SCHOOL OF MECHANICAL,
INDUSTRIAL & AERONAUTICAL
ENGINEERING

Research Report

Quadrotor Visual Servoing For Moving Target Tracking

MECN4006 - Research Project

Edward Rycroft

Student Number: 1478715

Supervisor: Ms. C. Kuchwa-Dube

A project report submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Bachelor of Science in Engineering (Mechanical).

Johannesburg, October 2019

ABSTRACT

An investigation into position-based visual servoing through end-point open-loop control was conducted for estimation of the three-dimensional position and yaw orientation of a moving target using a single camera, where a Bitcraze Crazyflie 2.1 quadrotor then tracked this position and yaw orientation autonomously. Moreover, inexpensive and low-end hardware with marginal computational effort was used in the form of a Raspberry Pi Zero W 1.1 and Raspberry Pi Camera Module 2.1 for image processing with OpenCV to utilise computer vision techniques. Comparing grayscale and colour processing for initial target detection, it was evident that grayscale processing allowed for an increased frame rate compared to colour processing by an average percentage difference of 23.2% while also eliminating more background noise for a better interpretation of the target. During the implementation, it was found that the most satisfactory resolution was 160px by 120px for lightweight processing, where the capability to generate setpoints for the quadrotor was at a frequency of 30.1Hz. The pinhole camera model was also validated and successfully implemented at a resolution of 160px by 120px which resulted in a focal length of 113.5px found through experimental calibration. This allowed for the pinhole camera model to be used to develop a control script to reconstruct the state of the target relative to the camera with the centroid of the target providing latitude and longitude coordinates, area of the target providing the altitude of the camera, and rotation providing the yaw orientation when using a disproportioned marker. With this information, the quadrotor was able to successfully mirror the translational and rotational motion of the target while maintaining the altitude of the camera. Over eight independent tests, the real-time effectiveness was captured by a Qualisys motion capture system and exhibited an average lag of 0.732s, average position deviations of 83.4mm, minimum yaw orientation deviation of -9.54^o, and maximum yaw orientation deviation of 9.87^o. However, if the average lag is compensated in each test, the average position deviation reduces to only 54.0mm, while the minimum and maximum yaw orientation deviations reduce to only -4.76^o and 4.46^o respectively. An altitude could also be maintained effectively, where the quadrotor only drifted by an average of 3.75% downwards or 3.98% upwards before returning to the correct altitude.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

The general symbols used, with a description and relevant units, are as follows:

a	first arbitrary variable	-
A_{uv}	virtual image area	px^2
A_{xy}	actual area	m^2
b	second arbitrary variable	-
$b(t)$	feedback signal	-
d_+	plus-mode moment arm	m
$d_\times$	cross-mode moment arm	m
D_{xy}	latitude and longitude position deviation	m
D_ψ	yaw orientation deviation	rad or $^{\text{o}}$
$e(t)$	error signal	-
f	focal length	px
$f(t)$	control signal	-
F	resultant thrust N	
F_{M1}	motor one thrust	N
F_{M2}	motor two thrust	N
F_{M3}	motor three thrust	N
F_{M4}	motor four thrust	N
g	gravitational acceleration; or grayscale value	9.807m/s^2; or -
g_{new}	new grayscale value	-
g_{thr}	grayscale threshold value	-
h	hue value	-
h_{max}	maximum hue value	-
h_{min}	minimum hue value	-
H	height	px
I_x	mass moment of inertia about x-axis	kg.m^2
I_y	mass moment of inertia about y-axis	kg.m^2
I_z	mass moment of inertia about z-axis	kg.m^2
k	camera correlation coefficient	m.px
K_D	derivative gain	-
K_I	integral gain	-
K_P	proportional gain	-
l	lightness value	-
l_{max}	maximum lightness value	-
l_{min}	minimum lightness value	-
m	mass; or arbitrary virtual image distance	kg; or px

$\dot{m}$	arbitrary virtual image velocity	px/s
μ	air molar mass	0.02896kg/mol
n	arbitrary actual distance	m
$\dot{n}$	arbitrary, actual velocity	m/s
p	pressure; or pixel colour value	Pa; or -
p_0	pressure at the lower troposphere limit	101350Pa
$\%p_a$	actual percentage difference	%
$\%p_m$	magnitude percentage difference	%
ϕ	roll rotation	rad or ^{o}
$\ddot{\phi}$	roll angular acceleration	rad/s^2
ψ	yaw rotation or orientation	rad or ^{o}
ψ_i	initial yaw orientation	rad or ^{o}
ψ_q	quadrotor yaw angle	rad or ^{o}
ψ_t	target yaw angle	rad or ^{o}
$\ddot{\psi}$	yaw angular acceleration	rad/s^2
r_H	longitude resolution	px
r_W	latitude resolution	px
$r(t)$	reference signal	-
R	universal gas constant	8.315kg.m^2/(s^2.mol.K)
s	saturation value	-
s_{max}	maximum saturation value	-
s_{min}	minimum saturation value	-
t	time	s
T	temperature	K
T_0	temperature at the lower troposphere limit	288.15K
T_{M1}	motor one torque	N.m
T_{M2}	motor two torque	N.m
T_{M3}	motor three torque	N.m
T_{M4}	motor four torque	N.m
$T_{\phi,+}$	plus-mode roll torque	N.m
$T_{\phi,\times}$	cross-mode roll torque	N.m
$T_{\psi,+}$	plus-mode yaw torque	N.m
$T_{\psi,\times}$	cross-mode yaw torque	N.m
$T_{\theta,+}$	plus-mode pitch torque	N.m
$T_{\theta,\times}$	cross-mode pitch torque	N.m
τ	incremental time placeholder	s
τ_0	temperature gradient within the lower troposphere	-0.0065K/m
θ	pitch rotation	rad or ^{o}

$\ddot{\theta}$	pitch angular acceleration	rad/s^2
u	virtual image latitude coordinate or distance	px
u_i	initial latitude coordinate	px
v	virtual image longitude coordinate or distance	px
v_i	initial longitude coordinate	px
W	width	px
x	actual latitude coordinate or distance	m
x_q	quadrotor latitude position	m
x_t	target latitude position	m
y	actual longitude coordinate or distance	m
y_q	quadrotor longitude position	m
y_t	target longitude position	m
z	altitude	m
z_{sea}	altitude relative to sea level	m
$\ddot{z}$	acceleration along z-axis	m/s^2

Throughout the report, the relevant description and units are displayed with the symbols presented in equations. If there is a discrepancy or confusion between the displayed information and this listed information, the displayed information should be assumed to supersede this listed information.

LIST OF ACRONYMS

The general acronyms used, with a corresponding description, are as follows:

3D	Three-Dimensional
AMD	Advanced Micro Devices
API	Application Programming Interface
BGR	Blue, Green, and Red
BLE	Bluetooth Low Energy
CCD	Charge-Coupled Device
CPPM	Chaotic Pulse Position Modulation
CPU	Central Processing Unit
CMOS	Complementary Metal-Oxide Semiconductor
CSI	Camera Serial Interface
D	Derivative
DC	Direct Current
DOF	Degrees Of Freedom
EEPROM	Electrically Erasable Programmable Read Only Memory
ESC	Electronic Speed Controller
FAST	Feature from Accelerated Segment Test
FPS	Frames Per Second
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
HLS	Hue, Lightness, and Saturation
HP	Hewlett-Packard
HSI	Human System Interactions
HVS	Hue, Value, and Saturation
I	Integral
IBVS	Image-Based Visual Servoing
ICUAS	International Conference on Unmanned Aircraft Systems
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
I2C	inter-Integrated Circuit
LAN	Local Area Network
LED	Light Emitting Diode
LiPo	Lithium-Polymer

MIMO	Multiple-Input Multiple-Output
MOCAP	Motion Capture
NIR	Near Infra-Red
NoIR	No Infra-Red
P	Proportional
PCM	Pinhole Camera Model
PI	Proportional-Integral
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
PBVS	Position-Based Visual Servoing
PPE	Personal Protective Equipment
PRX	Primary Receiver Mode
PTX	Primary Transmitter Mode
PWM	Pulse Width Modulation
QTM	Qualisys Track Manager
RAM	Random Access Memory
RDP	Remote Desktop Protocol
RGB	Red, Green, and Blue
RTOS	Real-Time Operating System
SD	Secure Digital
SIFT	Scale-Invariant Feature Transform
SPF	Seconds Per Frame
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSH	Secure Shell
SURF	Speeded-Up Robust Features
ToF	Time-of-Flight
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
WiFi	Wireless Fidelity

Throughout the report, the relevant description is displayed with the first appearance of uncommon acronyms. If the acronym is commonly known or may be better known by the acronym itself, the acronym is used directly with the first appearance, such is the case for USB and HDMI as examples.

1 INTRODUCTION

In industries requiring dynamic surveillance, inspection, or exploration, it is often useful to accurately track moving targets with an unmanned aerial vehicle (UAV) using visual servoing. These industries include activities involving photography, cinematography, traffic and transportation, surveying, mapping, search and rescue, navigation, and agriculture, with increasing presence in the evolution of the fourth industrial revolution [1]. So, it is necessary to establish an initial basis for visual servoing using an UAV and infer if there are reasons for further investigation into moving target tracking.

1.1 BACKGROUND

A multirotor is generally regarded as an unmanned rotorcraft or rotary-wing aircraft which generates thrust and motion using two or more fixed-pitch rotors as actuators and allows for simpler rotor mechanics required for flight control using sensors, as compared to other types of aircraft [2]. A quadrotor is a multirotor with four rotors and propellers, where the size may typically vary with propeller diameters from 45mm for small aircraft to 480mm for large aircraft and the cost may relatively vary from economical for hobbies to extravagant for specialised activities [2]. A variety of multirotor layouts, including a quadrotor, are seen in Figure 1. (Occasionally, the terms "multicopter" and "quadcopter" are used to refer to a multirotor or quadrotor respectively, but from linguistic perspectives and origins, it is actually more accurate to use the terms "multirotor" and "quadrotor" respectively [3]).

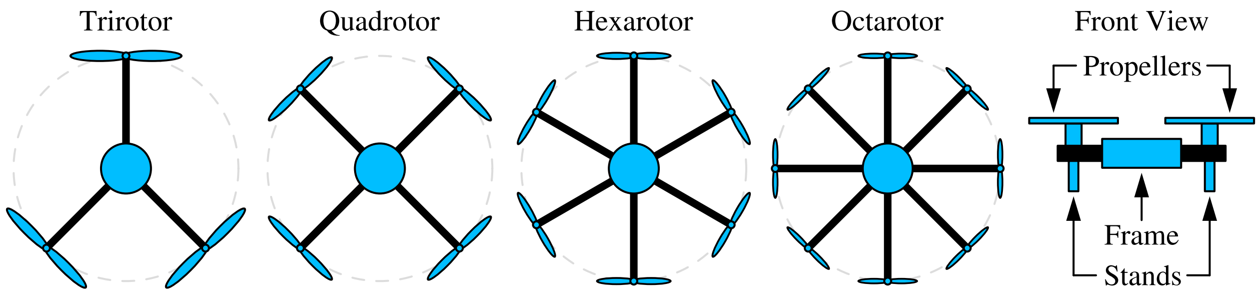


Figure 1: Examples of the common multirotor layouts for three (left), four (left-middle), six (middle), and eight (right-middle) rotors with a general layout of a front view (right).

