when they had dissertations of their own to complete.

I would like to thank my family for providing me the opportunity to study what I love.

Finally, I would like to thank my girlfriend, Leonor, for the encouragement and especially the love and support when I needed it most.

Resumo

Nas décadas recentes, os unmanned aerial vehicle (UAV), vulgarmente conhecidos por drones, têm-se vindo a tornar cada vez mais populares, tanto profissionalmente como comercialmente, permitindo tarefas de supervisão e do género. Contudo, a sua baixa autonomia energética tornou-se um obstáculo à completa autonomia energética dos sistemas que integram estes robôs. Entre as várias soluções formuladas para resolver este problema, uma que se destaca é o uso de estações de carregamentos. Estas são postos onde o UAV pode aterrar para recarregar a sua bateria via um par de bobinas. No entanto, desta tecnologia surge outro desafio: aterrar com precisão nas estações de carregamento. Então, esta dissertação visa melhorar trabalho passado através do desenvolvimento de um sistema robusto de aterragem autónoma baseado em visão por computador. O algoritmo usa uma câmara e um marcador fiducial para controlar o voo do UAV até ao marcador e aterrar precisamente nele. Esta dissertação também se foca em comparar diferentes marcadores fiduciais para determinar quão apropriados são para este fim, bem como desenvolver um marcador fiducial customizado e um algoritmo de deteção para o mesmo. Uma experiência controlada foi feita para comparar os alcances de deteção e errors de estimação de posição dos marcadores. Outra experiência foi realizada em ambiente de simulação para testar e validar o algoritmo de aterragem.

Keywords: UAV, Drone, Aterragem Autónoma, Visão por Computador, Marcadores Fiduciais

Abstract

In the recent decades, UAVs have become increasingly popular both professionally and commercially, allowing for automation of surveillance tasks and the like. However, their low energetic autonomy has become an obstacle in the way of full automation of systems integrating these robots. Among the many solution that have been explored, one that shows promise is inductive wireless charging via charging stations. These are outposts where a UAV can land to have its battery recharged via a pair of transmitter-receiver coils which raise their own challenge: landing precisely on them. Thus, this dissertation looks to improve upon previous work by developing a robust vision-based landing algorithm. The algorithm uses a camera and a fiducial marker to control the UAV's flight towards the marker and land precisely on top of it. This dissertation also focuses on comparing different fiducial markers to determine how well suited they are for this application, as well as developing a custom fiducial marker (CFM) with its own detection algorithm. A controlled experiment was done to compare the markers' detection range and pose estimation errors. Another experiment was done in simulation to test and validate the landing algorithm.

Keywords: UAV, Drone, Autonomous Landing, Computer Vision, Fiducial Markers

"Our business is not to see what lies dimly at a distance, but to do what lies clearly at hand."

— Thomas Carlyle

Contents

List of Acronyms

2D	two-dimensional
6-DoF	six-degrees of freedom
AI	artificial intelligence
API	application programming interface
CFM	custom fiducial marker
CNN	convolutional neural network
CPU	central processing unit
ENU	east-north-up
ESD	east-south-down
FCU	flight controller unit
FLANN	fast library for approximate nearest neighbours
GCN	graph convolutional network
GCS	ground control station
GPS	Global Positioning System
ID	identification
IMU	inertial measurement unit
IO	input-output
MPC	model predictive control
PI	proportional-integral

PID	proportional-integral-derivative
QR code	quick response code
RGB	red, green and blue
ROS	Robot Operating System
RTK-GNSS	real-time kinematic global navigation satellite system
SBC	single-board computer
SDRAM	synchronous dynamic random-access memory
SITL	software in the loop
SLAM	simultaneous location and mapping
SURF	speeded up robust features
UAV	unmanned aerial vehicle
USB	universal serial bus
USD	United States dollar

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Context and motivation

UAVs, more commonly known as drones, are aircrafts that can be remotely controlled, fly autonomously, or a mixture of both, and carry a multitude of sensors and devices to perform a variety of tasks [2]. Their advantages include a simple structure, reduced size, low cost and flexibility. They have uses in many fields, which can be categorised into: search and rescue; coverage; construction; delivery of goods [3]. The global UAV market was worth, in terms of revenue, United States dollar (USD) 18.28 billion in 2020 and it is forecast to reach USD 40.90 billion in 2027, with a compound annual growth rate of 12.27 % from 2021 to 2027 [4].

In the last decade, UAVs have become increasingly more popular among professionals and civilians and their main challenges have also become clearer. One of these challenges pertains to energetic autonomy, which is one of the main drawbacks of these systems. The simplest way to solve this problem is using batteries with greater capacity, but, as the capacity of a battery increases, so does its weight, its cost and its size. A greater weight increases the amount of thrust needed to keep the UAV in the air, thus increasing the energy consumption of the UAV. Although there are studies and commercial products aimed at maximising the autonomy of UAVs, most of them are expensive and may require an aircraft adapted to the solution, which makes said solution unfeasible for commercial drones. So, literature has focused on four other solutions: battery management, battery recharging, solar power, and machine learning and communication techniques [5]. This work is aimed at battery recharging, whose main solutions are tethering, energy harvesting and charging stations.

Tethering consists of linking the UAV to a ground control station (GCS) via a tether, which allows for efficient charging and other perks, such as a wired connection to the Internet [6]. However, tethering severely limits the UAV's work space and creates new challenges in terms of flight control, due to the weight of the tether and its variable length [7].

Energy harvesting is a technique which collects energy from ambient sources. These

sources include but are not limited to solar energy, wind energy, vibration and electromagnetic fields [8]. It is a very situational strategy as it depends on the weather conditions or on large electromagnetic sources, such as power lines, which are not always available and induce interference on the connections between the UAV and other devices, like a radio remote controller or a GCS.

Charging stations are posts where the UAV can land to have its battery recharged. Though this solution does not provide charging as efficiently as tethering and, unlike energy harvesting, requires the aircraft to land, it has become the preferred solution among investigators due to the high degree of design freedom and the number of different charging methods applicable. By using charging stations instead of tethering or energy harvesting, there is great freedom for placement of scattered stations, which allow UAVs to cover large distances, albeit not continuously, and perform long missions with full autonomy [9]. These charging stations may also serve as GCSs where the drone may communicate gathered data to a remote server or receive new missions. The DJI Dock is an example of a commercial product with the previously mentioned features [10].

As this solution requires the drone to land on a relatively small structure, autonomous landing is of the utmost importance for recharging. There is a multitude of autonomous landing solutions, one of which is vision-based autonomous landing. This is due to cameras being passive sensors, generally cheap, lightweight and easily adaptable to existing platforms, which makes them very sought after in the UAV field [11]. A solution vision-based UAV navigation and localisation that is commonly relied on is the use of visual fiducial markers, highly recognisable patterns with strong geometric and visual features which may also encode information and fail-safes with error correction. A great deal of the already available fiducial markers have associated software libraries that provide pose estimates of the markers relative to the camera, which makes them ideal for landing.

1.2 Objectives

This dissertation has two main objectives: comparing the performance of some of the most popular fiducial markers and a custom marker for the purpose of vision-based autonomous landing; developing an autonomous landing algorithm which relies on the detection of fiducial markers.

1.3 Document structure

This dissertation is organised as follows:

- Chapter 2 - Related Work contextualises this work in the literature of visual fiducial markers and vision-based autonomous UAV landing;
- Chapter 3 - Methods provides the theoretical background for the design and detection of the CFM and the autonomous landing algorithm;
- Chapter 4 - Experimental Work lists the hardware and software components utilised in the experimental phase of this dissertation and discusses in detail the experimental work done;
- Chapter 5 Conclusion and Future Work summarises what was concluded from the results of this work and suggests improvements.

2 Related Work

This chapter provides background and discusses the history, progress and state of the literature on major topics of research concerning UAVs, namely, autonomous UAV landing and fiducial markers.

2.1 Fiducial markers

A visual fiducial marker is an object in the field of view of an imaging system that serves as a reference or measurement for it. In the field of mobile robotics, the objects known as fiducial markers, usually called tags, are artificial visual features of known dimensions that are easy to recognise and distinguish from one another. These may also support encoding and error detection and correction. As mentioned in Section 1.1, fiducial markers are very useful in robotics for their ability to provide pose estimates and even encoded information, making them great for vision-based navigation and localisation. Fiducial markers have a multitude of purposes in the field, such as providing instructions for a robot, a six-degrees of freedom (6-DoF) pose estimate from a single marker with known dimensions or ground truths for various tasks and aiding in simultaneous location and mapping (SLAM) and navigation problems, and providing .

One of the most popular markers is ARTag [12], which is based on ARToolKit[1], as most square-shaped markers are. ARToolKit is a system initially designed to track markers for augmented reality video conferences. Its marker detection consists of applying a global threshold to the image to search for areas which can be fitted by four line segments. Unlike ARToolKit, ARTag uses digital coding theory to design the pattern inside the marker, allowing for encoding of information and error correction. It also uses a gradient-based strategy to detect line segments which form a quadrilateral, which yields more reliable detections and better performance with partially occluded markers. The internal pattern of this marker is

[1]See `http://www.hitl.washington.edu/artoolkit.html`

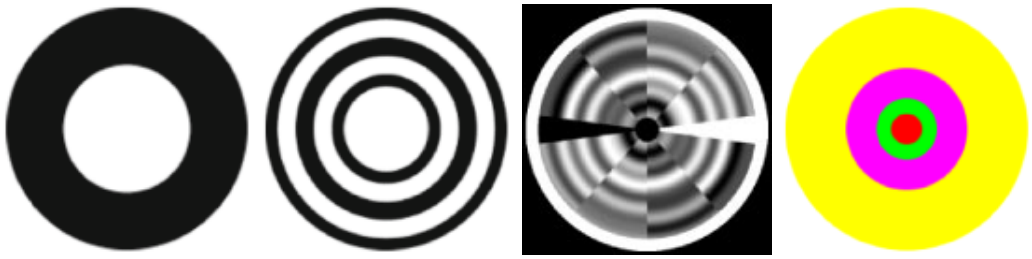
a 6-by-6 matrix, i.e., a 36-bit code. Example markers of ARToolKit and ARTag are shown in Figures 2.1a and 2.1b, respectively.

(a) ARToolKit. (b) ARTag.

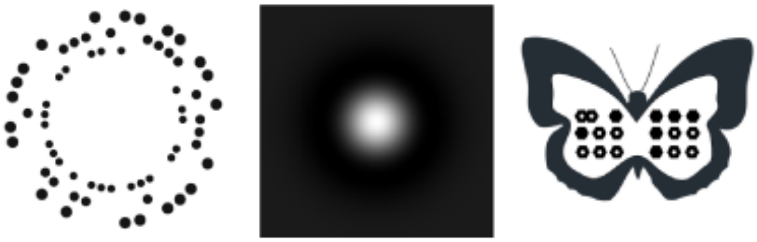
Figure 2.1: Examples of ARToolKit and ARTag markers.

Although a great majority of the popular fiducial markers are based on square 2D barcodes, also known as quick response codes (QR codes), there are other which use different geometric structures, such as BullsEye, CCC, CCTag, FourierTag, InterSense, Multi-ring, Pi-Tag, RUNE-Tag, SIFT, STag, Topotag, TRIP and WhyCode [13]. Some of note among these are:

- CCC [14] (Figure 2.2a), which is the simplest fiducial and does not provide any information besides position;
- CCTag [15] (Figure 2.2b), which expands upon CCC by adding rings with different radii, which can encode information, much like a traditional bar code;
- FourierTag [16] (Figure 2.2c), which is designed to have a detection that degrades continuously as the distance to it increases and encodes data in the form of a greyscale pattern based on the Fourier transform;
- Multi-ring [17] (Figure 2.2d), which encodes data using colour;
- RUNE-Tag [18] (Figure 2.2e), in which the information is encoded in dots of varying sizes arranged in a circular pattern;
- SIFT [19] (Figure 2.2f), which aims to create artificial SIFT features in a scene and cannot encode information or provide orientation information due to its radial symmetry;



(a) CCC. (b) CCTag. (c) FourierTag. (d) Multi-ring.



(e) RUNE-Tag. (f) SIFT. (g) TopoTag.

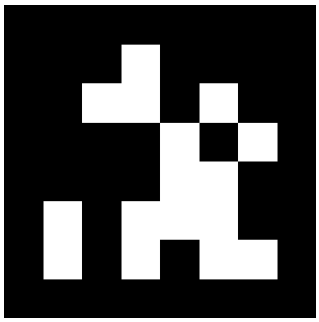
Figure 2.2: Examples of CCC, CCTag, FourierTag, Multi-ring, RUNE-Tag, SIFT and TopoTag markers.

- TopoTag [20] (Figure 2.2g), which allows custom-shaped markers.

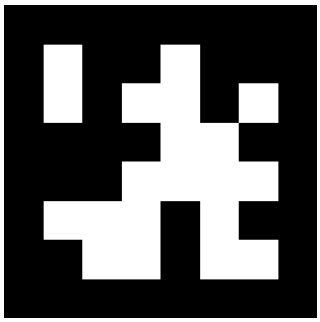
Among all the markers mentioned, the ones selected for the purposes of this dissertation are ArUco, AprilTag and STag, mainly due to the availability of an OpenCV module for detection and pose estimation of ArUco markers, a Python module for detection of AprilTag markers with pose estimation provided by OpenCV and a ROS package for the detection and pose estimation of STag markers. Additionally, a custom marker has been developed from scratch and after all the modifications it arrived at a design very similar to CCTag, although independently.

AprilTag [21] was built on top of ARTag's foundations and improves the original in a number of different ways. The image segmentation is based on graphs and gradient patterns to perform quad extraction, which allows non-intersecting edges to be considered marker candidates. This marker also improves upon the 2D barcode system and increases its robustness against rotation and reduces the detection of false positives outdoors. The new encoding system also increases robustness against warping and occlusion and bolsters a decrease in misdetection rates. On top of these advantages, the detection algorithm is to this day being further developed by investigators. AprilTag has several different marker dictionaries, each with its own unique combination of number of unique marker IDs, Hamming distance between marker IDs and number of bits. Three example markers from the 36h11 dictionary are presented in Figures 2.3a-2.3c.

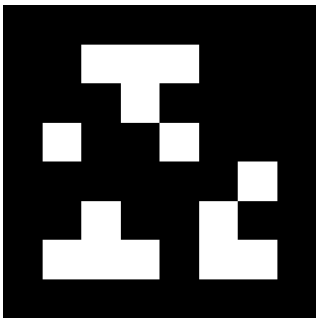
The Hamming distance is an information theory concept which represents the number of different symbols in two strings of equal length, i.e., the minimum number of substitutions needed to transform one of the strings into the other. In encoding systems, greater Hamming distances translate to better error correction, since more errors are necessary to transform one code to another in the dictionary. However, to obtain greater Hamming distances, it is usually necessary to decrease the number of unique IDs in a dictionary.



(a) AprilTag 36h11 marker with ID 0.



(b) AprilTag 36h11 marker with ID 1.

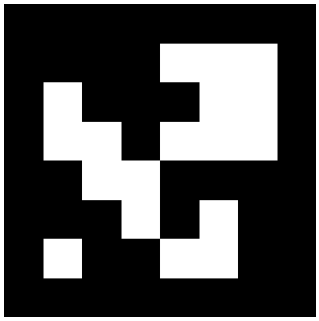


(c) AprilTag 36h11 marker with ID 2.

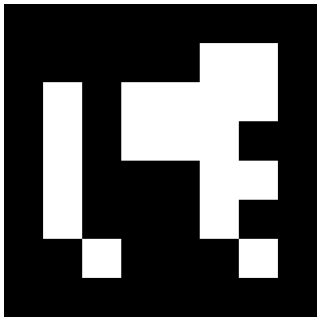
Figure 2.3: Examples of AprilTag markers from the 36h11 dictionary with different IDs.

ArUco is also based on ARTag and ARToolKit and its biggest contribution is allowing the user to configure their own dictionaries, instead of relying solely on previously available ones, like most other markers. Since the dictionaries are custom-generated, they do not need to include as many markers as other dictionaries, which decreases the computation time and effort. The dictionaries created by the user only contain markers with the maximum Hamming distance possible. Figures 2.4a-2.4c display examples of ArUco markers from ArUco 6x6 250 dictionary, which as a dictionary size of 250 and a marker size of 36.