With regards to a quadrotor, visual servoing for moving target tracking is essentially the technique in which the motion of the quadrotor is controlled by commands generated based on a passive vision-based sensor. This visual servoing may be in the form of end-point closed-loop or eye-in-hand control with a camera mounted on-board the quadrotor to observe the relative position and motion of a target. An alternative method of visual servoing is end-point open-loop or eye-to-hand control, where multiple external cameras are fixed in the surroundings and form a motion capture system observing the absolute position and motion of a target. These methods are demonstrated in Figure 2.

To distinguish within visual servoing, target detection is concerned with using a captured image to identify a designated target, while target tracking is concerned with recognising the movements of the

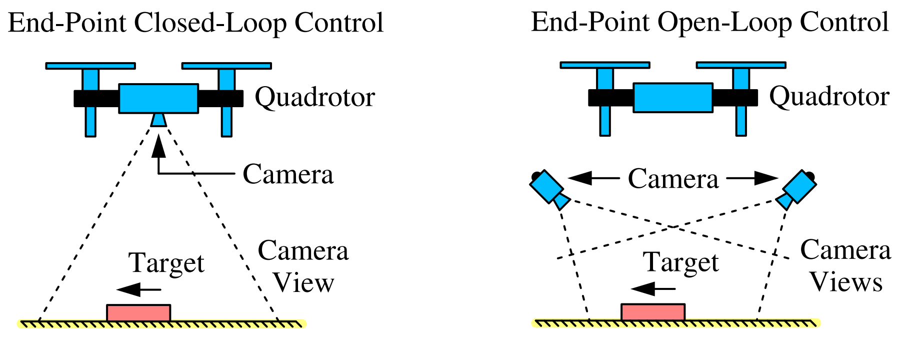


Figure 2: Examples of end-point closed-loop control (left) and end-point open-loop control (right).

designated target. Particularly, the target is identified through image processing with computer vision techniques based on distinct features such as colours, areas, edges, corners, and centroids. However, this becomes more difficult with a moving target where variable factors become apparent, which can possibly include changes in the target movement, colour pattern, and structure or geometry, with concurrent changes in the background environment and camera tilting orientations.

1.2 MOTIVATION

Due to their growing importance and utility, quadrotors are an important part of the robotics and engineering fields of research with the incorporation of knowledge from mechanics, aviation, electronics, and computer science [4]. Additionally, the incorporation of visual servoing with quadrotors has become highlighted when signals from global positioning systems (GPS) are unreliable or unavailable in indoor environments [5, 6]. The various methods of implementing visual servoing through computer vision have been extensively developed and show promising results with regards to target detection - this is especially evident in systems with end-point closed-loop control, as successfully demonstrated by Rabah et al. [1], Dunkley [2], Karlsson [6], and Kendall et al. [7] for examples.

However, there are still challenges with regards to implementing visual servoing through end-point open-loop control with only a single camera, such that a quadrotor is able to achieve optimal moving target tracking in three-dimensions while maintaining fully autonomous flight without manual inputs from a user. This is a difficult challenge as it requires the extraction of three-dimensional information from a two-dimensional vantage point. Moreover, a significant focus of published research into computer vision is only involved with isolating or segmenting targets for detection, while the more specific research into end-point open-loop control has been to integrate flight control and indirect computer vision techniques using motion capture systems with multiple cameras triangulating position, which can be expensive and difficult to access, and utilises systems with excess computational effort for image processing without taking into account the possibility of substantial limits on the available computational effort [1, 8, 9]. This is discussed by Mahony et al. [3] using a Vicon motion capture system requiring considerable resources for state estimation of an Ascending Technologies quadrotor; and Dunkley [2] and Mack et al. [10] utilised a Microsoft Kinect as an economic alternative to detect and control a Bitcraze Crazyflie quadrotor, but this features two cameras and a depth sensor while still accessing excess computational effort on their ground control laptops - unlike most of the end-point closed-loop control studies, these studies do not consider tracking moving targets.

Furthermore, a critical disadvantage of end-point open-loop control is the requirement for the external cameras to already be set up, where this would present an obstacle if operating in an unfamiliar environment since the setting up of multiple cameras may be tedious or inapplicable, but the use of a single camera would be much more efficient. Thus, there is a need to investigate the detection and tracking of a moving target using end-point open-loop control with only a single camera, which will reinforce the success of published research with fundamental similarities as a consequence.

1.3 PROBLEM STATEMENT

It is necessary to develop and assess the effectiveness of a control system using end-point open-loop visual servoing with a single camera to autonomously perform real-time moving target tracking. The camera is to be fixed in the surroundings and the tests should be initially performed in a controlled indoor environment. The detection of the target will be carried out using computer vision through various techniques, but this needs to be processed using hardware with limited or marginal computational effort which will likely require a self-derived algorithm utilising lightweight image processing. The tracking of the target will be performed such that a quadrotor effectively follows or mirrors the movement of the target in a manner where the position of the target can be seen to be mapped directly by the position of the quadrotor. This tracking should focus on the three degrees of translational freedom, but it may be possible to also track the orientation of the target with regards to yaw or rotation in the horizontal plane, depending on the arrangement. The relative distance error between the position of the target and position of the quadrotor should be minimised within an acceptable range for successful implementation. The effectiveness can then be measured based on the lag time, robustness, relative distance fluctuations, and ability to track the target at varying speeds. For supplement, this should be achieved using mostly inexpensive and open-source hardware and software.

Thus, a research question is proposed: *is it possible to implement end-point open-loop visual servoing in a three-dimensional space using a fixed single camera for moving target tracking with limited computational effort for image processing and a quadrotor following the movements of the target?*

2 LITERATURE REVIEW

To establish a basis from which to progress, existing research and resources for quadrotors and computer vision with regards to visual servoing for target tracking needs to be reviewed and documented. It is also essential to understand certain theory and ideas behind the apparatus and methodology which will be used, such that these concepts can be identified if they do arise even though they are not expected to be direct factors during the tests and in the results. Although many of the concepts discussed are generally applicable to visual servoing through other means as well, it should be noted that the information will be presented primarily in relation to quadrotors and target tracking.

2.1 VISUAL SERVOING

Due to the aim of the research, there will be a focus on visual servoing through end-point open-loop control, rather than end-point closed-loop control. This may employ position-based visual servoing or image-based visual servoing. For position-based visual servoing, information is extracted from an image based on features in the image and used to reconstruct the current three-dimensional position of the target, where this can then be combined with the knowledge about the three-dimensional position of the quadrotor to generate control and actuation corrections such that the quadrotor moves to the desired position to follow the target [5]. For image-based visual servoing, an error signal is computed directly from the features in an image, without performing a three-dimensional reconstruction, where this error signal is then used to generate control and actuation corrections to match the current image features with a desired arrangement [5]. It is also possible to use a mixed scheme with a combination of aspects from both position-based visual servoing and image-based visual servoing [5].

Considering position-based visual servoing, the primary advantage is the convenience of generating a setpoint as an absolute coordinate in three-dimensional space, but the primary disadvantage is the uncertainty in the results of the three-dimensional reconstruction since there is a dependence on the optical parameters, calibration of the vision system, and requisite knowledge about the dimensions of the target prior to implementation. Considering image-based visual servoing, the primary advantage is the insensitivity of the results to camera calibration, but the primary disadvantage is the necessity to relate the features in the image to actions through non-linear or empirical relationships. The methods of position-based visual servoing and image-based visual servoing are illustrated in Figure 3 for an image, where these processes can then be repeated for each frame in a sequence.

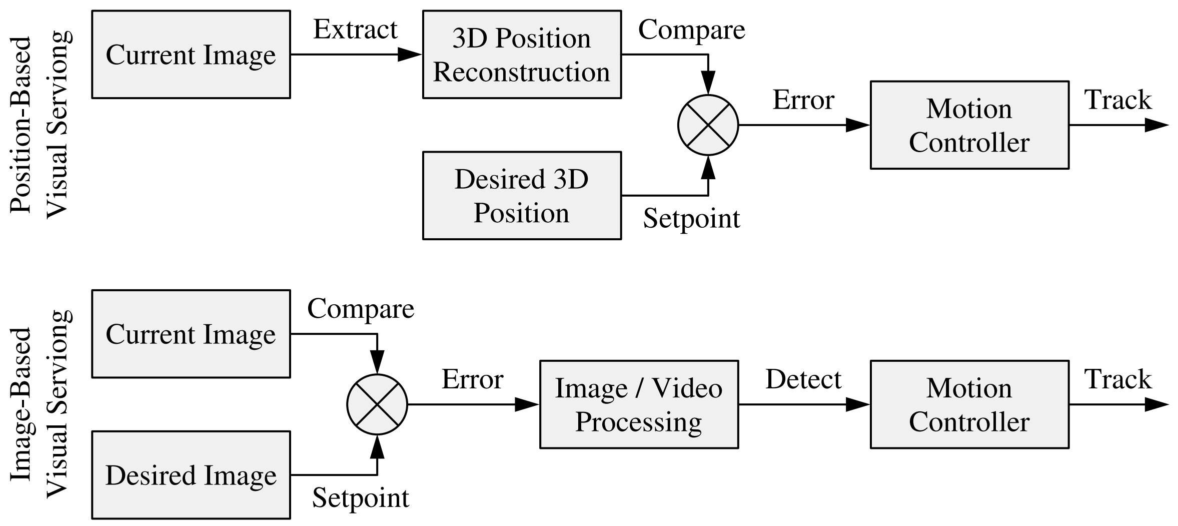


Figure 3: Comparison between the basic methods of position-based visual servoing (top) and image-based visual servoing for an image - this process is then repeated for each frame in a sequence.

2.2 QUADROTOR DYNAMICS

As briefly mentioned, a quadrotor is a rotorcraft with four rotors which have fixed pitches and are used to create motion by varying the thrust generated at each rotor [2]. The rotors usually consist of propellers or blades in the form of aerofoils which are mounted on high-speed direct current (DC) motors, while the other parts of a typical quadrotor include a battery as the power source, various sensors to estimate real-time position or velocity, cross frame holding the parts, and flight controller for stabilisation [2]. The very basic sensors include an inertial measurement unit, where a gyroscope stabilises rotational motion and an accelerometer detects the current orientation - although these sensors alone may lead to cumulative errors over time which will result in drifting with an ever-increasing difference between the estimated state and actual state [2]. More advanced and accurate sensors include on-board ultrasonic or laser range finders to measure distances, GPS for precise location, barometers to measure air pressure changes with altitude, magnetometers to determine which direction is north, and monocular or stereo camera for measurements relative to the environment being observed [2]. With these parts, a quadrotor is able to vertically take-off and land with exceptional abilities to hover at a fixed position and simultaneously manoeuvre vertically, latitudinally, and longitudinally.

With regards to the motion of the quadrotor, there are three degrees of freedom for translational displacement, which include forward or backward, left or right, and up or down; and three degrees of freedom for rotational displacement, which include roll, pitch, and yaw. These motions can be illustrated on a conventional coordinate system relative to the quadrotor, as shown in Figure 4, where translation along the x-axis, y-axis, and z-axis correspond to the translation displacements respectively and rotation about the x-axis, y-axis, and z-axis correspond to the rotational displacements respectively. Thus, there is a total of six degrees of freedom for the quadrotor.

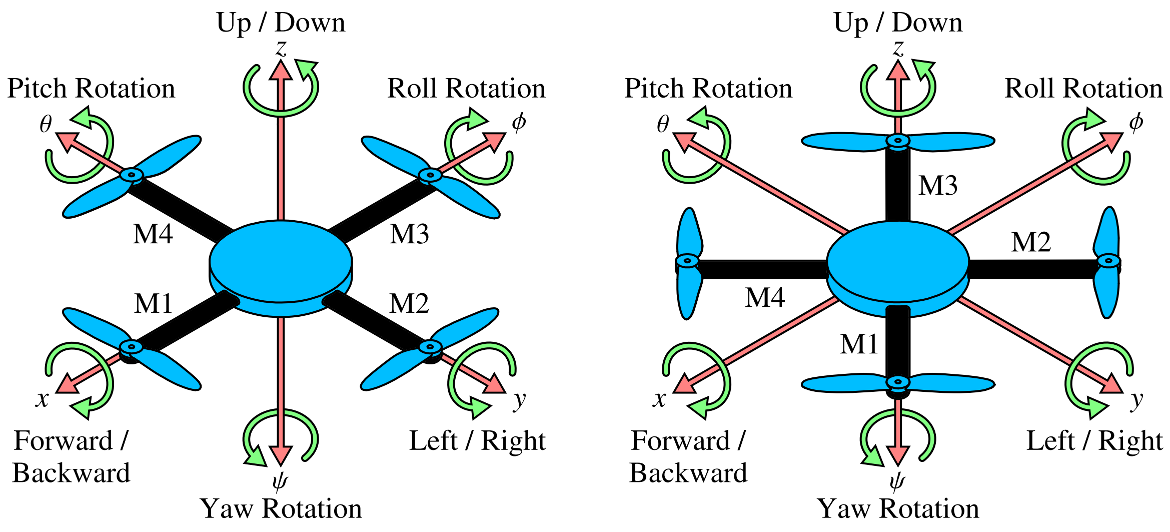


Figure 4: Coordinate system relative to the quadrotor showing the three translational and three rotational displacements. The quadrotor may operate in a plus-mode (+) (left) or cross-mode (×) (right).

The quadrotor is then able to generate these degrees of freedom by varying the thrust generated by each of the four rotors, as seen in Figure 5 and Figure 6, such that a simple mechanical design can be used with a complex controller design. With regards to forces, each rotor produces an upwards thrust, torque about the centre of rotation, and drag opposite to the motion of the quadrotor. To move up or down along the z-axis, the thrust simply needs to be increased or decreased respectively. To move forward or backward along the x-axis, the thrust needs to be maintained while pitching clockwise or counter-clockwise about the y-axis respectively. To move left or right along the y-axis, the thrust needs to be maintained while rolling counter-clockwise or clockwise about the x-axis respectively.

The roll and pitch control will depend on whether the quadrotor is using a plus-mode (+) or cross-mode (×), but the manner of operation is identical where a resultant torque is produced to induce rolling or pitching. Considering a plus-mode, the roll with rotation about the x-axis is controlled by the left and right rotors, where the left rotor thrust needs to be greater than the right rotor thrust to roll clockwise and the right rotor thrust needs to be greater than the left rotor thrust to roll counter-clockwise; and the pitch with rotation about the y-axis is controlled by the front and rear rotors, where the rear rotor thrust needs to be greater than the front rotor thrust to pitch clockwise and the front rotor thrust needs to be greater than the rear rotor thrust to pitch counter-clockwise (clockwise and counter-clockwise directions are taken relative to the perspective of the quadrotor along the x-axis for rolling and y-axis for pitching). For the cross-mode, the side selection of rotors for rolling and pitching is identical, but the rotors will be used in pairs without intermediate rotors.

Finally, yawing with rotation about the z-axis is created using the resulting torque produced from the rotors spinning against the air, where the direction of this torque is in the opposite direction to the direction of the combined angular velocity from the rotors - for flight without yawing, this effect is

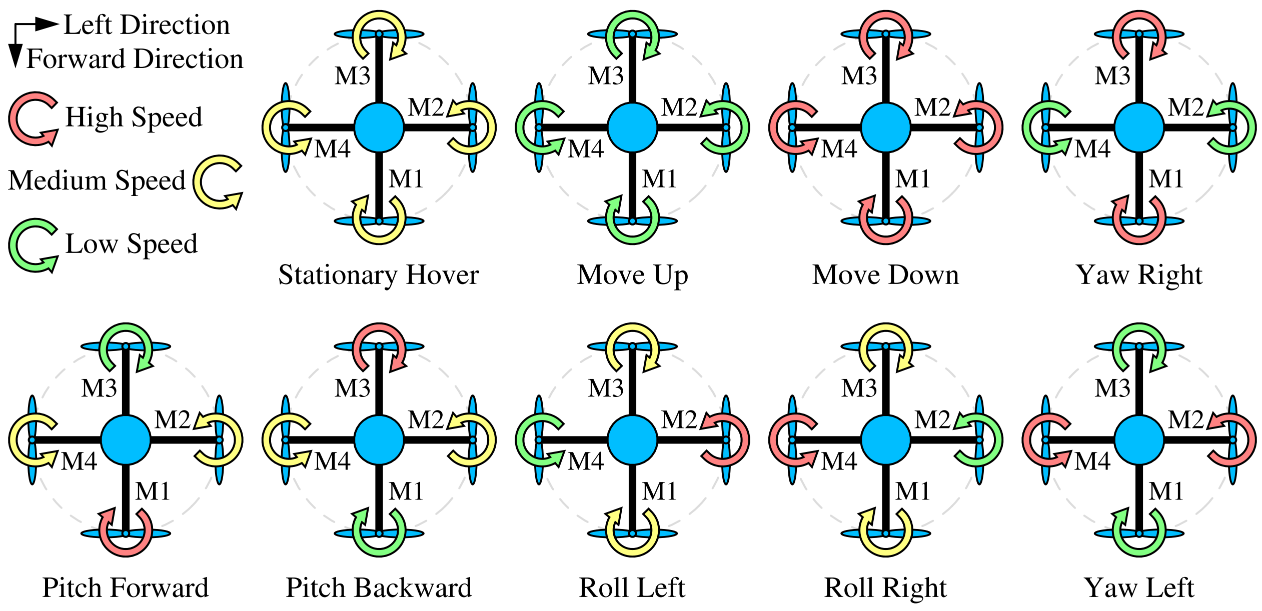


Figure 5: Plus-mode (+) rotor arrangement to enable the translational and rotational displacements.

cancelled by letting the two diagonally-opposite rotors spin clockwise while the other two diagonally-opposite rotors spin counter-clockwise [2, 8]. So, when yawing is desired in a certain direction, the angular velocities of the two rotors spinning in the opposite direction are decreased and the angular velocities of the two rotors spinning in the same direction are increased, which will create rotation about the *z*-axis such that the total thrust remains constant without rolling or pitching [2, 8].

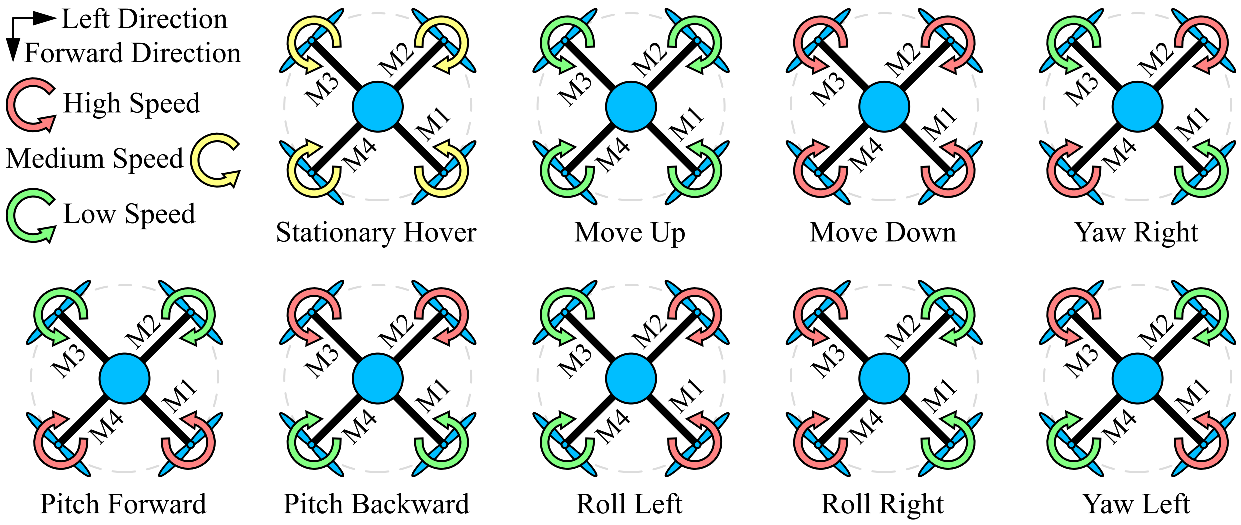


Figure 6: Cross-mode (×) rotor arrangement to enable the translational and rotational displacements.

For further understanding, a simplified analytical model of a quadrotor is given by Equation 1, which describes the resultant thrust acting on the quadrotor from the thrust of each rotor, and Equation 2 to Equation 4 for plus-mode or Equation 5 to Equation 7 for cross-mode, which respectively describe the resultant torque acting on the quadrotor while rolling, pitching, and yawing. These relationships are fairly basic and present an ideal model, which can be implemented using the inherent relationships between thrust, torque, and supplied voltage for each rotor determined experimentally [11].

$$F = F_{M1} + F_{M2} + F_{M3} + F_{M4} \longrightarrow F - mg = m\ddot{z} \tag{1}$$

$$T_{\phi,+} = d_{+}(F_{M4} - F_{M2}) = I_x\ddot{\phi} \tag{2}$$

$$T_{\theta,+} = d_{+}(F_{M3} - F_{M1}) = I_y\ddot{\theta} \tag{3}$$

$$T_{\psi,+} = T_{M1} + T_{M3} - T_{M2} - T_{M4} = I_z\ddot{\psi} \tag{4}$$

$$T_{\phi,\times} = d_{\times}((F_{M3} + F_{M4}) - (F_{M1} + F_{M2})) = I_x\ddot{\phi} \tag{5}$$

$$T_{\theta,\times} = d_{\times}((F_{M2} + F_{M3}) - (F_{M1} + F_{M4})) = I_y\ddot{\theta} \tag{6}$$

$$T_{\psi,\times} = T_{M1} + T_{M3} - T_{M2} - T_{M4} = I_z\ddot{\psi} \tag{7}$$

Where F, resultant thrust, N; F_{M1}, motor one thrust, N; F_{M2}, motor two thrust, N; F_{M3}, motor three thrust, N; F_{M4}, motor four thrust, N; m, mass, kg; g, gravitational acceleration, 9.807m/s^2; $\ddot{z}$, acceleration along z-axis, m/s^2; $T_{\phi,+}$, plus-mode roll torque, N.m; d_{+}, plus-mode moment arm, m; I_x, mass moment of inertia about x-axis, kg.m^2; $\ddot{\phi}$, roll angular acceleration, rad/s^2; $T_{\theta,+}$, plus-mode pitch torque, N.m; I_y, mass moment of inertia about y-axis, kg.m^2; $\ddot{\theta}$, pitch angular acceleration, rad/s^2; $T_{\psi,+}$, plus-mode yaw torque, N.m; T_{M1}, motor one torque, N.m; T_{M2}, motor two torque, N.m; T_{M3}, motor three torque, N.m; T_{M4}, motor four torque, N.m; I_z, mass moment of inertia about z-axis, kg.m^2; $\ddot{\psi}$, yaw angular acceleration, rad/s^2; $T_{\phi,\times}$, cross-mode roll torque, N.m; $d_\times$, cross-mode moment arm, m; $T_{\theta,\times}$, cross-mode pitch torque, N.m; and $T_{\psi,\times}$, cross-mode yaw torque, N.m.