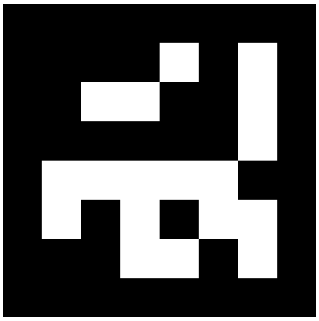
The collection of markers taken into account in a particular application is known as a dictionary of markers. It is merely a list of each marker's binary codifications. A dictionary is described by two values: the dictionary size and the marker size. The dictionary size is the number of unique IDs in the dictionary. The marker size is the number of bits that the markers have. Some dictionaries do not specify their size, stating instead the Hamming distance between IDs. One instance of a dictionary which specifies its dictionary and marker sizes is the ArUco 4x4 50, which has a dictionary size of 50 and a marker size of 16 (4x4 means $4 \cdot 4 = 16$ bits). One instance of a dictionary which instead specifies the Hamming distance between different IDs and its marker size is the AprilTag 36h11, which has a marker size of 36 (which means the square codes are 6-by-6 bits) and a Hamming distance between IDs of 11.



(a) ArUco 6x6 250 marker with ID 0.



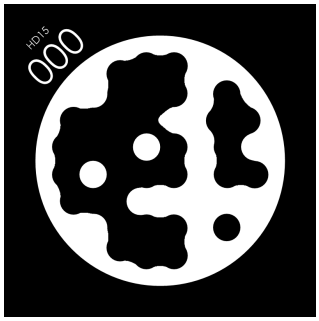
(b) ArUco 6x6 250 marker with ID 1.



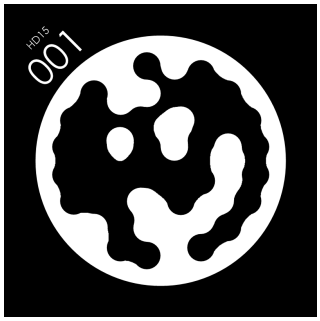
(c) ArUco 6x6 250 marker with ID 2.

Figure 2.4: Examples of ArUco markers from the 6x6 250 dictionary with different IDs.

STag markers, like the two previous markers, are also surrounded by a square border, but the markers themselves are circular. The focus of STag is achieving great stability in pose estimation measurements. Thus, after the line segmentation stage, a preliminary homography is computed for the detected marker, which is then refined with ellipse fitting. This refining stage is shown to provide better results in localisation compared to quadrilateral extraction methods. It also increases the stability of the measurements [22]. An instance of an STag marker is shown in Figures 2.5a-2.5c.



(a) STag HD15 marker with ID 0.



(b) STag HD15 marker with ID 1.

(c) STag HD15 marker with ID 2.

Figure 2.5: Examples of STag markers from the HD15 dictionary with different IDs.

2.2 Vision-based autonomous landing

Saripalli [23]-[24] proposed a system to autonomously land a UAV that combined Global Positioning System (GPS), a three-axis gyroscope, a three-axis accelerometer, an ultra-sonic sonar and a colour camera. Apart from the aircraft and its components and sensors, the system also had a GCS that sent high-level commands and differential GPS corrections to

the UAV. The vision system identified the landing target by utilising Hu's moment of inertia descriptors for the geometric figure it detects in the image and the control architecture was behaviour-based with three modes: search mode that used GPS to approach the general vicinity of the landing area; track mode that used vision to align the UAV with the landing pad; landing mode. It achieved landing times between 62 s and 112 s, with a mean of 73 s and a mean autonomous flight time of 234 s. The yaw misalignment was between 0° and 15°, with a mean of 6° and a standard deviation of 5°. The mean error was position is 40 cm. The approach did not include the inertial measurement unit (IMU) measurements in the control loop. At 10 frames per second, the total central processing unit (CPU) time of acquiring images, thresholding and filtering them, segmenting them, labelling the components and compute the Hu's moments of inertia of the components was about 89 %.

Venugopalan [25] suggested a new approach to the UAV-GCS relationship by placing the computer acting as the controller on the UAV instead of the GCS. Although it placed a greater energetic stress on the aircraft and was constrained by a reduction in the available computational resources, there was no input lag in the control system, as the control measurements and commands did not have to be communicated over a wireless network. In the experimental work, a coloured marker with no information encoded was used for landing with vision. Another improvement achieved by this work was the correction of the error in the estimation of the pose of the landing target that arised from non-zero roll and pitch angles. When these angles are non-zero, the target appears offset in the image, which, without compensation, induces a significant error in the pose estimation that increases with the altitude of the UAV.

Baca [26]-[27] proposed a vision-based landing control strategy that utilised model predictive control (MPC). The solution presented in the paper was for the one of the trials in the Mohamed Bin Zayed Internation Robotics Challenge (MBZIRC) 2017 competition, which challenged the competitors to land a UAV autonomously on top of a moving target as fast as possible. The state of the moving target was previously known. The aircraft was equipped with magnets on the landing gear to adhere to the target, a rangefinder, a camera and a real-time kinematic global navigation satellite system (RTK-GNSS) receiver, and used an unscented Kalman filter (UKF) to estimate the state of the car and MPC to generate the desired state for the UAV. The team that devised this approach achieved the fastest landing time in the competition, 25 s.

Sudevan [28] used template matching with speeded up robust features (SURF) and fast library for approximate nearest neighbours (FLANN) to find the landing target in the image

and used a state machine architecture for the landing behaviour of the UAV. The controller was a proportional-integral-derivative (PID) controller.

Lee [29] departed from traditional computer vision and relied on an artificial intelligence (AI) agent to command the aircraft to approach the landing target, although the agent was taken off command when the UAV was 1.5 m above the target due to safety reasons. Nonetheless, it proved an AI agent trained in simulated environments with reinforcement learning could emulate a human pilot and safely approach a landing destination.

Nguyen [30] used the lightweight lightDenseYOLO convolutional neural network (CNN), a combination of the lightDenseNet and YOLO v2 neural networks, the first as a feature extractor and the second as the marker detection module. On top of the previously mentioned network, Profile Checker v2 was also added to enhance the predicted results. The UAV was equipped with a gimbal that kept the camera's optical axis at a 90° angle with the ground. The neural network ran on an Android device mounted on the UAV.

Liu [31] uses height-adaptive control, where the PID controller's constants vary with the relative altitude of the drone, to account for the increased influence of the velocity commands on the position of the target in the image when the camera is closer to the target.

Zhang [32] further explored machine learning applied to vision-based autonomous landing by implementing a novel graph convolutional network (GCN) hybrid decision network strategy to land a UAV. The evaluation of this method was only in a simulated environment, however.

Keipour [33] used a gimbal mounted on the drone to point the camera towards the landing target. By providing a way to detect the target without needing to be directly over it, this approach shortens the landing times and facilitates landing in moving targets. The target in the experimental work of the paper was moving in a straight line with constant velocity of 15 km/h so that, if it got out of the field of view of the camera, the UAV would simply need to accelerate in the direction where the target was last seen. Out of 22 recorded trials where the controller needed to bring the UAV close to the target 19 were successful. The three 3 failed trials were due to excessive sunlight reflecting on the target. Out of 9 recorded landing trials 5 were successful. The time between first detecting the target and landing varied between 5.8 s and 6.5 s. The paper provides videos of the trials from the point of view of the aircraft's camera.

Persson [34] also used MPC but improved upon it by introducing a variable look-ahead horizon which computed the best possible value to make the controller problem solvable but not so much that it became too computationally expensive. In the experimental outdoors

work, the wind speed was of about 5 m/s. As a method to counteract the effect of the wind, the controller was allowed to utilise not only the first optimal input computed by the variable horizon MPC, in case the control problem was not being solved fast enough.

Lee [35] proposed an approach to solve the problem of autonomously landing a UAV on a target placed on a ship, which has heave, sway and surge movements that must be accounted for. In order to solve the problem, machine learning and classical computer vision were fused. The machine learning component performed long-distance object detection to track the ship and the classical computer vision component estimated the pose of the UAV relative to the ship. The experimental work consisted of over 100 landing trials and the aircraft successfully landed on the 2 m square target with a safe landing threshold of 0.35 m every time.

Saj [36] claims to have proved that their model of reinforcement learning performs better than a classical PID controller for landing UAV's autonomously, even with relatively strong winds. Although the results defend the claim, there is no variety to the test environments to make the claim irrefutable.

3 Methods

3.1 Custom fiducial marker

The CFM has been designed from scratch and was inspired by the classic helipad marker usually present in helicopter landing sites, shown in Figure 3.1. It was designed with simplicity and a good detection range in mind, at the expense of encoding. It consists of an outer ring for long-distance detection, an inner ring to increase the robustness of detection and decrease the number of false positives, and a smaller inner ring for when the camera is too close to the marker to be able to see the other two rings. The relative dimensions of the rings' inner and outer radii are shown in Table 3.1 and the fiducial itself is shown in Figure 3.2.

Figure 3.1: Classic helipad pattern.

The detection algorithm for this marker was also built from the ground up and is based mostly on ellipse fitting and subsequent filtering. It can be split in three stages: ellipse filtering; detecting rings; detecting the marker based on ring matching.

Although the marker is composed of circular rings, if perspective projection is assumed, the distorted circumferences are elliptical. Hence, the ellipse filtering stage.

Table 3.1: Dimensions of the CFM relative to x, the inner radius of the inner ring.

Ring	Inner radius	Outer radius
Outer	$40x$	$50x$
Middle	$15x$	$25x$
Inner	x	$3x$

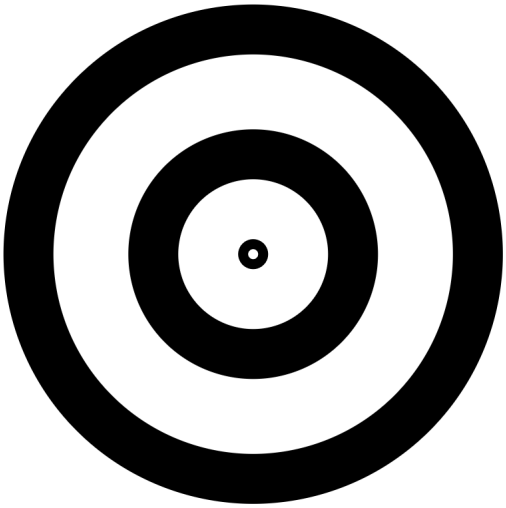


Figure 3.2: Custom fiducial marker.

3.1.1 Pre-processing

Before entering the three-stage detection pipeline, an image must first be pre-processed in order to get better results in the following stages.

The first step is applying a bilateral filter to the image. This filter removes noise, preservers edges and is non-linear. It replaces each pixel with a weighted mean of the intensities of the neighbouring pixels. The weights typically follow a Gaussian distribution, but other distributions can be used. The formal definition of the bilateral filter is presented in Equation 3.1, below,

$$I_{\text{filterred}}(p) = \frac{1}{w_n} \sum_{p_i \in N} w_r \left(\|I(p_i) - I(p)\| \right) \cdot w_s(\|p_i - p\|) \cdot I(p_i) \tag{3.1}$$

$$w_n = \sum_{p_i \in N} w_r(\|I(p_i) - I(p)\|) \cdot w_s(\|p_i - p\|) \tag{3.2}$$

Where

- $I_{\text{filterred}}$ is the output, i.e., the filtered image;
- w_n is the normalisation weight, computed with the range and spatial kernels;

- p is the coordinates of the current pixel;
- N is the neighbourhood of pixels around p;
- p_i is the pixel of N which is currently being accounted for;
- w_r is the range kernel, which assigns more weight to pixels closer to p in intensity;
- w_s is the spatial kernel, which assigns more weight to pixels spatially closer to p;
- I is the input image.

OpenCV's implementation, `bilateralFilter`[1], utilises Gaussian distributions for w_r and w_s, and therefore takes as inputs σ_r and σ_s, the standard deviations of the Gaussian distributions for the range and spatial kernel, respectively. For this purpose, this function was called with $\sigma_r = \sigma_s = 175$ in 5-by-5 neighbourhoods.

The next step is binarising the image, i.e., replacing every pixel with either a 0 or the desired maximum value, depending on whether they meet a certain condition. The binarisation method chosen for this detector is a simple threshold binarisation, in which pixels below the intensity threshold are set to 0 and the others are set to the desired maximum value, as described in Equation 3.3.

$$I_{\mathrm{b}}(p) = \begin{cases} 0, & I(p) < T \\ M, & I(p) \geq T \end{cases} \tag{3.3}$$

Where M is the desired maximum value and T is the fixed threshold value. Since the image is in a single-channel 8-bit format, the maximum value of a pixel is $2^8 - 1 = 255$ and that is the value chosen for M.

Obtaining a binary image is crucial to the following step, finding the contours in the binary image, since it provides much better accuracy. This process is handled by OpenCV's `findContours`[2] function, which takes as inputs the source image, the contour retrieval mode and the contour approximation method. The retrieval mode determines whether the output list containing the contours organises them hierarchically and how it organises them. It was decided to use the flag `RETR_LIST`, which returns the contours without hierarchical

[1] See https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed

[2] See https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0

organisation. The contour approximation modes are used to compress contours, which are essentially lists of coordinates of the pixels belonging to said contour. For instance, if some points of the contour all fall in a straight line arrangement, only the end points are kept, in order to save memory. Since the UAV's on-board computer has limited resources, the chosen flag was `CHAIN_APPROX_SIMPLE`, which works as previously mentioned.

Now that the image has been properly readied for the detection pipeline, it is fed into the first stage, ellipse filtering.

3.1.2 Ellipse filtering

The purpose of the first stage is determining which contours returned from pre-processing are valid ellipses. The filter iterates over each element of the contour list and checks a number of conditions to determine if it is a valid ellipse.

Firstly, if the contour has 4 or less points, it is discarded. Due to the `CHAIN_APPROX_SIMPLE` flag, a contour with four points is a quadrilateral, which is not detailed enough to be considered an ellipse. Following the same logic, contours with less than four points are also invalid.

Then, an ellipse is fitted to the contour using OpenCV's `fitEllipse`[3] function, which, in Python, returns a tuple containing the coordinates in pixels of the centre of the ellipse, x and y, a tuple with the lengths in pixels of its minor and major axes, a and b and its orientation in degrees, ϕ, and the index of its contour in the contour list, c. To ensure the contour is elliptical, its area must be the same of the fitted ellipse. The area of the fitted ellipse is computed with Equation 3.4.

$$A_{\text{ellipse}} = \frac{ab\pi}{4} \tag{3.4}$$

Using the area of the fitted ellipse, the inferior and superior limits of the interval within which the area of the contour, A_{contour}, must lie, $A_{\min}$ and $A_{\max}$, are computed using equations 3.5 and 3.6. The area of the contour is computed with OpenCV's `contourArea`[4] function.

$$A_{\min} = A_{\text{ellipse}}(1 - t_{\text{area}}) \tag{3.5}$$

$$A_{\max} = A_{\text{ellipse}}(1 + t_{\text{area}}) \tag{3.6}$$

[3]See `https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#gaf259efaad93098103d6c27b9e4900ffa`

[4]See `https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html`

Algorithm 1: Ellipse filter.

Input: List of image contours, C

Output: List of valid ellipses, E

$t_{\text{area}} \leftarrow 0.05$

$E = [\,]$

for $C_i \in C$ **do**

  if $\text{card}\,(C_i) > 4$ **then**

    $[x, y, a, b, \phi, c] \leftarrow \texttt{fitEllipse}(C_i)$

    $A_{\text{ellipse}} \leftarrow \frac{ab\pi}{4}$

    $A_{\min} \leftarrow A_{\text{ellipse}}(1 - t_{\text{area}})$

    $A_{\max} \leftarrow A_{\text{ellipse}}(1 + t_{\text{area}})$

    $A_{\text{contour}} \leftarrow \texttt{contourArea}$

    if $A_{\text{ellipse}} \in [A_{\min}, A_{\max}]$ **then**

      $E.\text{append}([x, y, a, b, \phi, c])$

    end

  end

end

Where t_{area} is the relative area tolerance, which is set to 0.05. If the contour's area is outside the $[A_{\min},\ A_{\max}]$ interval, it is discarded. This is the final condition the candidate contours must meet and the end of the ellipse filtering stage.

The filter is described in Algorithm 1. Each member of the output list of valid ellipses contains the coordinates of the centre of the ellipse, its major and minor axes lengths, its orientation and the index of the contour which delimits it. The index identifies the contour in the input list.