When considering the representation of a quadrotor or another body with six degrees of freedom in three-dimensional space, it is possible to use a reference frame with a coordinate system either fixed to an observer or fixed to the quadrotor [2, 6, 11]. For an observer-fixed reference frame, it is assumed that the observer as the origin is arbitrarily on the surface of the Earth, which is seen to be flat and stationary. For a body-fixed reference frame, the centre of gravity of the quadrotor is conditioned as the origin, which is convenient when accounting for inertial properties. Followingly, the conversion between the observer-fixed and quadrotor-fixed reference frame can be seen as a relative transformation which is divided into the difference in distance along each axis and then the change in rotational angles described by Figure 7 to produce a rotation matrix in Equation 8 [2, 6, 11].

$$D = \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\theta)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta) \\ \cos(\phi)\sin(\theta)\cos(\phi) + \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{8}$$

Where θ, pitch rotation, rad or $^{\text{o}}$; ψ, yaw rotation, rad or $^{\text{o}}$; and ϕ, roll rotation, rad or $^{\text{o}}$.

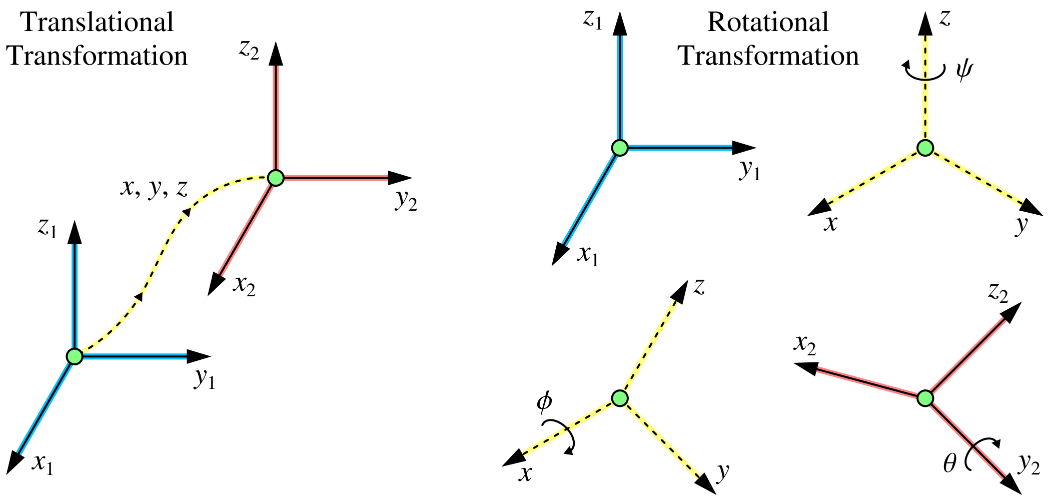


Figure 7: Demonstration of the conversion between two reference frames with translation (left) and rotation (right). (In this context, the rotational angles can also be referred to as Euler angles).

2.3 RASTER IMAGES

A raster image or bitmap is composed of multiple pixels with the amount of pixels in the horizontal and vertical direction forming a rectangular grid and resolution. The process of rasterisation involves representing a real object or vector image in the form of a raster image, where the continuous data describing the real object or vector image is converted into discrete data for representation through the raster image. However, there will always be information lost during rasterisation due to the discretization error arising from a limit in the ability to completely render the continuous data. The degree to which information is lost is dependent on the construction and resolution of the raster image, which is a principal concern for lower resolutions when features reduce in quality and cannot be resolved. The process of rasterisation and rendering a raster image is demonstrated in Figure 8.

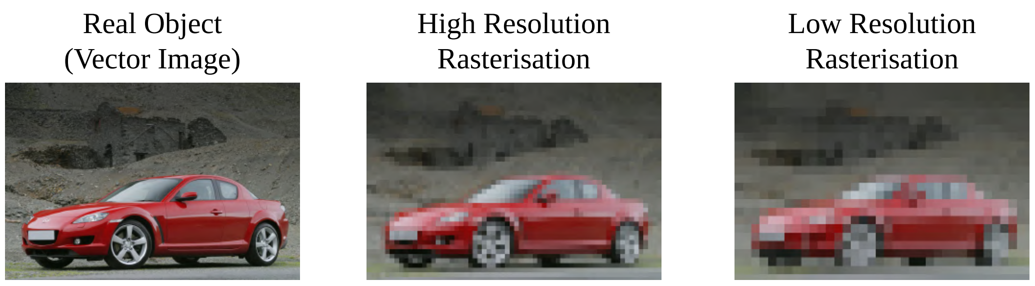


Figure 8: Demonstration of the rasterisation of a real object with high and low resolutions. (Although the original image of the real object is actually a raster image, its resolution is sufficiently large to avoid raster artefacts and it serves as a view from an observer for this demonstration).

2.4 PINHOLE CAMERA MODEL

In fundamental terms, digital cameras capture raster images by measuring the amount and wavelength or colour of light projected onto an imaging sensor with pixels (analogue cameras will not be considered) [2]. This imaging sensor is usually a charge-coupled device (CCD) or a complementary metal-oxide semiconductor (CMOS) array [2, 6, 12]. As abridged in Figure 9, there will be rays of light emitted from each point on the target. This light will spread and be incident on the lens of the camera which will then focus the light to the corresponding points on the imaging sensor. So, it is essential for the lens to be located at the exact focal length to ensure the light is focussed to a singular point, rather than distorted or blurred discs which create an unclear image [2].

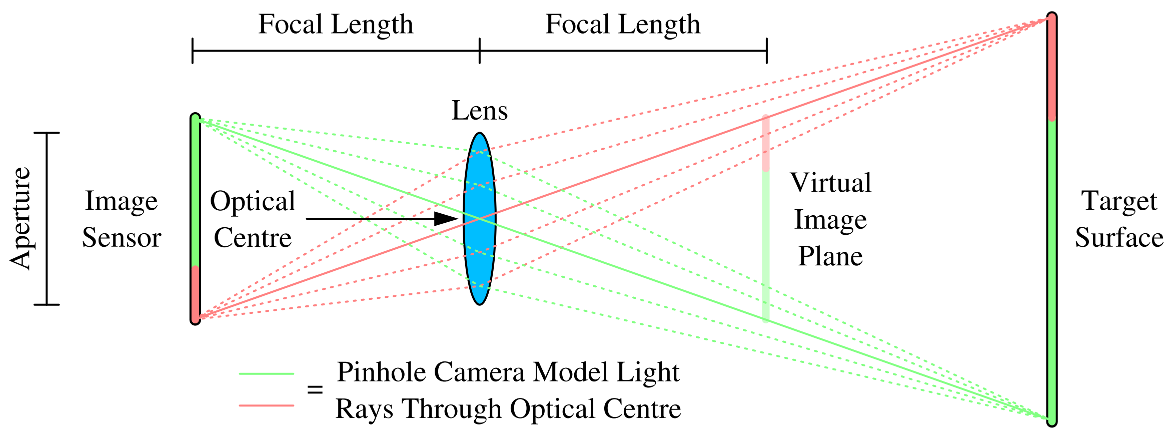


Figure 9: Simplification of a model for a pinhole camera with a lens and imaging sensor.

For simplification while maintaining reasonable accuracy, a pinhole model can be considered for the camera, where it is assumed that the rays of light only pass through the optical centre of the lens or, in other words, the rays of light not passing through the optical centre of the lens are neglected - this is equivalent to an infinitesimal point as the aperture [2]. With this model, each ray of light can be seen to travel in a straight line from the target to the imaging sensor and, for final simplification, the virtual image plane can be considered with the uninverted projection [2]. Thus, these assumption allow for the image to be regarded as a graphically equivalent description of the observed environment without distortions or blurring, which is fairly valid as long as the lens is accurately located at the focal length.

Using the pinhole model and considering the virtual image plane, the associated relationship between distances in the image can be related to actual distances in three-dimensional space. This is essentially based on geometrically similar triangles formed by the rays of light between the actual point, projection on the virtual image plane, and optical centre of the lens, as explained in Figure 10 to produce Equation 9 [2, 6]. As a result, Equation 10 describes the latitude relationships and Equation 11 describes the longitude relationship, where the focal length is a parameter of the camera and the virtual image coordinates can be found from the image. Thus, the pinhole model can be seen as a first-order approximation for the mapping of the observed environment. (If the real image plane is considered, it is either necessary to redefine the directions or use a negative sign with the relationships).

$$\frac{f}{z} = \frac{u}{x} = \frac{v}{y} \tag{9}$$

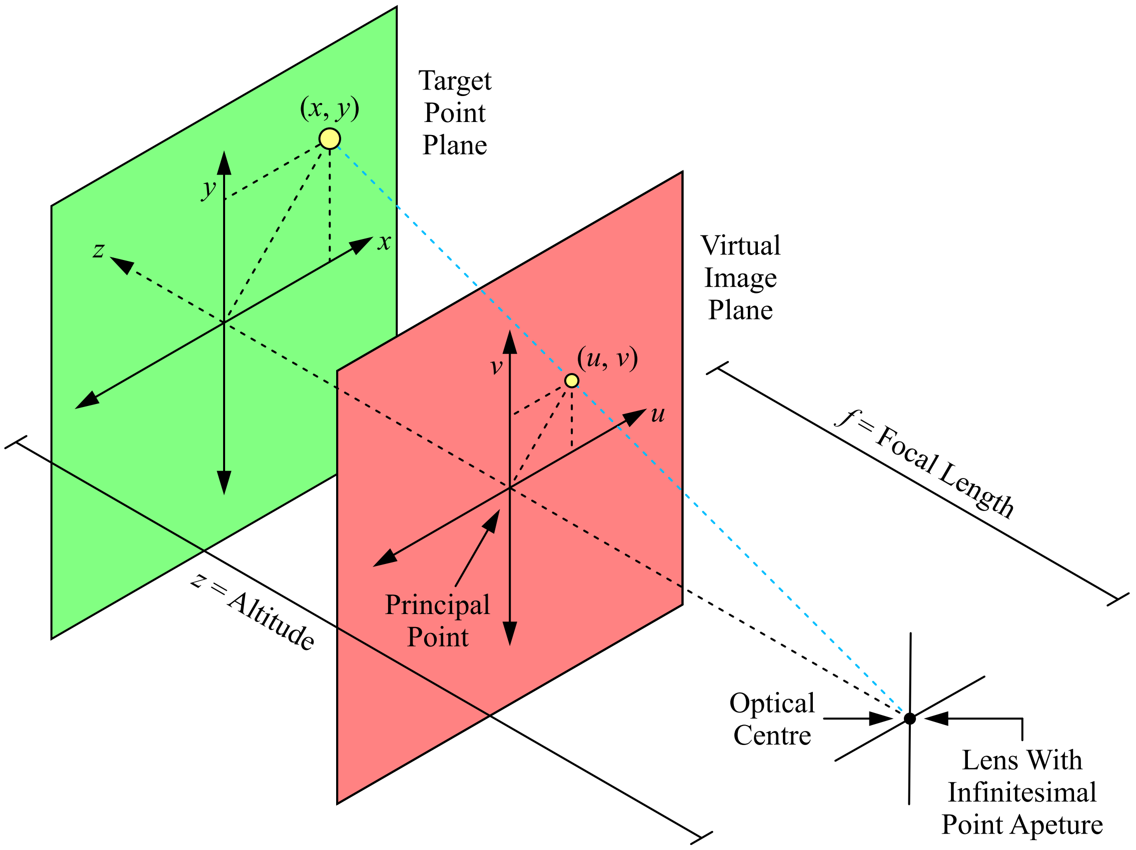


Figure 10: Geometric relationships resulting from the pinhole camera model.

$$u = f\,\frac{x}{z} \tag{10}$$

$$v = f\,\frac{y}{z} \tag{11}$$

Where f, focal length, px; z, altitude, m; u, virtual image latitude coordinate, px; x, actual latitude coordinate, m; v, virtual image longitude coordinate, px; and y, actual longitude coordinate, m.

If the pinhole model is valid, straight lines in reality are projected as straight lines on the imaging sensor - rectilinearity is preserved [2]. If the pinhole model is not valid due to excessive distortions, these distortions will create radial and tangential inaccuracies in the image. The radial inaccuracies result in straight lines appearing curved, while the tangential inaccuracies occur because the lens is not aligned perfectly parallel to the imaging plane so some areas in the image appear nearer than expected. It is most common for fish-eye distortion with large radial exaggerations towards the edges of the field of view, as described by Figure 11, where this distortion is usually more pronounced on wide angle lenses with large fields of view which capture a greater portion of the environment [2]. The distortion present in the actual images will need to be tested with the camera and deemed to be insignificant without the need for compensation or significant with the need for compensation.

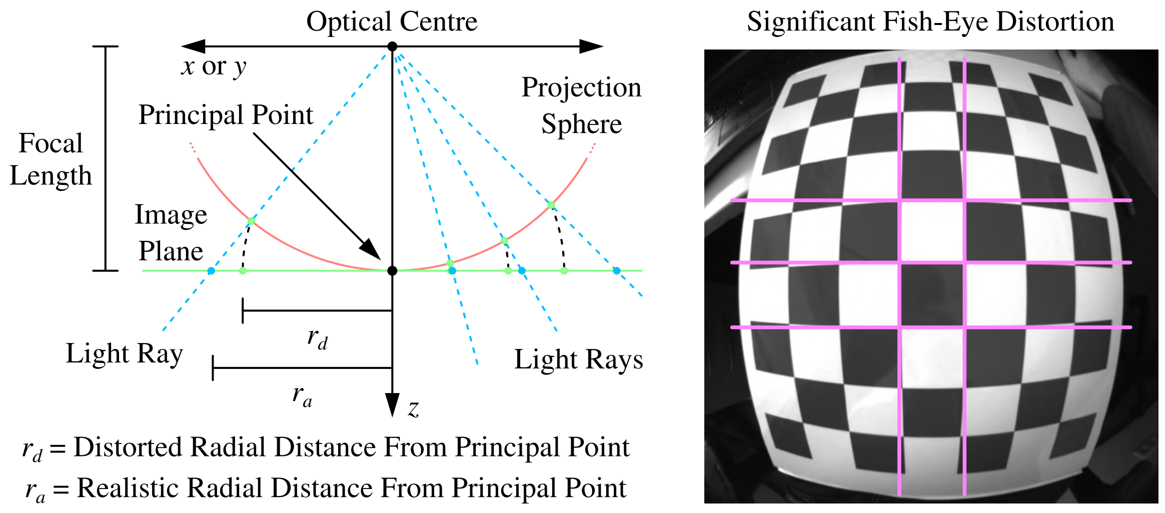

Optical Centre
x or y
Projection Sphere
Principal Point
Focal Length
Image Plane
Light Ray
r_d
r_a
Light Rays
z
r_d = Distorted Radial Distance From Principal Point
r_a = Realistic Radial Distance From Principal Point
Significant Fish-Eye Distortion

Figure 11: Fish-eye distortion model representing projections of rays of light in a wide field of view (left). Example of an image from a camera with very significant fish-eye distortion (right) [2].

Similarly, for the common type of camera, a rolling shutter mechanism is used and the image is not actually captured at a single instant, but rather the imaging sensor sequentially scans through the field of view either horizontally or vertically over a very short time period [2, 12]. As a result, there is an opportunity for distortions to arise due to a target moving during the scanning process, which would result in an inaccurate representation of the target in the captured image through shearing, smearing, or deforming - although it should be emphasised that the speed of the target needs to be exceptionally high for a significant effect [2, 12]. These effects are avoided with a global shutter mechanism which captures the entire field of view in a single instant, but this is usually only available on high-performance cameras. There are in-depth methods to compensate for rolling shutter, but for the operating speeds in this research, it can confidently be assumed that there will be no distortions from rolling shutter and the camera essentially performs as though it has a global shutter.

2.5 COMPUTER VISION

Computer vision focusses on gaining high-level understanding from images and basically uses a sequence of consecutive images, where each frame is processed and relevant information is extracted. The camera used for target detection and tracking may be monocular with a single lens for monoscopic vision or binocular with two lenses for stereoscopic vision. The advantage of a binocular camera is that three-dimensional information can be extracted more accurately since the environment can be compared from two vantage points, but this comes at a much higher cost. However, only a monocular camera is available and, so, only monoscopic vision from a single vantage point will be considered. Notably, other sensors could actually be used to achieve a similar overall result of target detection and tracking, with methods based on sound, lasers, optics, or tactile interaction [2].

The image from the camera will require processing in order to detect the target in the image, with examples of this processing seen in Figure 12. A colour image is described by an array of red, green, and blue (RGB) 8-bit integers for each pixel between 0 for each minimum and 255 for each maximum (alternative colour modes are available which offer advantages in different situations, such as hue, value, and saturation (HVS)). The image can be converted to grayscale where each pixel is described by a single scalar as an 8-bit integer for each pixel between 0 for black and 255 for white, which is usually performed to reduce the required computational effort when processing an image [13]. Blurring can also be performed through convolving the image with a low-pass filter, which is usually employed to aid in noise reduction [13]. Thresholding aims to provide a simple method of segmenting a grayscale image, where a binary image is created using only 0 as black, which is applied when the value of a pixel is below a threshold value, and 1 as white, which is applied when the value of a pixel is above a threshold value [13]. After applying a threshold, a morphology transformation can be performed to further noise reduction - this uses an erosion transformation to remove noise of white value or dilation transformation to connect regions of white value [13].

To develop basic target detection and shape analysis, there are existing algorithms for detecting edges using high-pass or gradient filters, such as the Canny edge detection which is a multi-stage algorithm with noise reduction, gradient intensity finding, non-maximum suppression, and hysteresis thresholding [13]. Alternatively, it is possible to implement contour finding to locate curves joining the continuous points which have the same colour or intensity along a boundary - these curves can then be used to analyse the centroid, area, orientation, and other characteristics of the target [13]. The level of image processing can vary depending on the situation, where minimal processing will reduce the delay before the response commands can be executed, but excessive processing will be more reliable with more computational effort and heavier algorithms for more accurate results [14].

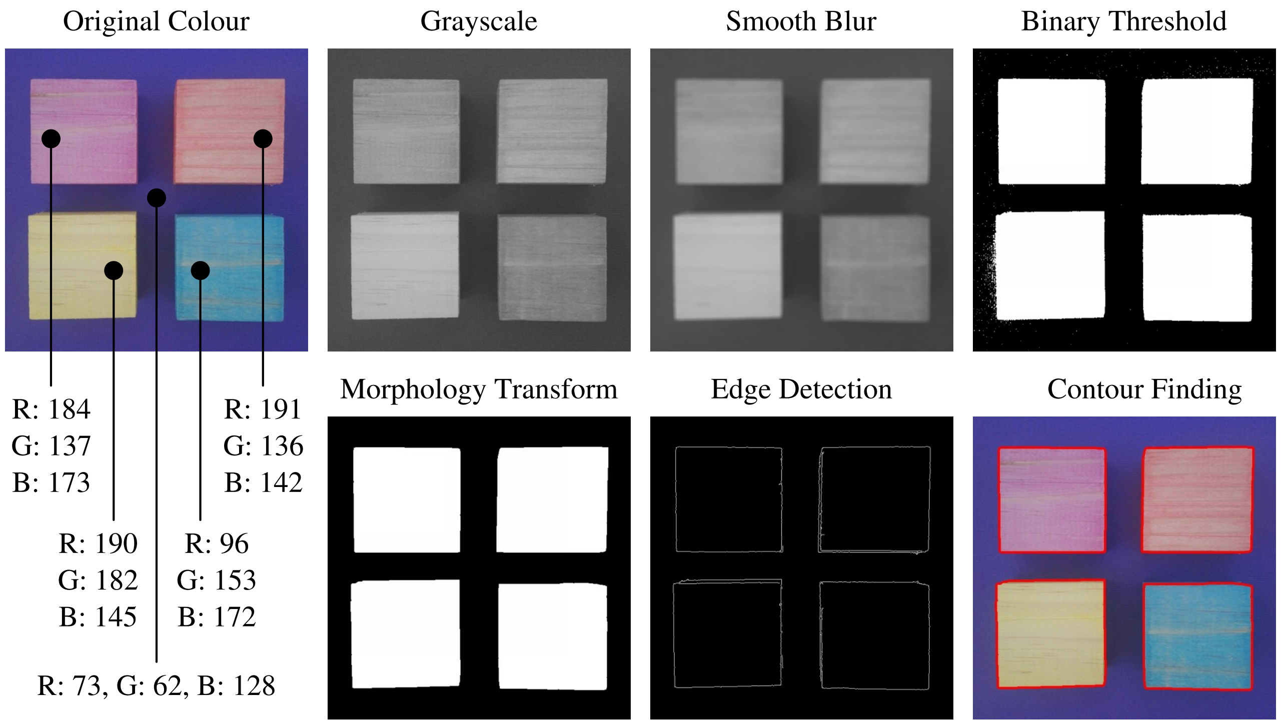


Figure 12: Example of an image from a camera in original colour and with processing through grayscaling, blurring, thresholding, morphology transforming, edge detecting, and contour finding.