3.1.3 Ring detection

Before starting the ring detection stage, the number of elements of E is checked and, if the list doesn't contain at least two elements, the current image is discarded and it is considered as containing no markers.

In order to determine which pairs of ellipses in E are valid rings, the algorithm must iterate over every combination of ellipses. The list of all combinations of two ellipses is E^2.

In order for a ring to be valid it must pass the following checks:

Algorithm 2: Checking if two ellipses are concentric.

Input: Coordinates of the centres of the two ellipses, c_1 and c_2

Output: Boolean indicating if the ellipses are concentric (`true`)

Function *twoAreConcentric*(c_1, c_2)**:**

    if $\|c_1 - c_2\| \leq t_{concentric}$ **then**
        return `true`
    else return `false`

end

- Concentricity: the ellipses must be concentric;
- Eccentricity: the ellipses must have the same eccentricity;
- Orientation: the ellipses must have the same orientation;
- Type: the ring must be of either the outer, middle or inner types;
- Blackness: the ring must have at least a certain percentage of black points;

Concentricity

In order to meet this condition, the centres of the ellipses in the pair must coincide. Since the camera sensor is prone to noise, a tolerance is defined to account for innate errors. This tolerance, $t_{concentric}$, is set to 10 pixels. The condition is shown in Algorithm 2.

Eccentricity

The non-negative real number that uniquely identifies the shape of a conic section is called its eccentricity, ε. More formally, if one considers a conic section as the intersection of a plane with a double-napped cone, as shown in Figure 3.3, its eccentricity can be defined as shown in Equation 3.7.

$$\varepsilon = \frac{\sin\beta}{\sin\alpha},\ \alpha \in \left]0, \frac{\pi}{2}\right[,\ \beta \in \left[0, \frac{\pi}{2}\right] \tag{3.7}$$

Since this stage always deals with ellipses, the eccentricity formula can be replaced with the ellipse eccentricity formula presented in Equation 3.8. Similarly to the concentricity condition, a tolerance $t_{eccentricity}$ with a value of 0.2 is given to this check.

$$\varepsilon = \frac{\sqrt{b^2 - a^2}}{b},\ b \geq a \tag{3.8}$$

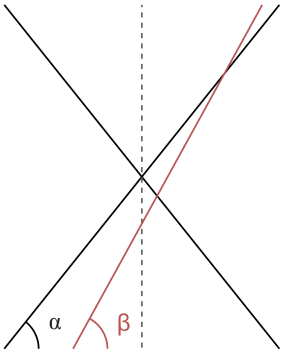


Figure 3.3: Simplified 2D diagram of a conic section seen as the intersection of a double-napped cone, in black, and a plane, in red.

Algorithm 3: Checking if two ellipses have similar eccentricities.

Input: Lengths of the minor and major axes of both ellipses, a and b

Output: Boolean indicating if the ellipses have the same eccentricity (`true`)

Function *`eccentricityIsEqual`*(a, b)**:**

$\varepsilon \leftarrow \frac{\sqrt{b^2-a^2}}{b}$

if $|\varepsilon_1 - \varepsilon_2| \leq t_{eccentricity}$ **then**

return `true`

else

return `false`

end

end

For the purposes of this condition, the absolute value of the difference between the eccentricities of both ellipses is taken, i.e., $|\varepsilon_1 - \varepsilon_2|$. Similarly to the concentricity condition, a tolerance is given to this check. Its value is 0.2. The check is described in Algorithm 3

Orientation

When using OpenCV, the orientation of a fitted ellipse is defined as the angle between the horizontal axis and the minor axis, increasing in a clockwise fashion, unlike the standard in mathematics. In order to check if the ellipses meet this condition, the difference between their orientations, $\Delta\phi$, is taken. Since it is convenient to constrain the difference of angles to the $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ interval, instead of simply taking their arithmetic difference, the formula in Equation 3.9 is used, because the function `atan2` from Python's `math` module outputs an angle in the desired interval. The condition check is described in Algorithm 4.

Algorithm 4: Checking if two ellipses have the same orientation.

Input: Orientation of both ellipses, ϕ_1 and ϕ_2

Output: Boolean indicating if the ellipses have the same orientation (`true`)

Function *twoHaveEqualOrientation*(ϕ_1, ϕ_2)**:**

$\Delta\phi \leftarrow \arctan \frac{\sin(\phi_1 - \phi_2)}{\cos(\phi_1 - \phi_2)}$

if $|\Delta\phi| \leq t_{orientation}$ **then**

return `true`

else

return `false`

end

end

$$\Delta\phi = \arctan \frac{\sin(\phi_1 - \phi_2)}{\cos(\phi_1 - \phi_2)} \tag{3.9}$$

Like in the previous two conditions, this one has a tolerance $t_{orientation}$ of 20°.

Type

The three rings types are: outer ring; middle ring; inner ring. Their type is determined by the ratio between the radii of the inner and the outer circumferences which delimit the ring. Looking at Table 3.1, one can see the ratios are $\frac{4}{5}$, $\frac{3}{5}$ and $\frac{1}{3}$, respectively.

Since at times the ring suffers from projection distortion and becomes elliptical, the evaluation of the ratio of radii must be changed to the evaluation of the ratio between the lengths of the major axes of both ellipses, r_b, and the ratio between the lengths of the minor axes of both ellipses, r_a. This is shown in Equations 3.10 and 3.11.

$$r_a = \frac{a_1}{a_2},\ a_1 < a_2 \tag{3.10}$$

$$r_b = \frac{b_1}{b_2},\ b_1 < b_2 \tag{3.11}$$

As in all the other checks, the type check has a tolerance t_{type} of 0.1 for all types of ring. If both r_a and r_b are within the tolerance for the same type of ring, the ring is considered of that type. The check is described in Algorithm 5.

Algorithm 5: Evaluating the type of a ring.

Input: Lengths of the minor and major axes of the ring's ellipses a and b

Output: Value indicating the type of the ring: outer (0), middle (1), inner (2) or invalid (-1)

Function *getRingType*(a, b)**:**

$r_a \leftarrow \frac{a_1}{a_2}$

$r_b \leftarrow \frac{b_1}{b_2}$

if $|r_a - \frac{4}{5}| < t_{type} \ \wedge \ |r_b - \frac{4}{5}| <_{type}$ **then**

return 0

else if $|r_a - \frac{3}{5}| < t_{type} \ \wedge \ |r_b - \frac{3}{5}| < t_{type}$ **then**

return 1

else if $|r_a - \frac{1}{3}| < t_{type} \ \wedge \ |r_b - \frac{1}{3}| < t_{type}$ **then**

return 2

else

return -1

end

end

Blackness

In order to pass the blackness check, the ring must have a percentage of black points, B, greater than or equal to $t_{black\%} = 0.8$. A pixel is considered black if its intensity value is less than the black threshold, t_B, which is described in Equation 3.12. The intuition behind the formula follows this logic: it is desired that the black threshold sit 30% of the way between the darkest and the brightest pixel in the image; therefore, the difference between said pixels is taken and 30 % of that is computed; then, the value of the darkest pixel must be added to the result in order to obtain a value 30 % of the way between the reference pixels. This check does not have a tolerance. The blackness check is shown in Algorithm 6.

$$t_B = 0.3 \cdot (\max I - \min I) + \min I \tag{3.12}$$

After checking for all conditions

After meeting all conditions, the ring is considered valid and its centre and orientation are computed by taking the mean centre, c_R, and orientation, ϕ_R, of the ellipses that make

Algorithm 6: Determining if a ring has enough black points.

Input: List of ellipses that make up the ring E, Image I, List of contours C

Output: Boolean indicating if the ring defined by E in the image I has enough black points

Function *isBlack*$(E,\ I,\ C)$**:**

$t_B \leftarrow 0.3 \cdot (\max I - \min I) + \min I$

$p \leftarrow C$.`getRingPointsCoordinates`()

$n_p \leftarrow \text{card}(p)$

$n_B \leftarrow \text{card}(I(p_i) < t_b),\ p_i \in p$

if $\frac{n_B}{n_p} \leq t_{black\%}$ **then** return `false`

return `true`;

end

up the ring. Its type, centre coordinates and orientation are appended to the list of valid rings, R. In order for the ring to be considered black enough The ring detection algorithm is summarised in Algorithm 7.

Algorithm 7: Ring detection.

Input: Grayscale image I, List of valid ellipses E, List of image contours C

Output: List of valid rings, R

$E^2 \leftarrow \text{combinations}(E, 2)$

$R = [\,]$

for $E_i^2 \in E^2$ **do**

$[x, y, a, b, \phi, c] \leftarrow E_i^2$

if not `twoAreConcentric`$([x_1, y_1],\ [x_2, y_2])$ **then** continue

if not `eccentricityIsEqual`$(a,\ b)$ **then** continue

if not `twoHaveEqualOrientation`$(\phi_1,\ \phi_2)$ **then** continue

if `getRingType`$(a,\ b) = -1$ **then** continue

if not `isBlack`$(E_i^2,\ I,\ C_c)$ **then** continue

$c_R = \left[\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right]$

$\phi_R = \frac{\phi_1+\phi_2}{2}$

R.append($[$`ringType`$, c_R, \phi_R]$)

end

3.1.4 Marker detection based on ring matching

This is the final stage of the marker detection process and it is skipped if no rings were detected in the last stage. This algorithm matches the rings detected in the previous stage and checks if any of the combinations result in a valid marker.

The first condition the algorithm checks for is the number of rings that have been detected. Although the marker is comprised of three rings, if the camera is too far away or too close to it, it will see less than three rings and thus not all rings need to be detected for a combination of rings to be considered a valid marker.

If R contains a single ring, it is immediately considered a valid marker and a ROS `geometry_msgs/Point` message is published with the pixel coordinates of the centre of the ring. If R contains two rings, the rings must have different types to be considered a valid marker. They must not be the inner and outer ring as well. I.e., the valid two-ring combinations are outer-middle and middle-inner. They must also be concentric and have the same orientation. If the two rings meet these conditions, the mean of their centres in pixels is taken and published as the marker's centre. If R contains three rings, they must all be of different types, be concentric and have the same orientation. If they do, the mean of their

Algorithm 8: Checking if three ellipses are concentric.

Input: Coordinates of the centres of the three ellipses, c_1, c_2 and c_3

Output: Boolean indicating if the ellipses are concentric (`true`)

Function *threeAreConcentric*(c_1, c_2, c_3)**:**

$d_{12} \leftarrow |c_1 - c_2|$

$d_{13} \leftarrow |c_1 - c_3|$

$d_{23} \leftarrow |c_2 - c_3|$

if $\max(d_{12}, d_{13}, d_{23}) \leq t_{concentric}$ **then**

return `true`

else return `false`

end

Algorithm 9: Checking if three ellipses have the same orientation.

Input: Orientation of the ellipses, ϕ_1, ϕ_2 and ϕ_3

Output: Logical value indicating if the ellipses have the same orientation (`true`)

Function *threeHaveEqualOrientation*(ϕ_1, ϕ_2, ϕ_3)**:**

$\Delta\phi_{12} \leftarrow \arctan \frac{\sin(\phi_1 - \phi_2)}{\cos(\phi_1 - \phi_2)}$

$\Delta\phi_{13} \leftarrow \arctan \frac{\sin(\phi_1 - \phi_3)}{\cos(\phi_1 - \phi_3)}$

$\Delta\phi_{23} \leftarrow \arctan \frac{\sin(\phi_2 - \phi_3)}{\cos(\phi_2 - \phi_3)}$

if $\max(\Delta\phi_{12}, \Delta\phi_{13}, \Delta\phi_{23}) \leq t_{orientation}$ **then**

return `true`

else return `false`

end

centres in pixels is taken and published as the marker's centre. The concentricity, orientation and type checks are shown in Algorithms 8-10.

If R contains more than three rings, the evaluation is more complex. The marker in the image could be comprised of either two or three rings. Thus, the algorithm must iterate over every combination of two rings, R^2, to check if any of them are a valid marker. If not, it then iterates over every combination of three rings, R^3, to look for the marker. If none is found again, the image is discarded. Similarly to when R has two rings, the rings in R_i^2 must be of different types, one of them must be the middle ring and they must be concentric and have the same orientation. The three rings of R_i^3 must all be of different types, be concentric and have the same orientation. A summary of the one-ring, two-ring, three-ring

Algorithm 10: Checking if three ellipses have different types.

Input: Orientation of the ellipses, $\texttt{ringType}_1$, $\texttt{ringType}_2$ and $\texttt{ringType}_3$

Output: Logical value indicating if all the ellipses have different types (`true`)

Function *threeAreOfDifferentTypes(*$\texttt{ringType}_1$, $\texttt{ringType}_2$, $\texttt{ringType}_3$*)*:

```
allDifferent ← true
if ringType_1 = ringType_2 then return allDifferent ← false
if ringType_1 = ringType_3 then return allDifferent ← false
if ringType_2 = ringType_3 then return allDifferent ← false
return allDifferent
```

end

Algorithm 11: Marker detection matching when R contains one ring.

Input: Ring R

Output: Centre of the detected marker c_m

Function *getCentreOneRing(*R*)*:

$c_m \leftarrow R.c_R$

end

and more-than-three-rings scenarios is presented in Algorithms 11-14.

After filtering out the combinations that are valid markers, if there is more than one valid combination, the first one is considered the marker and the rest are discarded. Subsequently, the mean of the centres of the rings that compose the marker is taken and published. If no marker was detected, the centre's coordinates are set to -1 and they are interpreted as invalid.

This marker detection algorithm is laid out in Algorithm 15.

Algorithm 12: Marker detection matching when R contains two rings.

Input: List of rings R

Output: Centre of the detected marker c_m

Function *getCentreTwoRings(R)*:

$[\texttt{ringType}, c, \phi] \leftarrow R$

if $\texttt{ringType}_1 = \texttt{ringType}_2$ **then** return $c_m \leftarrow [-1, -1]$

if not `twoAreConcentric`$(c_1,\ c_2)$ **then** return $c_m \leftarrow [-1, -1]$

if not `twoHaveEqualOrientation`$(\phi_1,\ \phi_2)$ **then** return $c_m \leftarrow [-1, -1]$

$c_m \leftarrow \frac{c_1+c_2}{2}$

end

Algorithm 13: Marker detection matching when R contains three rings.

Input: List of rings R

Output: Centre of the detected marker c_m

Function *getCentreThreeRings(R)*:

$[\texttt{ringType}, c, \phi] \leftarrow R$

if not `threeAreOfDifferentTypes`$(\texttt{ringType}_1, \texttt{ringType}_2, \texttt{ringType}_3)$ **then** return $c_m \leftarrow [-1, -1]$

if not `threeAreConcentric`$(c_1,\ c_2,\ c_3)$ **then** return $c_m \leftarrow [-1, -1]$

if not `threeHaveEqualOrientation`$(\phi_1,\ \phi_2,\ \phi_3)$ **then** return $c_m \leftarrow [-1, -1]$

$c_m \leftarrow \frac{c_1+c_2+c_3}{3}$

end

Algorithm 14: Marker detection matching when R contains four or more rings.

Input: List of rings R

Output: Centre of the detected marker c_m

Function *getCentreFourOrMoreRings*(R)**:**

markers $= [\ ]$

$R^2 \leftarrow \text{combinations}(R, 2)$

for $R_i^2 \in R^2$ **do**

$[\texttt{ringType}, c, \phi] \leftarrow R_i^2$

if $\texttt{ringType}_1 = \texttt{ringType}_2$ **then** continue

if not twoAreConcentric$(c_1,\ c_2)$ **then** continue

if not twoHaveEqualOrientation$(\phi_1,\ \phi_2)$ **then** continue

markers.append$(\frac{c_1+c_2}{2})$

end

$R^3 \leftarrow \text{combinations}(R, 3)$

for $R_i^3 \in R^3$ **do**

$[\texttt{ringType}, c, \phi] \leftarrow R_i^3$

if not threeAreOfDifferentTypes$(\texttt{ringType}_1,\ \texttt{ringType}_2,\ \texttt{ringType}_3)$ **then** continue

if not threeAreConcentric$(c_1,\ c_2,\ c_3)$ **then** continue

if not threeHaveEqualOrientation$(\phi_1,\ \phi_2,\ \phi_3)$ **then** continue

$c_m \leftarrow \frac{c_1+c_2+c_3}{3}$

end

if card(markers) < 1 **then**

$c_M \leftarrow [-1, 1]$

else if card(markers) $= 1$ **then**

$c_M \leftarrow$ markers

else

$c_M \leftarrow \texttt{markers}_1$

end

end

Algorithm 15: Marker detection based on ring matching.

Input: List of valid rings R

Output: Centre of the detected marker c_m

if $\text{card}(R) = 1$ **then** $c_m \leftarrow$`getCentreOneRing` (R)

else if $\text{card}(R) = 2$ **then** $c_m \leftarrow$`getCentreTwoRings` (R)

else if $\text{card}(R) = 3$ **then** $c_m \leftarrow$`getCentreThreeRings` (R)

else $c_m \leftarrow$`getCentreFourOrMoreRings` (R)

3.2 Autonomous landing

The autonomous landing algorithm developed for this work is based on state-machine architecture and uses a proportional-integral (PI) controller to guide the UAV to the landing position.

The state-machine has three states: mission, descending and landing, and is shown in Figure 3.4. The mission state pertains to whatever mission has been assigned to the aircraft. The descending state starts when the mission is finished and activates the landing controller to align the UAV with the marker and descend towards it. Finally, the landing state starts when the UAV is in the descending state and reaches an altitude relative to the marker inferior to 80 cm. This value is stored in t_{alt}.

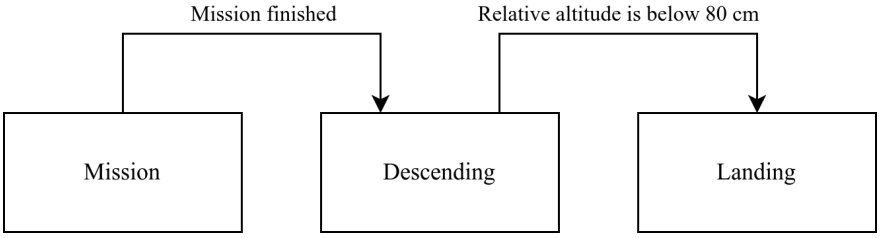


Figure 3.4: State-machine architecture of the landing algorithm.

The descending state supports wind simulation via random velocity controls.

3.2.1 Landing algorithm

This algorithm is entirely implemented within a single Pythons script. The first step when starting the landing controller node, `/landing_controller`, is initialising the necessary constants, variables, publishers and subscribers. These are: the alignment tolerance, the PI controller's constants and errors, the velocity limits for the UAV, the relative altitude threshold for landing and the wind simulation parameters.

This algorithm takes as an input the centre of the detected marker in the image and attempts to align the UAV with it. However, a tolerance must be given to the error between the centre of the image and the centre of the marker due to vibrations, errors in the estimation of the centre of the marker and wind noise. This tolerance is defined as a circular area concentric with the image, within which the error between the centre of the image and the centre of the marker is considered 0. The alignment tolerance, $t_{alignment}$, is the radius of the circular area and has a value of 300 pixels. The value does not need to be dynamic because, as the camera approaches the ground, the area on the ground projected inside the

circular area becomes smaller, effectively reducing the alignment tolerance in the real world and making the UAV approach the marker.

The type of controller chosen for this application is a PI controller, whose command signal, $u(t)$, is describer described in Equation 3.13.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau \tag{3.13}$$

Where K_p is the proportional gain, $e(t)$ is the error at the time instant instant t and K_i is the integral gain. The general intuition behind this controller is that the proportional component works in the present by adjusting the command signal to the current error and responds in greater magnitude to greater errors, while the integral component, due to the integral term, "remembers" the past errors of the system and adjusts the command signal according to them. However, since the controller is implemented digitally, as are its measurements, it must take the discrete form, presented in Equation 3.14.

$$u[k] = K_p e[k] + K_i \sum_{n=0}^{k} e[n] \tag{3.14}$$

Where k is the current discrete time step. The intuition is the same as for the continuous case, but with a sum in the place of the integral. The chosen values of K_p and K_i are 10^{-6} and $7 \cdot 10^{-7}$, respectively. This controller only applies to the x and y axes, as the z axis is controlled by a simple proportional controller with a gain K_{pz} of $-5 \cdot 10^{-4}$

The constants associated with velocity control are the lower and upper limits for v_x, v_y and v_z, the velocities of the UAV in each of the three Cartesian axes. These limits v_{min} and v_{max} are, respectively, $-0.5\,\mathrm{m/s}$ and $0.5\,\mathrm{m/s}$ for both v_x and v_y. For v_z the limits v_{zmin} and v_{zmax} are $10^{-5}\,\mathrm{m/s}$ and $0.5\,\mathrm{m/s}$.

Unfortunately, due to technical issues it was not possible to use Gazebo's native wind simulation. In its stead, random velocity commands were used to simulation wind noise. The wind values were approximated with a discrete low-pass filter and a random term following a Gaussian distribution. This approximation is shown in Equation 3.15.

$$v_w[k] = \lambda v_w[k-1] + (1-\lambda) v_{rand}[k] \tag{3.15}$$

Where $v_w[k]$ is the simulated wind velocity at discrete time step k, λ is a value between 0 and 1 and describes how significant the random component is in the value of $v_w[k]$, and v_{rand} is the random component which follows a Gaussian distribution. The value of λ is 0.9 and

Algorithm 16: Wind noise simulation.

Input: Wind noise in the last step, $v_w[k-1]$

Output: Wind noise for the current step, $v_w[k]$

Function *getWindVelocity*$(v_w[k-1])$**:**

$v_{rand} \leftarrow \mathcal{N}(\mu_w, \sigma_w)$

$v_w[k] \leftarrow \lambda v_w[k-1] + (1-\lambda)v_{rand}$

return $v_w[k]$

end

the distribution that the random component follows has a mean μ_w of 0 m/s and a standard deviation σ_w of 0.1 m/s.

After the initialisation phase, the script connects to the FCU through the `iq_gnc` application programming interface (API). The UAV is now ready to fly.

When the landing algorithm switches from the mission state to the descending state, the landing controller starts accepting messages from the CFM detection node with the centre of the marker in the image. Upon receiving a message, it will compute the values of wind noise for this iteration according to Algorithm 16.

Now, if the message received contains a point with the x and y coordinates set to -1, the algorithm does not feed it to the controller and instead simply publishes the wind velocity as noise in the system. If the point is valid, i.e., neither the x or the y coordinate is set to -1, the algorithm feeds the point to the controller, which computes the error and the corresponding velocities to correct it.

The error is defined as the Euclidean distance in pixels from the center of the marker in the image, c_M, to the center of the image, c_I in each of the x and y coordinates. However, if the Euclidean distance (not separated into the x and y coordinates) is less than $t_{alignment}$, it is instead set to 0. If the error is 0 and the altitude of the UAV relative to the marker is less than t_{alt} the API sends a request to the FCU to enter landing mode. In landing mode, the FCU guides the aircraft autonomously downward while maintaining it's x and y position. Only when the error is 0 does the controller act upon the z axis by computing the error $e_z[k]$. When the error is non-zero, the altitude of the UAV is considered 0 and thus ignored by the controller.

If the UAV does not enter landing mode, the controller converts the errors in the image's x and y coordinate system to the FCU's coordinate system. OpenCV's coordinate system is ESD and the FCU's coordinate system is ENU, as shown in Figure 3.5, which means the

Algorithm 17: Limiting a value to its lower and upper limits.

Input: Value to limit a, Lower limit a_m, Upper limit a_M

Output: Limited value, a'

Function *limitValue*(a, a_m, a_M)**:**

$a' \leftarrow \max(\min(a, a_M), a_m)$

return a'

end

conversion consists of simply inverting the y and z coordinates. After computing the errors, they are added to their respective integral running sums, $\sum_{n=0}^{k} e_x[n]$ and $\sum_{n=0}^{k} e_y[n]$, for the purposes of PI control.

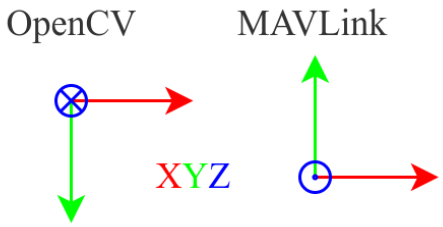


Figure 3.5: Diagram comparing OpenCV's ESD coordinates and the FCU's ENU coordinates.

Finally, the velocity commands are computed according to Equation 3.14 and limits to their minimum and maximum values using Algorithm 17.

Then, the wind noise is added to limited values and the result is published to `/mavros/setpoint_velocity/cmd_vel`.

The landing algorithm is summarised in Algorithm 18. The image width I_w is 1296 and the image height I_h is 972.

Algorithm 18: Landing algorithm.

Input: Centre of the marker in the image c_M

Output: Vector with the velocity commands, u

drone$\leftarrow$`iq_gnc`.initialiseAPI()

$v_w[k] \leftarrow$`getWindVelocity(`$v[k-1]$`)`

$[c_{Mx}, c_{My}] \leftarrow c_M$

if $c_{Mx} = -1 \ \vee \ c_{My} = -1$ **then**

  $[v_x, v_y, v_z] \leftarrow v_w[k]$

  return $u \leftarrow [v_x, v_y, v_z]$

else

  $e[k] \leftarrow \|c_M - c_I\|$

  if $e[k] < t_{alignment}$ **then** $e[k] \leftarrow 0$

  if $e[k] < t_{alignment}$ **then**

    $e_z[k] \leftarrow$drone.altitude()

    if $e_z[k] < t_{alt}$ **then** drone.setState(landing)

  else

    $e_z[k] \leftarrow 0$

  end

  $[v_{wx}, v_{wy}, v_{wz}] \leftarrow v_w[k]$

  $e_x[k] \leftarrow c_{Mx} - \frac{I_w}{2}$

  $e_y[k] \leftarrow \frac{I_h}{2} - c_{My}$

  $u_x \leftarrow$`limitValue(`$K_p e_x[k] + K_i \sum_{n=0}^{k} e_x[n],\ v_{min},\ v_{max}$`)`$+v_{wx}$

  $u_y \leftarrow$`limitValue(`$K_p e_y[k] + K_i \sum_{n=0}^{k} e_y[n],\ v_{min},\ v_{max}$`)`$+v_{wy}$

  $u_< \leftarrow$`limitValue(`$K_{pz} e_z[k],\ v_{zmin},\ v_{zmax}$`)`$+v_{wz}$

  return $u \leftarrow [u_x, u_y, u_z]$

end

4 Experimental Work

4.1 Hardware

Figures 4.1 and 4.2 show photographs of the UAV along with labelled components.

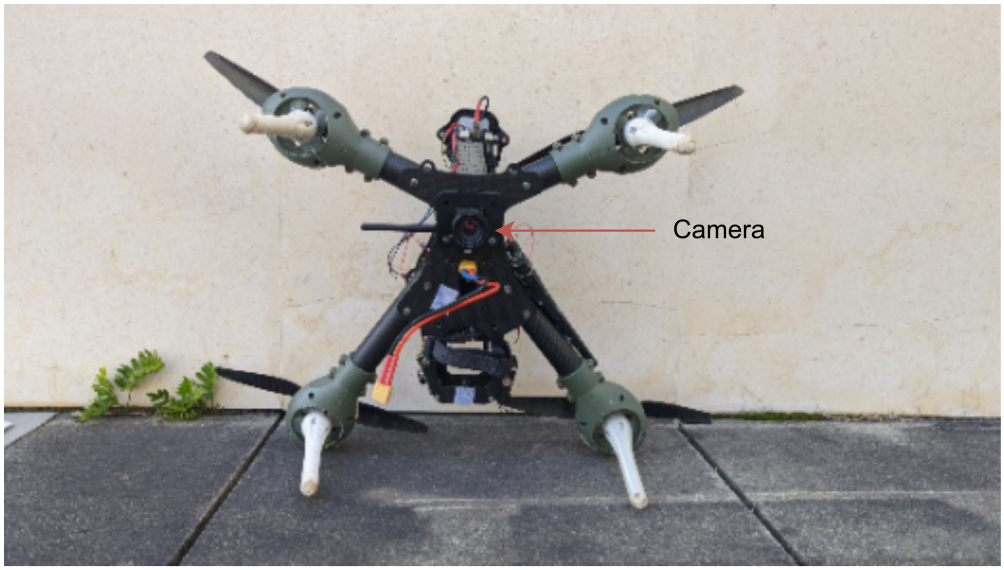


Figure 4.1: Photograph of the UAV showing its camera.

The UAV used in this work is a modified SkyHero Little Spyder equipped with a Pixhawk 4 FCU, an OrangePi PC Plus single-board computer (SBC), a Basler dart camera with an 8 mm Computar lens, an RTK-GNSS receiver with one antenna, a FrSky radio receiver and a WiFi antenna. A diagram of all these pieces of hardware is shown in Figure 4.3.

The Pixhawk 4[1] FCU comes equipped with an STM32F765 as the main processor, an STM32F100 input-output (IO) processor, an ICM-20689 IMU, an IST8310 magnetometer, and an MS5611 barometer.

The RTK-GNSS system is comprised of a u-blox ANN-MB-00-00[2], which supports GPS (L1, L2), GLONASS (G1, G2), BeiDou (B1, B2), Galileo (E1, E5b), QZSS (L1, L2) and

[1] See `https://docs.px4.io/main/en/flight_controller/pixhawk4.html`

[2] See `https://www.u-blox.com/en/product/ann-mb-series?legacy=Current`

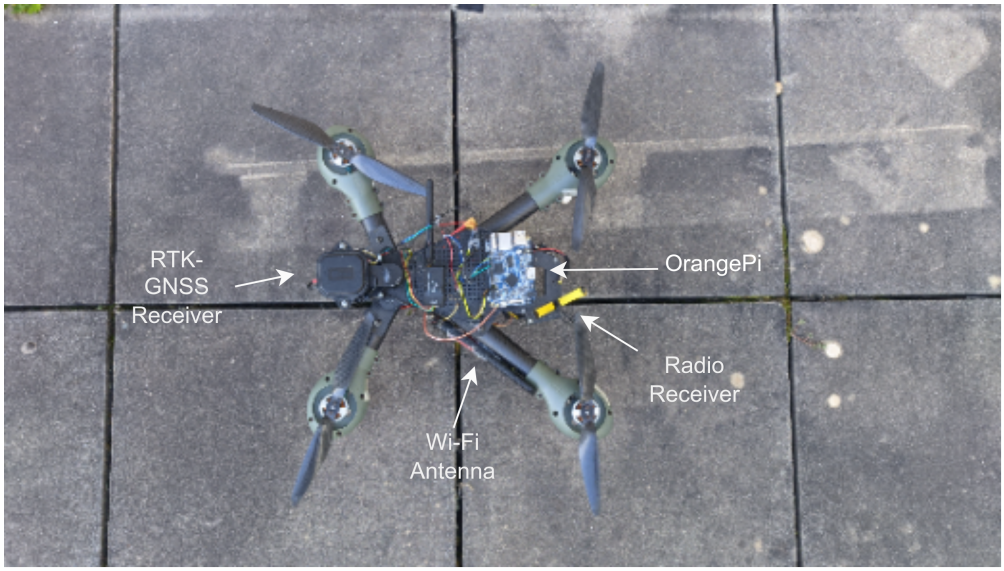


Figure 4.2: Photograph of the UAV with hardware components labelled.

SBAS (WAAS, EGNOS, MSAS and GAGAN) constellations, and a DROTEK DP0601[3] receiver with a horizontal precision of 1 cm plus 1 mm per km away from the base station and a vertical precision of 1 cm+1 ppm R50. A vertical precision of 1 cm+1 ppm R50 means a radius of 1 cm at 50 % circular error probability with data collected over 24 hours.

The OrangePi PC Plus[4] SBC is an open-source platform with and H3 Quad-core Cortex-A7 H.265/HEVC 4K CPU, 1 GiB of DDR3 synchronous dynamic random-access memory (SDRAM), a TF card slot (limited to a maximum of 32 GiB of storage) and three universal serial bus (USB) 2.0 host ports.

The camera is a Basler daA2500-14uc with a 1/2.5" CMOS sensor with a resolution of 2592-by-1944 pixels (5 MP) which provides coloured images and typically consumes 1.3 W. The lens mounted on the camera is a Computer M0814-MP2[5], which has a fixed focal length of 8 mm, its aperture set to its widest setting, f/1.4, and the focus ring set to near-infinity.

The radio receiver is a FrSky X8R[6] which listens at 2.4 GHz and the remote controller is a FrSky Taranis X9D Plus[7].