For more specific target detection, target recognition is considered to be concerned with recognising the designated target based on learnt features. There are common algorithms which include colour-based matching, template matching, meanshift, camshift (continuously adaptive meanshift), and corner detection. The overall techniques and methods of these algorithms are briefly explored:

- The colour-based matching algorithm simply compares the colour in patches of an image against a specific colour range for the target in an attempt to find a match within the range [13, 14]. This algorithm usually requires for the target to be a single colour and may produce false-positives when there are other objects of similar colours within the image [13, 14].
- The template matching algorithm compares patches of an image against a specific template for the target in an attempt to find a match based on a flexible threshold for success [13, 14]. For improved performance, multiple scaled and rotated templates of the target may be considered [13, 14]. However, this is likely to be ineffective when the orientation and size of the target constantly changes since many templates and increased computational effort will be required.
- The meanshift and camshift algorithms use an initial region of interest of the target separated from the background in colour mode and then this region is shifted in subsequent frames based on a local maximum of a probability distribution for the changes in the target location, orientation, and size [13, 14]. The camshift algorithm essentially builds on the meanshift algorithm with the use of an adaptive probability distribution to better account for changes in the orientation and size of the target (the meanshift algorithm uses a static probability distribution) [13, 14].

- A corner detection algorithm basically considers the change in gradient intensity where corner or blob points can be uniquely identified [2, 13]. There are various flavours of corner detection algorithms, such as the Harris, Shi-Tomasi, scale-invariant feature transform (SIFT), speeded-up robust features (SURF), or feature from accelerated segment test (FAST) [2, 13].

However, besides the colour-based matching algorithm, these algorithms generally require a large amount of computational effort and iterations with multiple frames for success and it is usually not possible to perform this processing on low-end hardware while achieving real-time performance [14].

Regardless of the detection or recognition algorithm, it is essential for the detection algorithm to have repeatability, where the same features of the target should be consistently detected in consecutive frames; distinctiveness, where the features of the target can easily be distinguished using their appearance; robustness, where detection is still possible in the presence of distortion or noise; low computational effort, where the processing must be as fast as possible for real-time operation; scale invariance, where the size of the target does not affect detection (with regards to a reasonable limit); illumination invariance, where the lighting or photometry from the environment does not affect detection; and orientation invariance, where the rotation of the target does not affect detection [2].

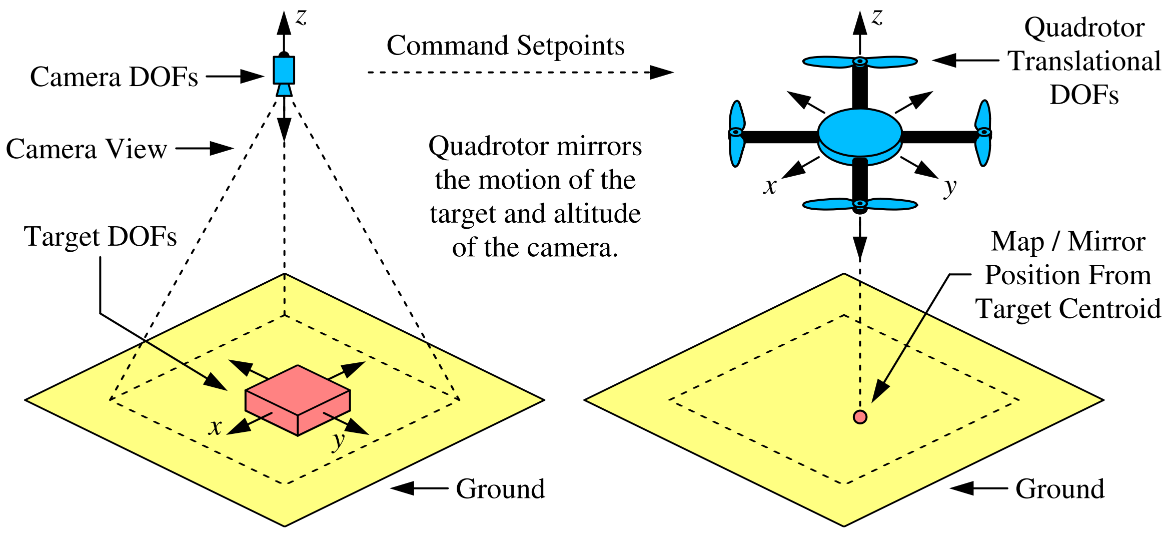
An additional computer vision technique which should be acknowledged is the use of optical flow. Instead of directly detecting a specific target, optical flow is concerned with motion detection and the changes in the patterns of features in the environment due to apparent motion caused by relative motion between the camera and environment [13]. Although it may not be useful in target detection, an optical flow sensor can be utilised to stabilise a quadrotor by discerning necessary adjustments.

2.6 LOCATION ESTIMATION

The attitude of the quadrotor refers to the angular orientation in three-dimensions and, along with the location in three-dimensions, the state of the quadrotor can then be described by the vertical altitude, position in the horizontal plane, and angular arrangement for movement. With regards to target tracking, the primary concerns are the three translational degrees of freedom, such that the desired altitude and position of the quadrotor relative to the altitude and position of the target can be determined through position-based visual servoing by processing the captured frame.

So, the captured and subsequent frames can be processed to detect the target and find the approximate area and centroid of the target as measured in terms of pixels in the frames. Thus, through a pinhole camera model, a reconstruction of the three-dimensional locations can be applied as a method to estimate the desired altitude for the quadrotor and current position of the target in terms of distances. This can be pursued with an arrangement similar to that displayed in Figure 13 - although the method of estimating the altitude is uncommon and does not directly appear in literature.

As mentioned, a complete motion capture system could be used to estimate the absolute position of the target and quadrotor, where multiple external cameras detect reflective infra-red markers attached



z
Camera DOFs
Camera View
Target DOFs
x
y
Ground
Command Setpoints
Quadrotor mirrors the motion of the target and altitude of the camera.
z
Quadrotor Translational DOFs
x
y
Map / Mirror Position From Target Centroid
Ground

Figure 13: Arrangement for end-point open-loop control through detection and tracking of a target.

to the target and quadrotor, and use triangulation to estimate the position of each marker in three-dimensions. To capture three translational degrees of freedom, it is only necessary for at least one marker to be used on a rigid body, but to capture three translational degrees of freedom and three rotational degrees of freedom, it is necessary for three or more markers to be used on a rigid body, preferably in a non-ambiguous asymmetric arrangement [2]. However, since the aim of the research is focussed on using a single inexpensive camera with a lightweight algorithm, a motion capture system will not be considered for target detection or tracking purposes - although a motion capture system is available and may be used to quantify and validate the effectiveness of the developed system.

2.7 CONTROL PRINCIPLES

The general goal in control theory is to generate a desired setpoint or reference state to achieve a matching output. The control may operate through an open-loop or closed-loop. For open-loop control, a desired setpoint is input to a controller as a reference state which generates a corresponding signal for actuators to manipulate the system appropriately, where it is then assumed that the desired setpoint is satisfied in the final output state - in other words, there is no sensor to decide whether the desired setpoint has actually been satisfied [15]. This process is seen in Figure 14. To attain successful open-loop control, the actuation must be repeatable and reliable with the ability to mitigate effects from disturbances and noise, such that these effects are predictable or become negligible [15]. If the open-loop control is not successful, the error will accumulate over time and rapidly diverge. It is also necessary to discuss closed-loop control for understanding of the flight controller on a quadrotor.

For closed-loop control, the current state of the system is accurately estimated through measurements from the sensors and this state is compared against the desired setpoint to find an error which needs to be reduced. The controller then uses this error as an input and outputs the necessary commands to correct the state of the system and bring it closer to the reference state. The error should be minimised to converge to zero as quickly as possible while the system remains stable, preferably

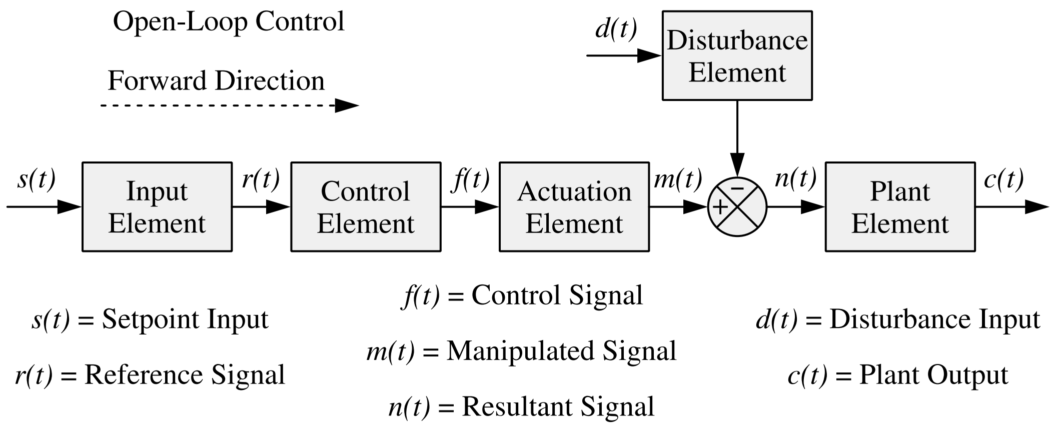


Figure 14: General schematic of an open-loop control system without feedback.

without oscillations - although it is unlikely for the error to be completely eliminated due to unknown external disturbances. This continuous closed-loop is represented in the control system shown in Figure 15, where a signal opposite to the forward direction is referred to as feedback.

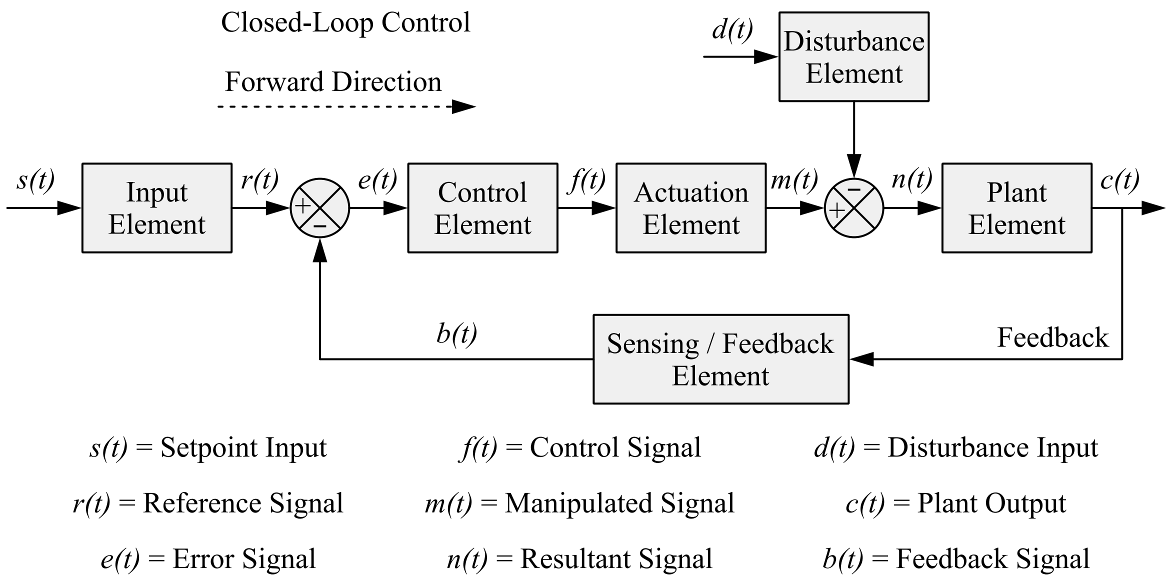


Figure 15: General schematic of a closed-loop control system with feedback.