The WiFi antenna supports both 2.4 GHz and 5 GHz bands and has a gain of 12 dBi. In order to avoid interference with the frequency band used by the radio receiver/controller pair, the antenna is used in the 5 GHz mode.

[3] See `https://www.u-blox.com/en/product/zed-f9p-module`

[4] See `http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-PC-Plus.html`

[5] See `https://computar.com/product/552/M0814-MP2`

[6] See `https://www.frsky-rc.com/product/x8r/`

[7] See `https://www.frsky-rc.com/product/taranis-x9d-plus-2/`

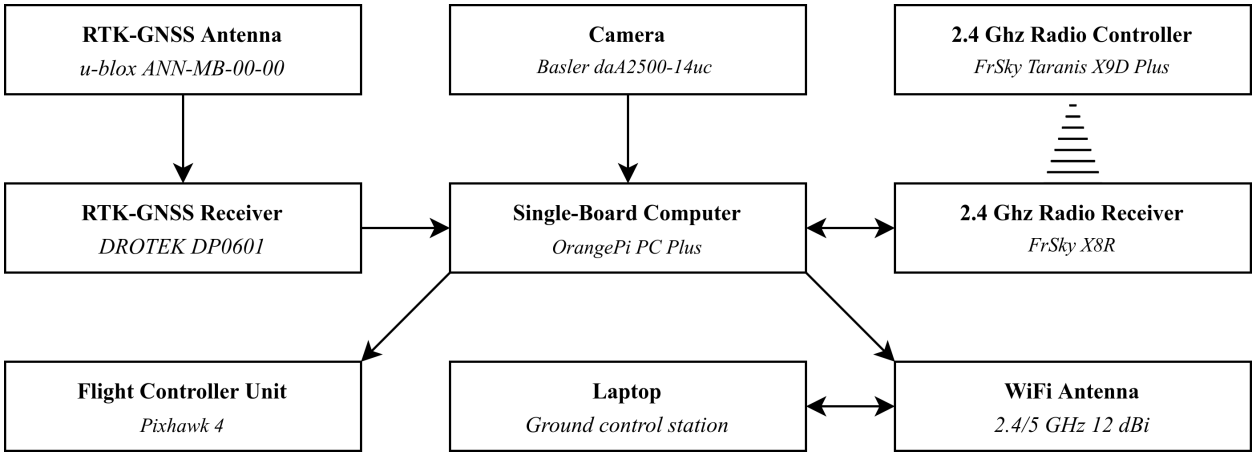


Figure 4.3: Diagram describing the connections between the hardware components.

4.1.1 Camera settings

The Basler daA2500-14uc offers a variety of configurable settings, most of which can be changed in pylon Viewer, Basler's official configuration and diagnostics tool. The most relevant camera settings that were changed for this dissertation were: the binning amount, the shutter mode and the encoding.

Binning amount

Binning is an imaging technique that groups neighbourhoods of pixels into super-pixels. Binning's advantages are more sensitivity to light intensity and higher frame rate due to the reduced resolution. Binning's main disadvantage is the reduced resolution. This setting is further explored in Section 4.3.

Shutter mode

Although there are three common shutter modes – rolling, global and global reset –, this camera only supports rolling shutter and global reset shutter. In rolling shutter, the data acquisition emulates the behaviour of a mechanical shutter and scans the pixel matrix line-by-line. This leads to even lighting across the image (provided the scene does not change abruptly between line scans) but if a subject is moving much faster than the shutter speed, motion artefacts arise. Global reset shutter is a variation of global shutter. Whereas global shutter scans all pixels at once in the same amount of time, global reset starts scanning all the lines at the same time, but reads the matrix a line at a time. This leads to brighter pixels towards the bottom of the image due to the increased exposure time. This is undesirable for this work because the CFM algorithm relies on thresholding images to detect the CFM.

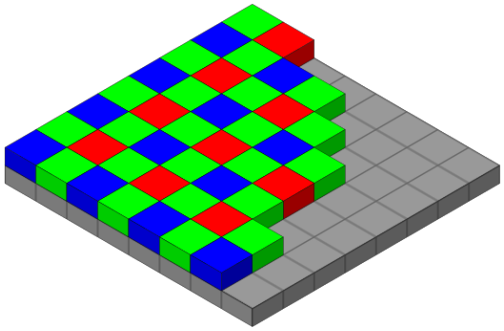


Figure 4.4: Diagram illustrating a Bayer encoding by showing an analogue Bayer filter over a camera sensor. Source: [1].

By having an area of the image significantly brighter than the rest, the thresholding process becomes biased in favour of separating the image into these two areas instead of properly thresholding it. Thus, the selected shutter mode is rolling.

Encoding

Encoding in this context refers to the format in which the data from the camera sensor is stored. The camera offers two encoding: red, green and blue (RGB) and Bayer, both 8-bit. Using RGB encoding, each pixel stores three values, one for each of the red, blue and green colours, which means it has complete colour information but it needs more data to store it. Bayer encoding utilises a Bayer pattern, shown in Figure 4.4, in which each pixel only stores one value, either red, green or blue, according to the element of the filter above it. This allows for colour image, though with incomplete information, but only needs a third of the data that RGB needs. This means a camera operating with Bayer encoding can have triple the frame rate of one operating with RGB encoding, given the same bandwidth. Since this work requires a good frame rate, RGB encoding was discarded and Bayer was used instead.

4.2 Software

The software for this work was developed in Python using the Robot Operating System (ROS)[8] framework. ROS was instrumental due to its support of simulation using Gazebo[9], support of data visualisation using RViz[10], support of data recording and playback using the `rosbag`[11] package and `bag` files, and a number of off-the-shelf packages that integrate different

[8] See `https://www.ros.org`
[9] See `https://gazebosim.org/home`
[10] See `http://wiki.ros.org/rviz`
[11] See `http://wiki.ros.org/rosbag`

pieces of hardware and software. In order for the UAV to be able to fly autonomously and have an autopilot system, MAVLink[12] and ArduPilot[13] were also used. ArduPilot was also essential for simulation, as it enables software in the loop (SITL) simulation.

ROS was used to construct a node network for messaging between the different parts of the system and due to very low workload of transferring code tested in simulation to the UAV. A number of off-the-shelf packages along with a custom one were used both in simulation and in the real world. The utilised off-the-shelf packages are the following:

- Camera driver - `pylon-ros-camera`[14];
- Fiducial marker detection - `aruco_ros`[15] and `stag_ros`[16];
- UAV simulation - `iq_sim`[17];
- Interface between the FCU and ROS - `mavlink`[18] and `mavros`[19].

A custom package, `msc`, was developed by building on top of an existing package, `iq_gnc`[20], which is an API that facilitates development of guided UAV missions using MAVLink. This custom packages contains the nodes listed below. The topics preceded by "Debug" are only published on or subscribed to when in debug mode.

- CFM detection node - `/target`
  - Subscribed topics:
    - `/pylon_camera_node/image_rect` (`sensor_msgs/Image`).
  - Published topics:
    - `/target/centre` (`geometry_msgs/PointStamped`);
    - Debug: `/target/ellipse_debugging` (`/sensor_msgs/Image`);
    - Debug: `/target/ring_debugging` (`/sensor_msgs/Image`);
    - Debug: `/target/target_debugging` (`/sensor_msgs/Image`).

[12]See `https://mavlink.io/en/`
[13]See `https://ardupilot.org/ardupilot/#`
[14]See `https://github.com/basler/pylon-ros-camera`
[15]See `https://github.com/pal-robotics/aruco_ros`
[16]See `https://github.com/usrl-uofsc/stag_ros`
[17]See `https://github.com/Intelligent-Quads/iq_sim`
[18]See `https://github.com/mavlink/mavlink`
[19]See `https://github.com/mavlink/mavros`
[20]See `https://github.com/Intelligent-Quads/iq_gnc`

- Guided mission controller node - `/landing_controller`
  - Subscribed topics:
    - `/pylon_camera_node/image_raw` (`/sensor_msgs/Image`)
    - `/target/centre` (`/geometry_msgs/PointStamped`)
    - `/mavros/local_position/pose` (`/geometry_msgs/PoseStamped`)
  - Published topics:
    - `/mavros/setpoint_velocity/cmd_vel` (`/geometry_msgs/TwistStamped`)

The `/target` node subscribes to the camera images topic and publishes the centre of the detected marker in pixels. The `/landing_controller` node subscribes to the centre of the marker and uses the measurement in a PID closed loop controller to control the approach velocity of the UAV towards the marker.

In order to facilitate debugging, the `/target` node publishes on three image topics and subscribes to one pose topic. Each of three image topics helps in debugging a different part of the CFM detection pipeline (ellipse fitting, ring detection, and marker detection based on ring matching). In order to help with the visualisation of the tolerance in the alignment of the marker with the centre of the image, a ring is drawn the image published on `/target/target_debugging`.

ArUco and STag markers are detected using off-the-shelf ROS packages but AprilTag markers are detected using Python's `apriltag` module. For this purpose a custom node was created. It uses the module to detect AprilTag markers and then compute their centre in the image. It then publishes the centre for the landing controller to subscribe.

Unfortunately, it was not possible to test the landing algorithm in the real world, so it was tested and validated in simulation. Gazebo was chosen to simulate the UAV landing due to how closely related it is to ROS. The UAV model was previously available in the `iq_sim` package and a camera was attached to its bottom, like in the real UAV. One great advantage of simulating the UAV is that Gazebo provides its exact pose, allowing for better evaluation of the performance of the landing algorithm.

4.3 Comparison of fiducial marker detection performance

4.3.1 Objectives

The camera driver allows the user to choose one of four values for the image binning in each direction, from 1 to 4. A value of 1 means using the full resolution in that direction, a value of 2 means grouping neighbourhoods of 2 pixels into super-pixels, in the chosen direction and so on for 3 and 4. Although binning divides the resolution of the image by a factor equal to the size of the pixel neighbourhood, it allows for higher frame rates and the resulting super-pixels are more sensitive to light intensity than the individual pixels.

In order to maintain the aspect ratio of the image, both horizontal binning and vertical binning are set to the same value. However, using 3-by-3 binning results in a different aspect ratio, so that option was discarded and only 1-by-1, 2-by-2 and 4-by-4 binnings were considered. Due to the technical limitations of the `pylon-ros-camera` package, the camera can only achieve about half of the maximum frame rate for a given set of settings, which makes 1-by-1 binning too slow to be of use, with a frame rate of 2.11 Hz using 8-bit RGB encoding and 6.32 Hz using 8-bit Bayer encoding. Such a low frame result results in severe rolling shutter artefacts. Thus, this option is discarded as well.

As such, one of the purposes of this experiment is comparing the performance of the remaining two binning values, two-by-two and four-by-four, and determining which one is best suited. The other purpose of the experiment is to compare the performance of the detection of each of the markers (ArUco, AprilTag, STag and the custom marker) for different binning values and positions, with even lighting across them.

4.3.2 Setup and procedure

The chosen experiment site was a balcony with a flat floor made of square tiles arranged in a grid pattern with consistent tile and gap sizes. This facilitates the establishment of ground truth measurements for the purposes of this experiment.

The camera was set in a way such that its optical axis is aligned with the floor tiles' grid and parallel to the floor plane. All markers were printed on A4 paper, with the same size and secured to the box in the same position, so that they are consistently parallel to the image plane and their position relative to the reference positions is always the same. Figure

4.5 shows a photograph of the setup.

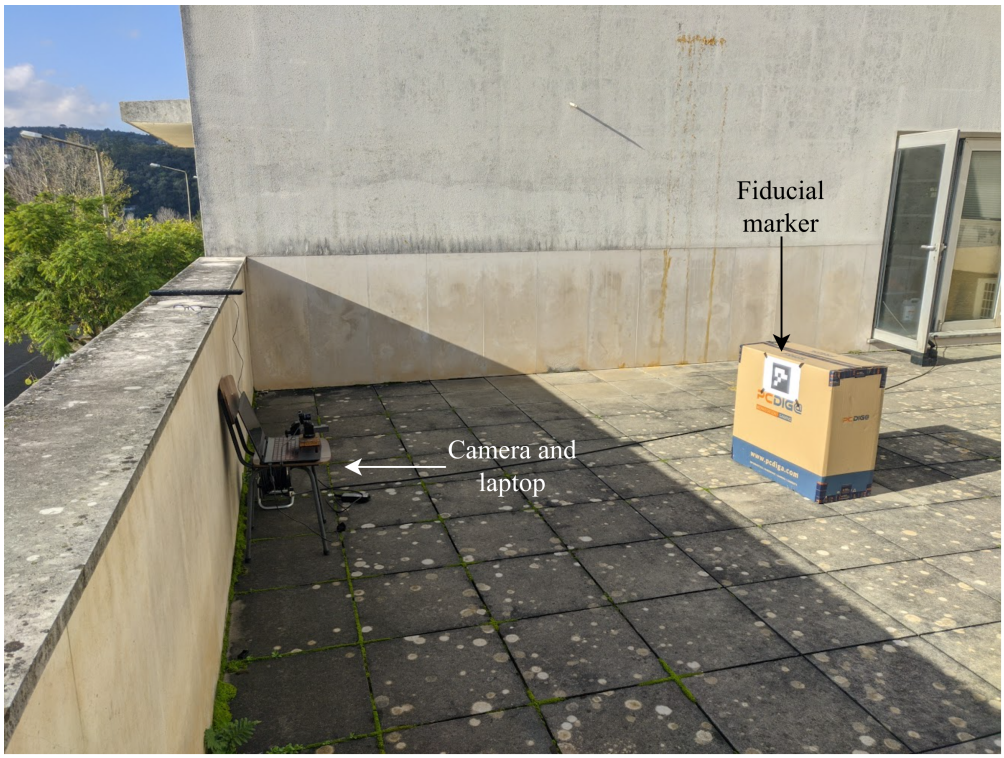


Figure 4.5: Experimental setup.

The reference positions are the intersections of the gaps between floor tiles, as shown in Figure 4.6. Their x coordinates range from 1 m to 13 m, in 1 m increments. Their y coordinates range depends on the x coordinates as described in Table 4.1, resulting in 37 different positions. A diagram of the positions is presented in Figure 4.6.

Table 4.1: Different sets within which y varies, depending on the value of x.

Set of x values [m]	Set of y values [m]
$\{1\}$	0
$\{2, 3\}$	$\{-0.5, 0, 0.5\}$
$\{4, 5, 6\}$	$\{-1, 0, 1\}$
$\{7, 8, 9, 10, 11, 12, 13\}$	$\{-2, 0, 2\}$

Each of the already available markers chosen for this experiment has various dictionaries, also known as tag families. Since only one at a time will be needed for landing and the marker needs a good detection distance, for each one, the dictionary with the smallest possible IDs and the smallest possible number of bits was chosen. By choosing the dictionary with the smallest IDs, one is ensuring they are choosing the dictionary which maximises the Hamming distance between the IDs, which allows for more error correction. By choosing the dictionary

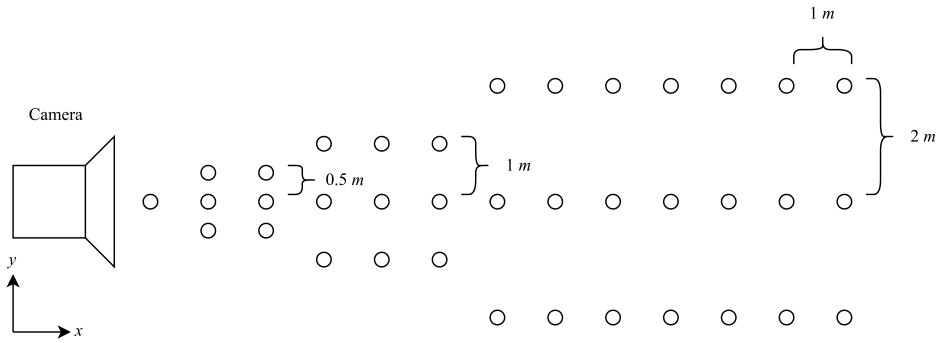


Figure 4.6: Diagram with dots representing the reference positions for the experiment and the frame of reference.

with the least amount of bits, one is ensuring they are choosing the dictionary with the biggest bit squares, thus maximising the detection distance. The latter is more important than the former.

Given these conditions, the chosen markers are:

- ArUco 4x4 50 IDs 0: marker with 16 bits and 50 different IDs;
- AprilTag 16h5 IDs 0: marker with 16 bits and a Hamming distance of 5;
- STag HD23 IDs 0: marker with a Hamming distance of 23.

The experimental procedure itself is the same for all eight trials – two for each of the four markers. Before starting each trial, the weather conditions were noted, as well as the light intensity. At the beginning, the marker was placed at the initial position, i.e., $(1, 0)$. Then, recording was started and after 5 s the marker is moved to the next position, always waiting at least the same amount of time at each position. By waiting at every position it is possible to gather more data at each one and better analyse the estimation performances.

4.3.3 Results and discussion

For analysis, 5 frames were selected for each position for each binning value for each marker. The frames were selected so that the marker does not move between them and that it is completely visible. Then, for each frame, the position is estimated, the method being different for each marker. For ArUco, the position was estimated using OpenCV's `aruco` Python sub-module. For AprilTag, the estimation was done by the `apriltag` Python sub-module, which returns the corners of the marker. These corners were used by OpenCV's `solvePnP` function, which estimates the position of the marker. For STag, the positions were

estimated live by the ROS package `stag_ros` and the position estimation output was record into a text file. Finally, for the custom marker, the method was the same as STag's position estimation.

For every marker, the statistical analysis of the results was done by a MATLAB script. The script estimates, for every frame, the error in each of the three axes and then, for each position, computes the mean error of that position's frames as well as its standard deviation. Tables with this data are shown in Appendix A. If at least one of the frames of a certain position did not yield a detection, the data is considered insufficient and the statistics are not computed. Additionally, the script computes and graphs the distance estimation for every marker and the ground truth distance, as shown in Figures 4.7 and 4.8, for 2-by-2 and 4-by-4 binning, respectively.

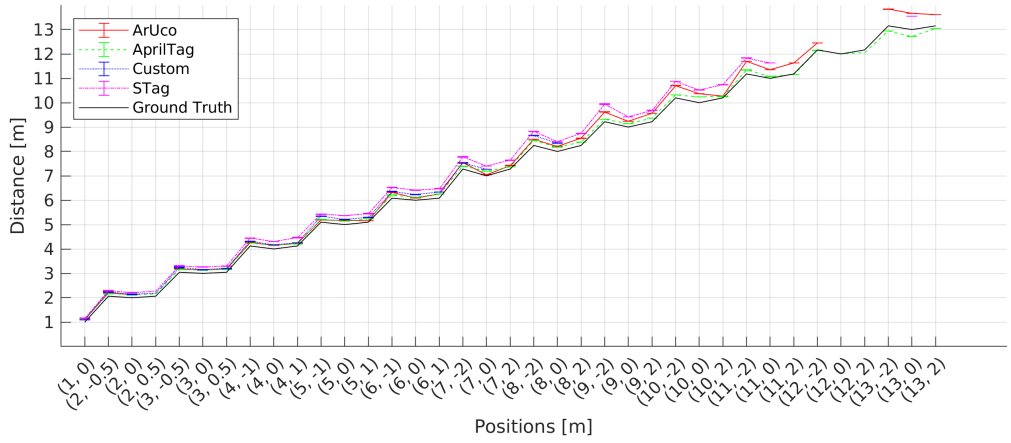


Figure 4.7: Mean of the estimated distance at each position for every marker and the ground truth using 2-by-2 binning.

Figure 4.8: Mean of the estimated distance at each position for every marker and the ground truth using 4-by-4 binning.

By looking at Figure 4.7 one can see only AprilTag achieved detections across all positions

and it did so with estimation errors overall smaller than the other markers. In Figure 4.8 it is noticeable that 4-by-4 binning had a great negative impact on the performance of all markers. All markers saw a reduction of about a half in their maximum detection distance, except for ArUco, which had sporadic detections in some of the greater distances, although only in one x value for each y value. This performance is too poor to be useful for landing the UAV and, as such, the marker chosen for this purpose was the AprilTag, due to the better detection distance and smaller error. Since 4-by-4 binning degrades the performance to an unacceptable degree, 2-by-2 binning is chosen instead.

4.4 Landing algorithm

4.4.1 Objectives

The objective of this experiment is to test and validate the landing algorithm developed for this dissertation and described in Section 2.2. Given the results of the experiment described in Section 4.3, AprilTag was chosen as the marker, instead of the CFM. The metrics evaluated in this experiment are the landing time and the error in position upon touchdown.

4.4.2 Setup and procedure

The simulation environment is a flat ground plane with the image of a runway and a box with the marker printed on its top face. The box's dimensions are 0.15341-by-0.15341-by-0.01 m. The marker's dimensions were chosen to mirror the size of the real AprilTag marker printed for the experiment in Section 4.3.It is shown in Figure 4.9.

The UAV starts in the origin of the world and the box is located at $(1.5, -1.5, 0)$ m. The landing controller is not immediately started because the wind simulation associated with it disturbs the behaviour of the FCU controller while the UAV gets in position to start a trial. Once both the autopilot and the SITL are running and ready, i.e., once the simulated RTK-GNSS is locked, the UAV is set to guided mode, where it can receive way points and velocity commands from MAVLink. It is then ordered to take off to 10 m above it's home position (i.e., the origin of the world) and only then is the landing controller node started. The behaviour of the drone is observed and record into a ROS bag file. The bag file records the following topics:

- Pose of the UAV estimated by the FCU: `/mavros/local_position/pose`

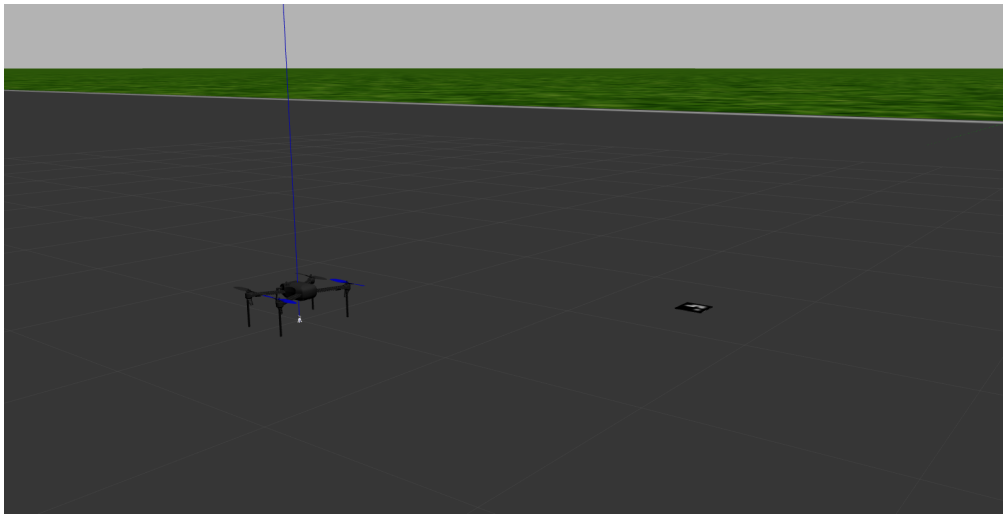


Figure 4.9: Screenshot depicting the simulation environment.

(`geometry_msgs/PoseStamped`);

- Velocity commands sent to the FCU: `/mavros/setpoint_velocity/cmd_vel` (`geometry_msgs/TwistStamped`);
- Centre of the target: `/target/centre` (`/geometry_msgs/PointStamped`);
- Images from the camera: `/iris/pylon_camera_node/image_rect` (`/sensor_msgs/Image`);
- Wind noise: `/wind` (`geometry_msgs/TwistStamped`).

4.4.3 Results and discussion

The data from the rosbag was written to text files that were loaded by MATLAB and organised into seven plots, discussed below. Since both the command and wind velocities only start publishing when the landing controller node is started, i.e., when the UAV is in position to start the trial, at 20 s, the lack of data before that time was replaced with zeros, as well as afterwards, when the UAV entered landing mode.

The trajectory of UAV can be seen in Figure 4.10. As can be observed, the UAV rapidly converges on top of the target, but the descent is slow, taking 55 s, to avoid crashing. The final position of the UAV is $(1.4999, 1.4318, -0.2137)$, which means the positional error is 0.0682 m.

Figures 4.11-4.13 show the evolution of the position of the UAV along the x, y and z axes, as well as the command velocity in the same axes.

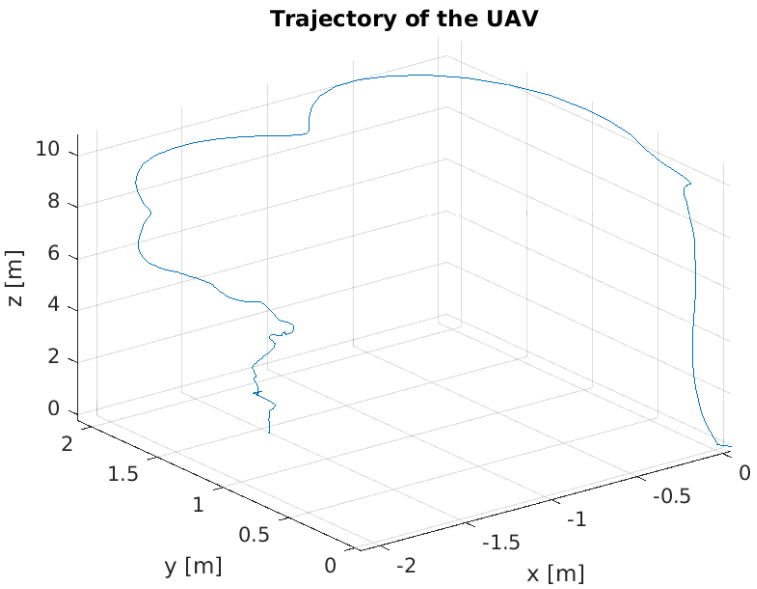


Figure 4.10: Trajectory of the UAV in the simulated trial.

Figure 4.11: Position and command velocity of the UAV in the x axis.

Figures 4.14-4.16 shown the velocity of the simulated wind in each of the three axes, as well as the command velocity of the UAV.

Analysing Figures 4.14-4.16 one can conclude that the controller successfully eliminated the effect of the wind, even when there was a sudden spike in velocity at 20 s. The controller shows some overshoot in both the x and y axis, but it is not severe and the UAV stabilises relatively fast, in about 20 s. This experiment was a success but the wind simulation is somewhat lacklustre and could be improved with the use of ordinary differential equations to better simulate its behaviour, as well as with increased magnitude and simulated wind gusts.

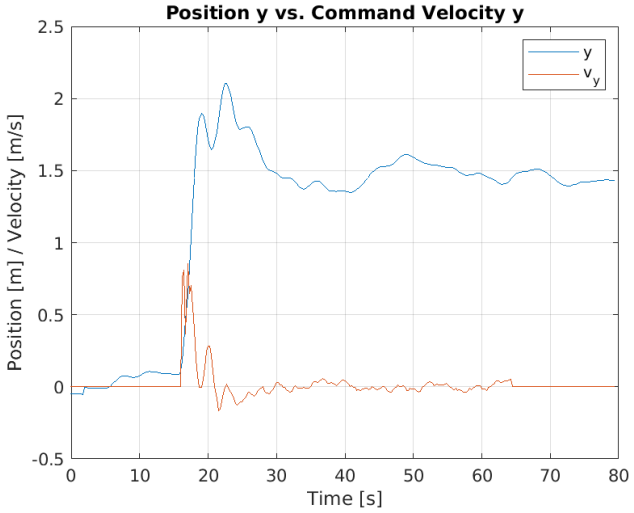


Figure 4.12: Position and command velocity of the UAV in the y axis.

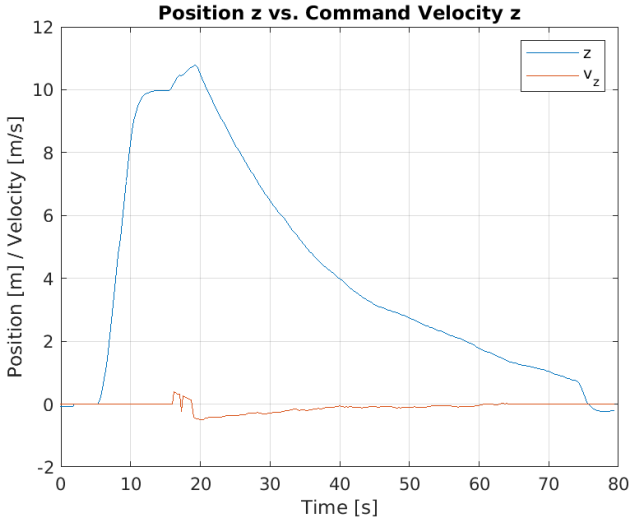


Figure 4.13: Position and command velocity of the UAV in the z axis.

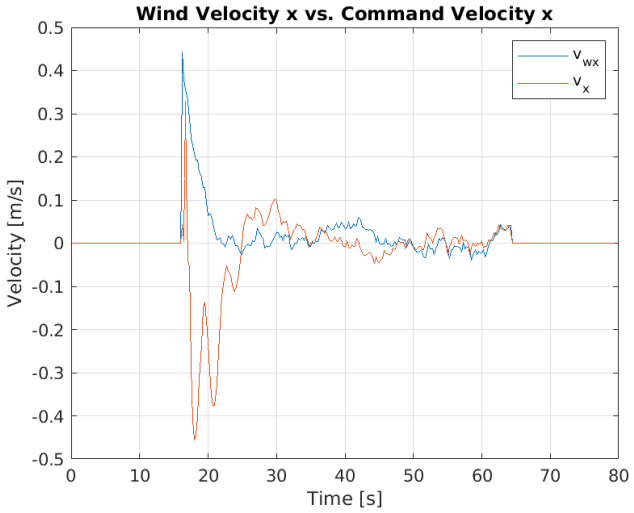


Figure 4.14: Wind and command velocities of the UAV in the x axis.

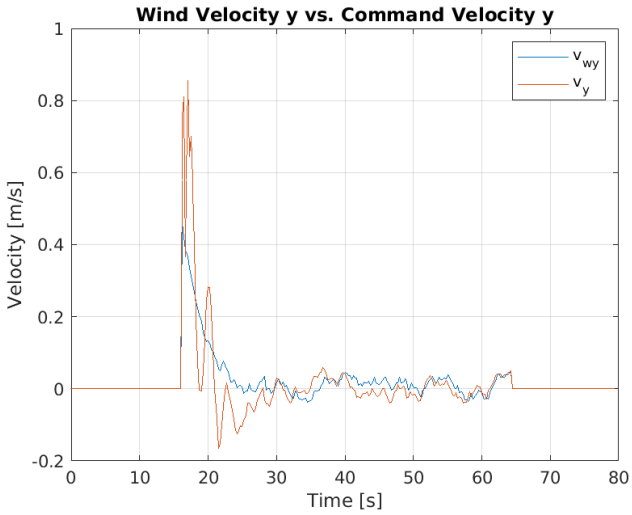


Figure 4.15: Wind and command velocities of the UAV in the y axis.

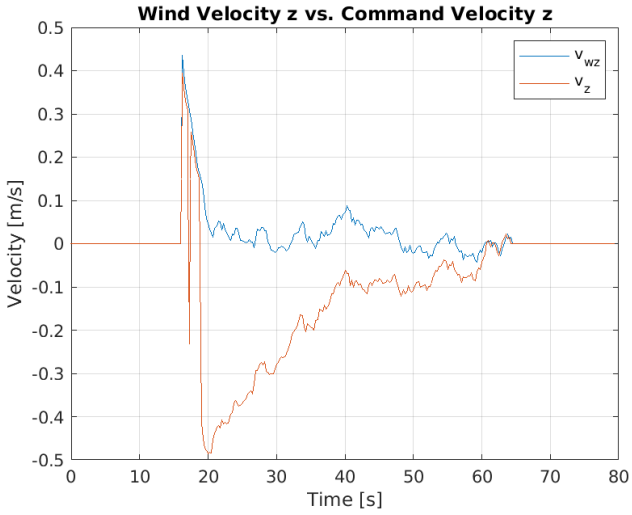


Figure 4.16: Wind and command velocities of the UAV in the z axis.

5 Conclusion and Future Work

One of the aims of this dissertation was to explore different fiducial markers and see which one was best suited for autonomous UAV landing. For this purpose, ArUco, AprilTag, STag and a CFM were compared in a controlled environment. The detection of the CFM was also custom built from scratch, using ellipse fitting and a series of conditions to match ellipses into rings and rings into valid markers. The conclusion of the comparison is that AprilTag is the best suited in every aspect, outperforming the others in detection range and pose estimation error, on top of being easily integrated into ROS and Python.