For a quadrotor, the controller will output a velocity command to the rotors in an attempt to correct the state of the system. To obtain satisfactory results, the controller will usually operate based on a proportional-integral-derivative (PID) control method. The proportional (P) component reacts directly to the magnitude of the current error, where an increased gain will allow for faster response but may exhibit unwanted oscillations [2, 6, 15]. The integral (I) component reacts relative to the accumulation of past errors over time, where there is an increasing effect over time to reach a steady-state [2, 6, 15]. The derivative (D) component reacts relative to the rate of change of the error signal with time for a prediction of future errors, where there is a reduction of oscillations by regulating the speed of the response and increasing stability [6, 15]. These components utilise Equation 12 for the error between the setpoint and actual signals with the implementation of PID control in the analytical form of Equation 13 or numerical form of Equation 14, where constant parameters or gains are used to distribute the effects of the respective components. There are alternate methods of control, such as non-linear sliding mode control or adaptive backstepping control, but PID control has been proven to be robust and reliable when knowledge of the underlying processes are not known [1, 2, 5]. (It should be noted that, in other situations, it may only be necessary for a subset of the control components, such as only P, PD, or PI controllers - for 90% of these cases, a PI controller will be sufficient [15]).

$$e(t) = r(t) - b(t) \tag{12}$$

$$f(t) = K_P e(t) + K_I \int_0^t e(t)\,dt + K_D \frac{de(t)}{dt} \tag{13}$$

$$f(t) \approx K_P e(t) + K_I \sum_{\tau=0}^{t} e(\tau) + K_D \frac{e(t) - e(t - \Delta t)}{\Delta t} \tag{14}$$

Where $e(t)$, error signal; $r(t)$, reference signal; $b(t)$, feedback signal; $f(t)$, control signal; t, time, s; K_P, proportional gain; K_I, integral gain; K_D, derivative gain; and τ, incremental time placeholder, s.

A quadrotor is a multiple-input multiple-output (MIMO) non-linear system with a high degree of coupling between the input and output variables, multiple aerodynamic effects which are difficult to measure or model precisely, and a time-varying nature as the battery is discharged and the voltage decreases [11]. Also, from the perspective of control systems, quadrotors are relatively complex to stabilise due to noise in sensors, model uncertainty, and other external disturbances [11]. Thus, it is a challenge to design a flight controller for a quadrotor and it is usually required for cascading or nested loops of multiple PID controllers directly monitoring and calculating the roll, pitch, yaw, and thrust with required operation at frequencies above 100Hz as a minimum criteria.

Without external sensors, it has also been shown that it is practically impossible to obtain stable control of a quadrotor, outside of simulations [11]. For unreliable control, the very basic sensors through an inertial measurement unit, including a gyroscope and accelerometer, are sufficient but the measurements from these sensors are usually noisy and inaccurate with large fluctuations and uncertainties. As a result, these basic sensors will not be able to completely compensate for drift from factors interfering with stability like wind and unbalanced weight distribution [2, 11]. For reliable control, it is required for a more advanced sensor which will be able to provide more accurate measurements of the state of the quadrotor, such as the mentioned sensors in the form of an external motion capture system, optical flow sensor, ultrasonic or laser range finder, GPS, or on-board camera.

3 OBJECTIVES

The primary objectives are defined as follows, with relation to the available apparatus in Section 4:

- Validate the accuracy of the pinhole camera model, with compensation for major distortions if required by the camera. (This will be done with a Raspberry Pi Camera Module 2.1).

- Compare target detection through grayscale and colour processing on low-end hardware with marginal computational effort. (A Raspberry Pi Zero W 1.1 will be used as a prime example).
- Develop and implement target detection and tracking through end-point open-loop visual servoing using the pinhole camera model and computer vision techniques for image processing on low-end hardware with marginal computational effort, where a quadrotor must remotely track the position of the target in a mirroring fashion while varying its altitude to match the camera. The tracking will be performed for three degrees of translational freedom and one degree of rotational freedom as yaw orientation. (The Raspberry Pi Camera Module 2.1 and Raspberry Pi Zero W 1.1 will be used along with OpenCV and a Bitcraze Crazyflie 2.1 quadrotor).
- Test the effectiveness of the target detection and tracking using an external motion capture system with multiple cameras, where the relative deviation between the desired position from the target and actual position of the quadrotor is measured throughout the experimentation. (The motion capture system will comprise of four Qualisys Miqus M1 cameras).

4 APPARATUS

The available apparatus consists of various parts for different systems, as broadly outlined in Figure 16. In these interactions, the quadrotor receives control setpoints from the central processing based on the visual servoing from the camera, while the motion of the target and quadrotor are tracked by the motion capture system (to be used only as validation for the effectiveness of the actual target tracking of the target by the quadrotor) and information about the current state is reported to a ground control laptop for monitoring. Moreover, the basic details of the utilised software need to be briefly explained, because it will form an intrinsic part in implementing the methodology presented in Section 5. Further resources from the manufacturers of the parts are included in Appendix A.

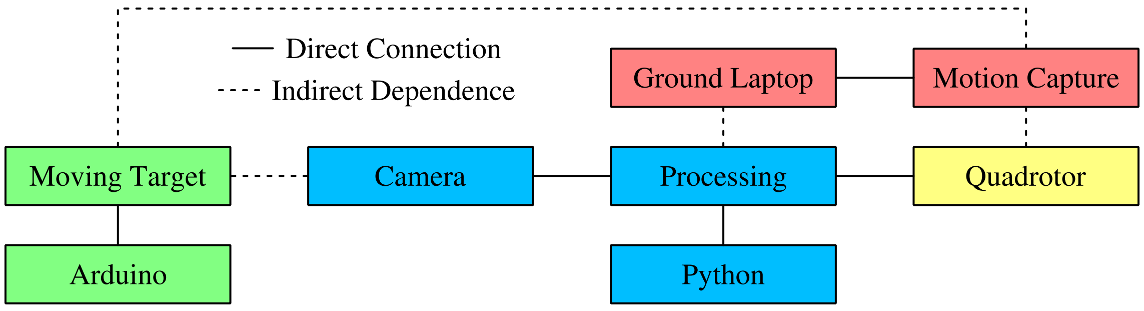


Figure 16: Outline and interactions of the apparatus from a broad overview.

4.1 QUADROTOR PARTS

For the quadrotor, the open-source Bitcraze Crazyflie 2.1 (subsequently referred to as Crazyflie) will be used. This quadrotor consists of a central control board with various on-board sensors (also acting as a lightweight frame), four rotor motors, four propellers, four stands, and battery - these components are seen in Figure 17 showing the complete assembly. Additionally, expansion decks can be connected to the control board for further on-board functionality and sensing, and a 3D printed propeller guard may be mounted with three 12mm motion capture markers - this leads to an additional payload of 7g which is below the supported additional payload of 14g. (Unfortunately, the motion capture markers could not be placed asymmetrically elsewhere as instability arose with unpredictable drift, but the chosen placement is acceptable since the yaw measurements will be accurate and are more important than the roll and pitch measurements which may be adversely affected).

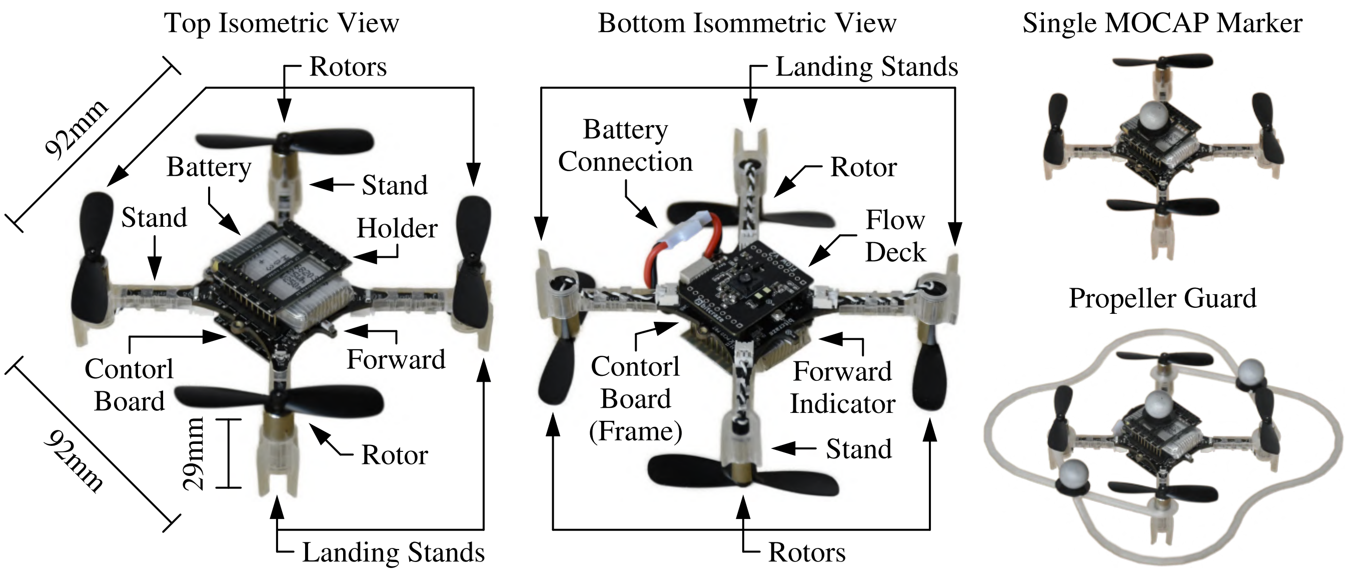


Figure 17: Photographs showing various isometric views of the assembled Bitcraze Crazyflie 2.1 quadrotor, including the Bitcraze Flow V2 expansion deck mounted at the bottom viewing the ground.

The control board, seen in Figure 18 with the basic schematics and photographs, executes the control setpoints for the motion of the quadrotor with the flashable firmware and on-board sensors. The firmware for the real-time operating system is based on the FreeRTOS kernel, which handles the scheduling of processes and calculations for the flight control - FreeRTOS is also open-source [4, 16]. This board has diagonals of 80mm and a mass of approximately 7g using a symmetric form factor and is equipped with a STM32F405 microcontroller as the main processor (Cortex-M4, 168MHz, 192kB SRAM, and 1MB flash), nRF51822 microcontroller for radio interfacing and power management (Cortex-M0, 32Mhz, 16kB SRAM, and 128kB flash), 2.4GHz ISM band radio with 20dBm or 100mW low-noise amplifier, 8kB EEPROM, Bluetooth low energy (BLE) module for interfacing through a smartphone application client, micro-USB Type-B for charging or interfacing over a wired connection, BMP388 pressure barometer, BMI088 inertial measurement unit containing a three-axis gyroscope and three-axis accelerometer, and JST-DS connector for connection to the battery [4, 6, 16, 17]. The processor and radio chips exchange data over an internal link protocol using UART as a physical interface, while the processor and sensors exchange data over I2C, PWM, and SPI - these interactions are illustrated in Figure 19 [4, 6, 16]. There is also an integrated attitude and heading reference system with an extended Kalman filter to provide a fairly reliable estimation of the current state of the quadrotor with regards to position, velocity, angular arrangement, and stability [4, 16].