The other aim was to develop, test and validate a vision-based autonomous landing algorithm for the UAV that landed it on top of a fiducial marker. For this, a custom ROS package was built on top of the existing `iq_gnc` package and deployed to a real UAV. Unfortunately, the UAV was damaged during development and rendered unable to be used for further testing. For this reason, the UAV was instead simulated using the simulation software Gazebo. The trials were successful and the UAV landed in 55 s with a positional error of 0.0682 m.

5.1 Future work

Though the design of the CFM is satisfactory, some improvements can be made to its detection. Future work will improve upon the detection by exploring different pre-processing strategies to try to maximise the detection range and minimise the error of estimation of the centre, as well as explore new combinations of thresholds.

In spite of the good experimental results obtained in Section 4.4, they were obtained in a highly controlled simulated environment, which has close to ideal conditions. In the future, the algorithms described in this work will be transferred to real hardware and tested in the real world. Control issues will arise due to the simulation of the wind being relatively simple and real wind being far more chaotic. The need for a PID controller may arise to compensate

for oscillations due to wind gusts.

6 References

[1] Wikipedia contributors, "Bayer filter — Wikipedia, the free encyclopedia," 2022. [Online; accessed 14-February-2023].

[2] C. Cai, J. Liu, S. Wu, Y. Zhang, L. Jiang, Z. Zhang, and J. Yu, "Development of a cross-type magnetic coupler for unmanned aerial vehicle ipt charging systems," *IEEE Access*, vol. 8, pp. 67974–67989, 2020.

[3] S. Hayat, E. Yanmaz, and R. Muzaffar, "Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2624–2661, 2016.

[4] "At 12.27% CAGR, Drone Market Size to hit USD 40.9 Bn in 2027, says Brandessence Market Research." [Online; Accessed 16-12-2022].

[5] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48572–48634, 2019.

[6] S. Zhang, W. Liu, and N. Ansari, "On Tethered UAV-Assisted Heterogeneous Network," *IEEE Transactions on Vehicular Technology*, vol. 71, pp. 975–983, Jan. 2022. Conference Name: IEEE Transactions on Vehicular Technology.

[7] F. Muttin, "Umbilical deployment modeling for tethered UAV detecting oil pollution from ship," *Applied Ocean Research*, vol. 33, pp. 332–343, Oct. 2011.

[8] C. Van Nguyen, T. Van Quyen, A. M. Le, L. H. Truong, and M. T. Nguyen, "Advanced hybrid energy harvesting systems for unmanned aerial vehicles (uavs)," *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 1, pp. 34–39, 2020.

[9] B. Zhang, C. H. Liu, J. Tang, Z. Xu, J. Ma, and W. Wang, "Learning-based energy-efficient data collection by unmanned vehicles in smart cities," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1666–1676, 2018.

[10] "Dji dock - for roads less traveled." `https://www.dji.com/pt/dock?site=brandsite&from=nav`, 2023. [Online; accessed 12-February-2023].

[11] B. Herisse, F.-X. Russotto, T. Hamel, and R. Mahony, "Hovering flight and vertical landing control of a VTOL unmanned aerial vehicle using optical flow," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 801–806, 2008.

[12] M. Fiala, "ARTag, a fiducial marker system using digital techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 590–596 vol. 2, 2005.

[13] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Fiducial markers for pose estimation," *Journal of Intelligent & Robotic Systems*, vol. 101, Mar. 2021.

[14] L. B. Gatrell, W. A. Hoff, and C. W. Sklair, "Robust image features: concentric contrasting circles and their image extraction," in *Cooperative Intelligent Robotics in Space II* (W. E. Stoney, ed.), vol. 1612, pp. 235 – 244, International Society for Optics and Photonics, SPIE, 1992.

[15] L. Calvet, P. Gurdjos, C. Griwodz, and S. Gasparini, "Detection and accurate localization of circular fiducials under highly challenging conditions," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 562–570, 2016.

[16] J. Sattar, E. Bourque, P. Giguere, and G. Dudek, "Fourier tags: Smoothly degradable fiducial markers for use in human-robot interaction," in *Fourth Canadian Conference on Computer and Robot Vision (CRV '07)*, pp. 165–174, 2007.

[17] Y. Cho, J. Lee, and U. Neumann, "A multi-ring color fiducial system and an intensity-invariant detection method for scalable fiducial-tracking augmented reality," 02 1970.

[18] F. Bergamasco, A. Albarelli, E. Rodolà, and A. Torsello, "RUNE-Tag: A high accuracy fiducial marker with strong occlusion resilience," in *CVPR 2011*, pp. 113–120, 2011.

[19] F. Schweiger, B. Zeisl, P. Georgel, G. Schroth, E. Steinbach, and N. Navab, "Maximum detector response markers for sift and surf," pp. 145–154, 01 2009.

[20] G. Yu, Y. Hu, and J. Dai, "Topotag: A robust and scalable topological fiducial marker system," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 9, pp. 3769–3780, 2021.

[21] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, IEEE, May 2011.

[22] B. Benligiray, C. Topal, and C. Akinlar, "Stag: A stable fiducial marker system," *Image and Vision Computing*, vol. 89, pp. 158–169, 2019.

[23] S. Saripalli, J. Montgomery, and G. Sukhatme, "Vision-based autonomous landing of an unmanned aerial vehicle," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 3, pp. 2799–2804 vol.3, May 2002.

[24] S. Saripalli, J. Montgomery, and G. Sukhatme, "Visually guided landing of an unmanned aerial vehicle," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 371–380, June 2003. Conference Name: IEEE Transactions on Robotics and Automation.

[25] T. K. Venugopalan, T. Taher, and G. Barbastathis, "Autonomous landing of an Unmanned Aerial Vehicle on an autonomous marine vehicle," in *2012 Oceans*, pp. 1–9, Oct. 2012. ISSN: 0197-7385.

[26] T. Baca, P. Stepan, and M. Saska, "Autonomous landing on a moving car with unmanned aerial vehicle," in *2017 European Conference on Mobile Robots (ECMR)*, pp. 1–6, Sept. 2017.

[27] T. Baca, P. Stepan, V. Spurny, D. Hert, R. Penicka, M. Saska, J. Thomas, G. Loianno, and V. Kumar, "Autonomous landing on a moving vehicle with an unmanned aerial vehicle," *Journal of Field Robotics*, vol. 36, no. 5, pp. 874–891, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21858.

[28] V. Sudevan, A. Shukla, and H. Karki, "Vision based autonomous landing of an Unmanned Aerial Vehicle on a stationary target," in *2017 17th International Conference on Control, Automation and Systems (ICCAS)*, pp. 362–367, Oct. 2017.

[29] S. Lee, T. Shim, S. Kim, J. Park, K. Hong, and H. Bang, "Vision-Based Autonomous Landing of a Multi-Copter Unmanned Aerial Vehicle using Reinforcement Learning," in

2018 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 108–114, June 2018. ISSN: 2575-7296.

[30] P. H. Nguyen, M. Arsalan, J. H. Koo, R. A. Naqvi, N. Q. Truong, and K. R. Park, "LightDenseYOLO: A Fast and Accurate Marker Tracker for Autonomous UAV Landing by Visible Light Camera Sensor on Drone," *Sensors*, vol. 18, p. 1703, June 2018. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[31] R. Liu, J. Yi, Y. Zhang, B. Zhou, W. Zheng, H. Wu, S. Cao, and J. Mu, "Vision-guided autonomous landing of multirotor UAV on fixed landing marker," in *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pp. 455–458, June 2020.

[32] Y. Zhang, J. Li, J. Chen, Z. Liu, and Z. Chen, "A GCN-Based Decision-Making Network for Autonomous UAV Landing," in *2020 5th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 652–657, Dec. 2020.

[33] A. Keipour, G. A. S. Pereira, R. Bonatti, R. Garg, P. Rastogi, G. Dubey, and S. Scherer, "Visual Servoing Approach for Autonomous UAV Landing on a Moving Vehicle," Apr. 2021. arXiv:2104.01272 [cs, eess].

[34] L. Persson and B. Wahlberg, "Variable Prediction Horizon Control for Cooperative Landing on Moving Target," in *2021 IEEE Aerospace Conference (50100)*, pp. 1–10, Mar. 2021. ISSN: 1095-323X.

[35] B. Lee, V. Saj, M. Benedict, and D. Kalathil, "Intelligent Vision-based Autonomous Ship Landing of VTOL UAVs," Sept. 2022. arXiv:2202.13005 [cs].

[36] V. Saj, B. Lee, D. Kalathil, and M. Benedict, "Robust Reinforcement Learning Algorithm for Vision-based Ship Landing of UAVs," Sept. 2022. arXiv:2209.08381 [cs].

Appendix A

Comparison of fiducial marker detection performance

A.1 AprilTag

Table A.1: Mean and standard deviation of the estimation error at each position using the AprilTag 16h5 ID 0 marker and 2-by-2 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.024	0.015	-0.573	0.000308	0.000026	0.000130
2	-1	0.022	-0.016	0.039	-0.546	0.000336	0.000076	0.000051
2	0	0.022	-0.036	0.032	-0.549	0.000757	0.000046	0.000021
2	1	0.022	-0.025	0.029	-0.551	0.000464	0.000092	0.000033
3	-1	0.022	-0.032	0.056	-0.521	0.000863	0.000148	0.000035
3	0	0.022	-0.052	0.053	-0.524	0.003590	0.000177	0.000183
3	1	0.022	-0.042	0.045	-0.527	0.000639	0.000092	0.000021
4	-1	0.022	-0.013	0.086	-0.497	0.000955	0.000253	0.000011
4	0	0.022	-0.071	0.062	-0.501	0.006669	0.000083	0.000222
4	1	0.022	-0.017	0.062	-0.508	0.001105	0.000268	0.000037
5	-1	0.022	-0.017	0.101	-0.471	0.001949	0.000428	0.000062
5	0	0.022	-0.052	0.086	-0.477	0.001700	0.000117	0.000188
5	1	0.022	-0.067	0.058	-0.480	0.003352	0.000628	0.000033
6	-1	0.022	-0.029	0.115	-0.448	0.000355	0.000063	0.000006
6	0	0.022	-0.044	0.098	-0.457	0.000713	0.000014	0.000009
6	1	0.022	-0.101	0.081	-0.457	0.008194	0.001195	0.000037
7	-2	0.022	0.005	0.136	-0.422	0.007639	0.002272	0.000064
7	0	0.022	-0.108	0.113	-0.428	0.004558	0.000079	0.000043
7	2	0.022	-0.033	0.079	-0.440	0.009845	0.002328	0.000075
8	-2	0.022	-0.079	0.178	-0.395	0.001943	0.000464	0.000120
8	0	0.022	-0.085	0.134	-0.403	0.007182	0.000173	0.000111

Table A.1 continued from previous page

8	2	0.022	-0.081	0.093	-0.411	0.002043	0.000399	0.000051
9	-2	0.022	0.016	0.165	-0.371	0.062603	0.014987	0.002071
9	0	0.022	-0.054	0.142	-0.381	0.019332	0.000337	0.000498
9	2	0.022	-0.093	0.108	-0.384	0.009534	0.002055	0.000340
10	-2	0.022	0.002	0.187	-0.345	0.015944	0.003320	0.001380
10	0	0.022	-0.149	0.161	-0.358	0.002322	0.000089	0.000048
10	2	0.022	0.008	0.142	-0.366	0.020195	0.003708	0.000464
11	-2	0.022	-0.042	0.213	-0.323	0.035134	0.006803	0.000675
11	0	0.022	0.004	0.174	-0.335	0.029150	0.000558	0.000660
11	2	0.022	0.081	0.176	-0.346	0.019869	0.003298	0.000301
12	-2	0.022	0.135	0.194	-0.306	0.016788	0.003240	0.000349
12	0	0.022	0.084	0.189	-0.318	0.011201	0.000185	0.000246
12	2	0.022	0.156	0.201	-0.326	0.015410	0.002388	0.000317
13	-2	0.022	0.323	0.170	-0.288	0.019244	0.003305	0.000396
13	0	0.022	0.379	0.196	-0.298	0.025783	0.000566	0.000583
13	2	0.022	0.166	0.217	-0.305	0.010601	0.001403	0.000216
Mean of all positions			-0.030	0.082	-0.476	0.004358	0.000458	0.000114

Table A.2: Mean and standard deviation of the estimation error at each position using the AprilTag 16h5 ID 0 marker and 4-by-4 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.012	0.015	-0.573	0.000627	0.000845	0.000303
2	-1	0.022	Insufficient data			Insufficient data		
2	0	0.022	Insufficient data			Insufficient data		
2	1	0.022	-0.040	0.026	-0.551	0.000679	0.000321	0.000094
3	-1	0.022	-0.051	0.058	-0.520	0.000506	0.000315	0.000008
3	0	0.022	Insufficient data			Insufficient data		
3	1	0.022	-0.058	0.034	-0.526	0.003302	0.000488	0.000066
4	-1	0.022	Insufficient data			Insufficient data		
4	0	0.022	0.066	0.062	-0.499	0.004273	0.000042	0.000415
4	1	0.022	-0.133	0.017	-0.503	0.001478	0.000502	0.000034
5	-1	0.022	-0.046	0.104	-0.462	0.001001	0.000222	0.000030
5	0	0.022	-0.078	0.080	-0.472	0.000278	0.000002	0.000009
5	1	0.022	Insufficient data			Insufficient data		
6	-1	0.022	0.027	0.097	-0.443	0.000788	0.000147	0.000026
6	0	0.022	-0.081	0.093	-0.450	0.010573	0.000098	0.000137
6	1	0.022	0.027	0.088	-0.455	0.002704	0.000448	0.000170
7	-2	0.022	Insufficient data			Insufficient data		
7	0	0.022	-0.026	0.111	-0.421	0.046885	0.000200	0.000831
7	2	0.022	Insufficient data			Insufficient data		
8	-2	0.022	Insufficient data			Insufficient data		
8	0	0.022	Insufficient data			Insufficient data		
8	2	0.022	Insufficient data			Insufficient data		
9	-2	0.022	Insufficient data			Insufficient data		
9	0	0.022	Insufficient data			Insufficient data		
9	2	0.022	Insufficient data			Insufficient data		
10	-2	0.022	Insufficient data			Insufficient data		
10	0	0.022	Insufficient data			Insufficient data		
10	2	0.022	Insufficient data			Insufficient data		
11	-2	0.022	Insufficient data			Insufficient data		
11	0	0.022	Insufficient data			Insufficient data		
11	2	0.022	Insufficient data			Insufficient data		
12	-2	0.022	Insufficient data			Insufficient data		
12	0	0.022	Insufficient data			Insufficient data		
12	2	0.022	Insufficient data			Insufficient data		
13	-2	0.022	Insufficient data			Insufficient data		
13	0	0.022	Insufficient data			Insufficient data		
13	2	0.022	Insufficient data			Insufficient data		
Mean of all positions			-0.034	0.065	-0.489	0.006091	0.000302	0.000177

A.2 ArUco

Table A.3: Mean and standard deviation of the estimation error at each position using the ArUco 4X4 50 ID 0 marker and 2-by-2 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.023	-0.004	-0.569	0.002232	0.000326	0.000644
2	-1	0.022	-0.064	0.005	-0.544	0.012908	0.003138	0.000434
2	0	0.022	-0.044	-0.012	-0.545	0.005415	0.000154	0.000180
2	1	0.022	Insufficient data			Insufficient data		
3	-1	0.022	-0.059	-0.006	-0.515	0.000520	0.000342	0.000034
3	0	0.022	-0.060	-0.022	-0.516	0.011004	0.000990	0.000540
3	1	0.022	-0.070	-0.029	-0.522	0.008188	0.001477	0.000739
4	-1	0.022	-0.069	-0.011	-0.486	0.001330	0.000081	0.000046
4	0	0.022	-0.064	-0.029	-0.491	0.000000	0.000000	0.000000
4	1	0.022	-0.030	-0.054	-0.498	0.008962	0.002929	0.000754
5	-1	0.022	-0.010	-0.025	-0.461	0.021383	0.004077	0.000767
5	0	0.022	-0.092	-0.032	-0.464	0.000000	0.000000	0.000000
5	1	0.022	0.032	-0.047	-0.473	0.030600	0.006401	0.001037
6	-1	0.022	-0.166	-0.012	-0.439	0.034044	0.005451	0.000659
6	0	0.022	-0.014	-0.045	-0.445	0.018815	0.001005	0.000937
6	1	0.022	-0.079	-0.080	-0.448	0.064299	0.010428	0.001167
7	-2	0.022	-0.188	0.015	-0.406	0.043338	0.012705	0.001312
7	0	0.022	0.037	-0.049	-0.417	0.048434	0.001225	0.002877
7	2	0.022	-0.038	-0.103	-0.425	0.003910	0.000723	0.000089
8	-2	0.022	-0.177	-0.008	-0.382	0.061746	0.015179	0.002065
8	0	0.022	Insufficient data			Insufficient data		
8	2	0.022	-0.188	-0.148	-0.399	0.088235	0.023001	0.001970
9	-2	0.022	-0.318	0.023	-0.346	0.000000	0.000000	0.000000
9	0	0.022	-0.148	-0.079	-0.365	0.000000	0.000000	0.000000
9	2	0.022	Insufficient data			Insufficient data		
10	-2	0.022	-0.423	0.028	-0.322	0.000000	0.000000	0.000000
10	0	0.022	-0.280	-0.087	-0.338	0.106562	0.000949	0.003169
10	2	0.022	Insufficient data			Insufficient data		
11	-2	0.022	Insufficient data			Insufficient data		
11	0	0.022	Insufficient data			Insufficient data		
11	2	0.022	Insufficient data			Insufficient data		
12	-2	0.022	-0.210	-0.036	-0.280	0.094698	0.015961	0.002122
12	0	0.022	Insufficient data			Insufficient data		
12	2	0.022	Insufficient data			Insufficient data		
13	-2	0.022	-0.608	-0.002	-0.255	0.005435	0.001048	0.001473
13	0	0.022	Insufficient data			Insufficient data		
13	2	0.022	Insufficient data			Insufficient data		
Mean			-0.152	-0.025	-0.437	0.022340	0.003000	0.000795

Table A.4: Mean and standard deviation of the estimation error at each position using the ArUco 4X4 50 ID 0 marker and 4-by-4 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.018	0.013	-0.573	0.002767	0.000824	0.000189
2	-1	0.022	-0.016	0.033	-0.545	0.000000	0.000000	0.000000
2	0	0.022	-0.036	0.029	-0.547	0.000000	0.000000	0.000000
2	1	0.022	-0.063	0.016	-0.550	0.000000	0.000000	0.000000
3	-1	0.022	-0.022	0.053	-0.519	0.005986	0.001722	0.001528
3	0	0.022	-0.046	0.038	-0.521	0.021414	0.000160	0.000263
3	1	0.022	0.002	0.043	-0.526	0.025622	0.003871	0.000338
4	-1	0.022	-0.088	0.090	-0.492	0.000372	0.000563	0.000049
4	0	0.022	Insufficient data			Insufficient data		
4	1	0.022	-0.204	0.008	-0.503	0.000000	0.000000	0.000000
5	-1	0.022	-0.042	0.101	-0.467	0.000000	0.000000	0.000000
5	0	0.022	-0.084	0.077	-0.477	0.104329	0.001412	0.003169
5	1	0.022	-0.064	0.056	-0.478	0.000000	0.000000	0.000000
6	-1	0.022	-0.324	0.156	-0.437	0.000000	0.000000	0.000000
6	0	0.022	-0.162	0.093	-0.453	0.016092	0.000221	0.000570
6	1	0.022	-0.180	0.053	-0.453	0.000000	0.000000	0.000000
7	-2	0.022	-0.090	0.155	-0.412	0.000000	0.000000	0.000000
7	0	0.022	Insufficient data			Insufficient data		
7	2	0.022	Insufficient data			Insufficient data		
8	-2	0.022	Insufficient data			Insufficient data		
8	0	0.022	Insufficient data			Insufficient data		
8	2	0.022	Insufficient data			Insufficient data		
9	-2	0.022	Insufficient data			Insufficient data		
9	0	0.022	Insufficient data			Insufficient data		
9	2	0.022	-0.366	0.044	-0.378	0.204342	0.042461	0.005518
10	-2	0.022	Insufficient data			Insufficient data		
10	0	0.022	Insufficient data			Insufficient data		
10	2	0.022	Insufficient data			Insufficient data		
11	-2	0.022	Insufficient data			Insufficient data		
11	0	0.022	-0.869	0.178	-0.308	0.280825	0.004151	0.007593
11	2	0.022	Insufficient data			Insufficient data		
12	-2	0.022	Insufficient data			Insufficient data		
12	0	0.022	-0.958	0.191	-0.277	0.054315	0.002293	0.001191
12	2	0.022	Insufficient data			Insufficient data		
13	-2	0.022	Insufficient data			Insufficient data		
13	0	0.022	Insufficient data			Insufficient data		
13	2	0.022	Insufficient data			Insufficient data		
Mean			-0.191	0.075	-0.469	0.037688	0.003036	0.001074

A.3 STag

Table A.5: Mean and standard deviation of the estimation error at each position using the STag HD23 ID 0 marker and 2-by-2 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.072	0.020	-0.575	0.001523	0.000030	0.000077
2	-1	0.022	-0.128	0.067	-0.546	0.001018	0.000218	0.000014
2	0	0.022	-0.127	0.024	-0.549	0.004453	0.000079	0.000118
2	1	0.022	-0.128	0.002	-0.552	0.002395	0.000520	0.000037
3	-1	0.022	-0.168	0.080	-0.521	0.010598	0.001883	0.000230
3	0	0.022	-0.173	0.053	-0.523	0.001415	0.000016	0.000200
3	1	0.022	-0.182	0.012	-0.527	0.007025	0.001060	0.000322
4	-1	0.022	-0.214	0.135	-0.496	0.000626	0.000161	0.000011
4	0	0.022	-0.220	0.069	-0.500	0.014974	0.000230	0.000264
4	1	0.022	-0.262	-0.011	-0.506	0.011595	0.002675	0.000171
5	-1	0.022	-0.225	0.147	-0.469	0.009138	0.001859	0.000131
5	0	0.022	-0.287	0.090	-0.474	0.009023	0.000186	0.000124
5	1	0.022	-0.279	0.008	-0.480	0.010504	0.001877	0.000210
6	-1	0.022	-0.334	0.167	-0.444	0.001307	0.000310	0.000116
6	0	0.022	-0.328	0.095	-0.453	0.029307	0.000400	0.000545
6	1	0.022	-0.312	0.030	-0.457	0.075361	0.011402	0.001266
7	-2	0.022	-0.359	0.253	-0.417	0.009292	0.002860	0.000171
7	0	0.022	-0.314	0.115	-0.425	0.078485	0.001142	0.001406
7	2	0.022	-0.296	0.005	-0.440	0.026582	0.007245	0.000355
8	-2	0.022	-0.437	0.276	-0.391	0.098930	0.026315	0.002029
8	0	0.022	-0.319	0.125	-0.401	0.009377	0.000206	0.000229
8	2	0.022	-0.426	-0.002	-0.409	0.151737	0.035487	0.002832
9	-2	0.022	-0.587	0.318	-0.362	0.021752	0.005095	0.000800
9	0	0.022	-0.330	0.139	-0.375	0.126023	0.001959	0.002575
9	2	0.022	-0.396	0.021	-0.383	0.038630	0.008057	0.000719
10	-2	0.022	0.000	0.000	0.000	0.000000	0.000000	0.000000
10	0	0.022	-0.438	0.160	-0.353	0.021881	0.000233	0.000332
10	2	0.022	-0.477	0.036	-0.362	0.134943	0.024987	0.003134
11	-2	0.022	-0.518	0.300	-0.318	0.019111	0.003797	0.000323
11	0	0.022	-0.545	0.176	-0.324	0.005324	0.000092	0.000078
11	2	0.022	Insufficient data			Insufficient data		
12	-2	0.022	Insufficient data			Insufficient data		
12	0	0.022	Insufficient data			Insufficient data		
12	2	0.022	-0.484	0.085	-0.317	0.018214	0.002742	0.000280
13	-2	0.022	Insufficient data			Insufficient data		
13	0	0.022	Insufficient data			Insufficient data		
13	2	0.022	Insufficient data			Insufficient data		
Mean			-0.302	0.097	-0.431	0.030663	0.004617	0.000616

Table A.6: Mean and standard deviation of the estimation error at each position using the STag HD23 ID 0 marker and 4-by-4 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.074	0.021	-0.573	0.001488	0.000087	0.000080
2	-1	0.022	-0.131	0.057	-0.543	0.005987	0.001496	0.000093
2	0	0.022	-0.118	0.030	-0.547	0.003302	0.000126	0.000109
2	1	0.022	-0.150	-0.005	-0.549	0.019379	0.004309	0.000126
3	-1	0.022	-0.182	0.083	-0.516	0.007687	0.001496	0.000133
3	0	0.022	-0.144	0.048	-0.520	0.026473	0.000441	0.000367
3	1	0.022	-0.203	0.021	-0.523	0.004877	0.000652	0.000160
4	-1	0.022	-0.159	0.110	-0.491	0.074531	0.019693	0.001434
4	0	0.022	-0.202	0.071	-0.496	0.033871	0.000484	0.000532
4	1	0.022	-0.348	-0.020	-0.500	0.004245	0.000962	0.000071
5	-1	0.022	-0.337	0.161	-0.462	0.002586	0.000116	0.000048
5	0	0.022	-0.301	0.089	-0.468	0.056200	0.000909	0.000983
5	1	0.022	-0.288	0.007	-0.474	0.041636	0.008778	0.000831
6	-1	0.022	-0.069	0.109	-0.441	0.010515	0.001952	0.000301
6	0	0.022	Insufficient data			Insufficient data		
6	1	0.022	Insufficient data			Insufficient data		
7	-2	0.022	Insufficient data			Insufficient data		
7	0	0.022	Insufficient data			Insufficient data		
7	2	0.022	Insufficient data			Insufficient data		
8	-2	0.022	Insufficient data			Insufficient data		
8	0	0.022	Insufficient data			Insufficient data		
8	2	0.022	Insufficient data			Insufficient data		
9	-2	0.022	Insufficient data			Insufficient data		
9	0	0.022	Insufficient data			Insufficient data		
9	2	0.022	Insufficient data			Insufficient data		
10	-2	0.022	Insufficient data			Insufficient data		
10	0	0.022	Insufficient data			Insufficient data		
10	2	0.022	Insufficient data			Insufficient data		
11	-2	0.022	Insufficient data			Insufficient data		
11	0	0.022	Insufficient data			Insufficient data		
11	2	0.022	Insufficient data			Insufficient data		
12	-2	0.022	Insufficient data			Insufficient data		
12	0	0.022	Insufficient data			Insufficient data		
12	2	0.022	Insufficient data			Insufficient data		
13	-2	0.022	Insufficient data			Insufficient data		
13	0	0.022	Insufficient data			Insufficient data		
13	2	0.022	Insufficient data			Insufficient data		
Mean			-0.193	0.056	-0.507	0.020913	0.002964	0.000376

A.4 Custom marker

Table A.7: Mean and standard deviation of the estimation error at each position using the custom marker and 2-by-2 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.034	-0.001	0.020	0.001661	0.000005	0.000006
2	-1	0.022	-0.091	-0.001	0.049	0.000285	0.000064	0.000034
2	0	0.022	-0.059	-0.012	0.047	0.000448	0.000033	0.000018
2	1	0.022	-0.046	-0.023	0.041	0.010765	0.002508	0.000310
3	-1	0.022	-0.116	-0.001	0.078	0.002880	0.000517	0.000100
3	0	0.022	-0.071	-0.021	0.075	0.000580	0.000021	0.000091
3	1	0.022	-0.073	-0.036	0.071	0.001245	0.000176	0.000047
4	-1	0.022	-0.120	-0.011	0.107	0.002813	0.000751	0.000179
4	0	0.022	-0.090	-0.029	0.100	0.001	0.000	0.000
4	1	0.022	-0.044	-0.045	0.093	0.004	0.001	0.000
5	-1	0.022	-0.171	-0.015	0.135	0.005	0.001	0.000
5	0	0.022	-0.126	-0.039	0.130	0.004	0.000	0.000
5	1	0.022	-0.109	-0.063	0.121	0.005	0.001	0.000
6	-1	0.022	-0.201	-0.025	0.157	0.012	0.002	0.000
6	0	0.022	-0.148	-0.052	0.152	0.009	0.000	0.000
6	1	0.022	-0.161	-0.077	0.143	0.010	0.002	0.001
7	-2	0.022	-0.181	-0.019	0.189	0.003	0.001	0.000
7	0	0.022	-0.182	-0.057	0.182	0.017	0.000	0.001
7	2	0.022	Insufficient data			Insufficient data		
8	-2	0.022	-0.328	0.019	0.220	0.013	0.003	0.000
8	0	0.022	-0.261	-0.071	0.205	0.002	0.000	0.000
8	2	0.022	Insufficient data			Insufficient data		
9	-2	0.022	Insufficient data			Insufficient data		
9	0	0.022	Insufficient data			Insufficient data		
9	2	0.022	Insufficient data			Insufficient data		
10	-2	0.022	Insufficient data			Insufficient data		
10	0	0.022	Insufficient data			Insufficient data		
10	2	0.022	Insufficient data			Insufficient data		
11	-2	0.022	Insufficient data			Insufficient data		
11	0	0.022	Insufficient data			Insufficient data		
11	2	0.022	Insufficient data			Insufficient data		
12	-2	0.022	Insufficient data			Insufficient data		
12	0	0.022	Insufficient data			Insufficient data		
12	2	0.022	Insufficient data			Insufficient data		
13	-2	0.022	Insufficient data			Insufficient data		
13	0	0.022	Insufficient data			Insufficient data		
13	2	0.022	Insufficient data			Insufficient data		
Mean			-0.131	-0.029	0.116	0.005263	0.000739	0.000183

Table A.8: Mean and standard deviation of the estimation error at each position using the custom marker and 4-by-4 binning.

Position [m]			Mean estimation error [m]			Standard deviation of the estimation error [m]		
x	y	z	x	y	z	x	y	z
1	0	0.022	-0.019	0.014	0.016	0.000417	0.000011	0.000023
2	-1	0.022	-0.021	0.028	0.044	0.000642	0.000168	0.000011
2	0	0.022	-0.045	0.027	0.041	0.000862	0.000062	0.000009
2	1	0.022	-0.051	0.028	0.038	0.000853	0.000252	0.000096
3	-1	0.022	-0.049	0.047	0.068	0.005785	0.000684	0.000224
3	0	0.022	-0.173	-0.001	0.091	0.015361	0.000076	0.000329
3	1	0.022	-0.073	0.026	0.062	0.002214	0.000351	0.000023
4	-1	0.022	Insufficient data			Insufficient data		
4	0	0.022	Insufficient data			Insufficient data		
4	1	0.022	Insufficient data			Insufficient data		
5	-1	0.022	Insufficient data			Insufficient data		
5	0	0.022	Insufficient data			Insufficient data		
5	1	0.022	Insufficient data			Insufficient data		
6	-1	0.022	Insufficient data			Insufficient data		
6	0	0.022	Insufficient data			Insufficient data		
6	1	0.022	Insufficient data			Insufficient data		
7	-2	0.022	Insufficient data			Insufficient data		
7	0	0.022	Insufficient data			Insufficient data		
7	2	0.022	Insufficient data			Insufficient data		
8	-2	0.022	Insufficient data			Insufficient data		
8	0	0.022	Insufficient data			Insufficient data		
8	2	0.022	Insufficient data			Insufficient data		
9	-2	0.022	Insufficient data			Insufficient data		
9	0	0.022	Insufficient data			Insufficient data		
9	2	0.022	Insufficient data			Insufficient data		
10	-2	0.022	Insufficient data			Insufficient data		
10	0	0.022	Insufficient data			Insufficient data		
10	2	0.022	Insufficient data			Insufficient data		
11	-2	0.022	Insufficient data			Insufficient data		
11	0	0.022	Insufficient data			Insufficient data		
11	2	0.022	Insufficient data			Insufficient data		
12	-2	0.022	Insufficient data			Insufficient data		
12	0	0.022	Insufficient data			Insufficient data		
12	2	0.022	Insufficient data			Insufficient data		
13	-2	0.022	Insufficient data			Insufficient data		
13	0	0.022	Insufficient data			Insufficient data		
13	2	0.022	Insufficient data			Insufficient data		
Mean			-0.061	0.024	0.051	0.003733	0.000229	0.000102