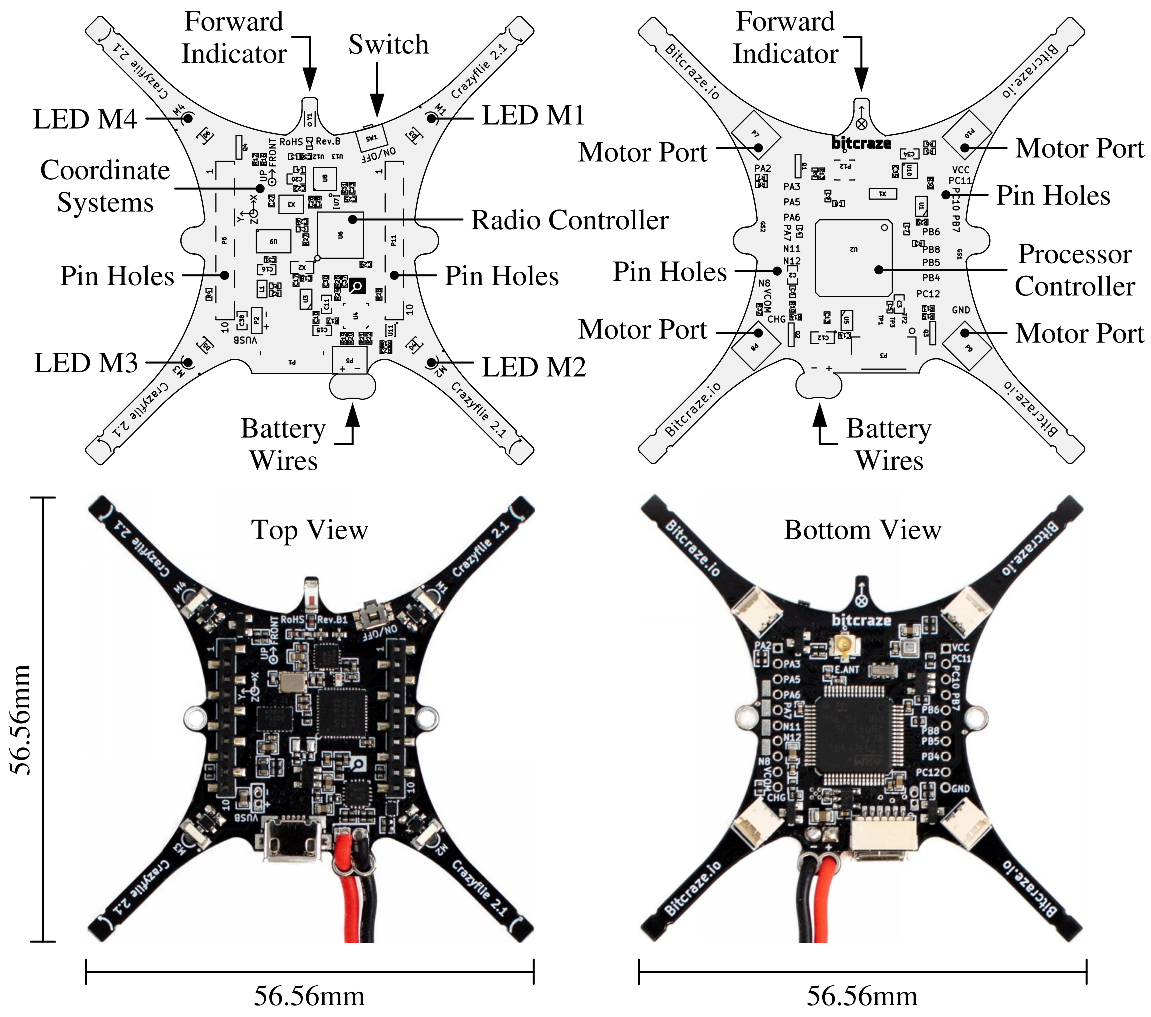


Figure 18: Schematics (top) and photographs (bottom) showing the top and bottom views of the Bitcraze Crazyflie 2.1 control board with the arrangement of the various sensors [18].

The Crazyflie is capable of a maximum speed up to 1.41m/s, although oscillations may be introduced if the Crazyflie needs to stop suddenly at a specific location. To achieve steady flight control and maintain its orientation, the control board implements internal cascading PID controllers for attitude control of position and velocity. In the cascade, there is an inner loop operating at 500Hz to control the thrust and angular velocity of roll, pitch, and yaw, and there is an outer loop operating at 250Hz to control the roll, pitch, and yaw angles for altitude and position control with stabilisation [2, 6, 16]. These controllers are conventionally described by Equation 13 and Equation 14, and have been tuned with default values based on the common construction of the Crazyflie for stable and agile flight.

The on-board sensors are sufficient to stabilise the orientation of the Crazyflie, but there will be a large uncertainty in position - in other words, it is possible for the on-board gyroscope and accelerometer to maintain the orientation and the on-board barometer could be used to estimate the altitude relative to sea level with Equation 15 based on pressure and temperature measurements, but this is extremely noisy and will be inaccurate [2, 11]. To accurately stabilise the position of the Crazyflie, it is required for additional sensors to perform measurements relative to the external environment [16].

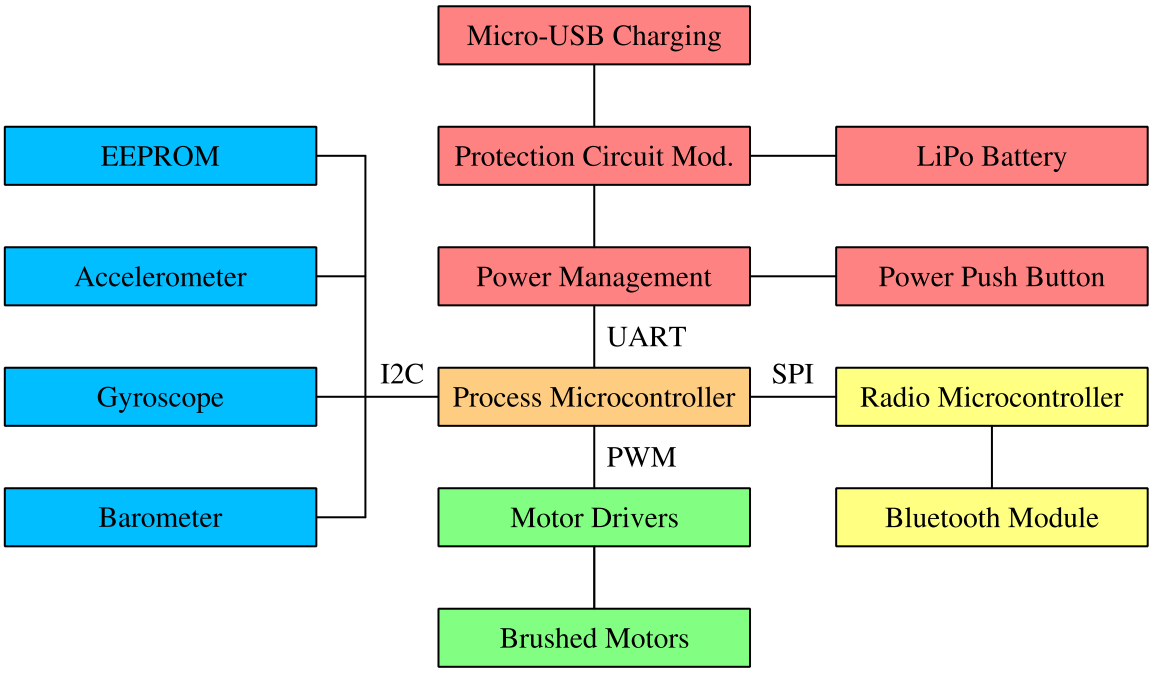


Figure 19: Interactions of the on-board microcontrollers and sensors of the Bitcraze Crazyflie 2.1.

So, the Bitcraze Flow V2 expansion deck (subsequently referred to as Flow deck) in Figure 20 will be mounted at the bottom on the control board to provide steady flight by detecting motion in any direction, which can then compensate for unwanted motion or measure desired motion [19]. The Flow deck uses a VL53L1x ToF sensor for laser-ranging and measuring the altitude to the ground up to 4m with a field of view of 25^{o} and an accuracy on the order of millimetres, and a PMW3901 sensor to employ optical flow and measure movements of the ground with a field of view of 42^{o} [19].

$$z_{sea} = \left(\frac{T}{\tau_0}\right)\left(1 - \left(\frac{p}{p_0}\right)^{-R\tau_0/(\mu g)}\right) = \left(\frac{T}{-0.0065\text{K/m}}\right)\left(1 - \left(\frac{p}{101325\text{Pa}}\right)^{0.19026}\right) \tag{15}$$

Where z_{sea}, altitude relative to sea level, m; T, temperature, K (if this cannot be measured, T_0 as the temperature at the lower troposphere limit may be used, 288.15K); τ_0, temperature gradient within the lower troposphere, -0.0065K/m; p, pressure, Pa; p_0, pressure at the lower troposphere limit,

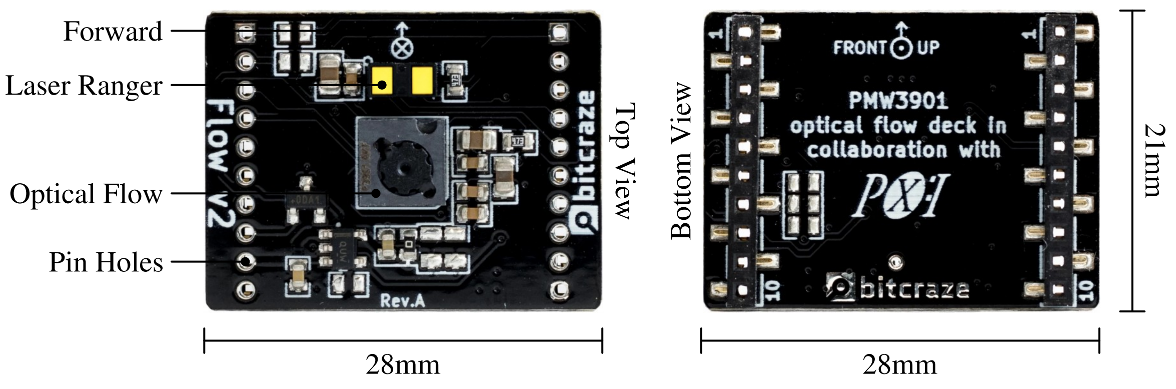


Figure 20: Photographs showing the top and bottom views of the Bitcraze Flow V2 expansion deck.

101350Pa; R, universal gas constant, 8.315kg.m^2/(s^2.mol.K); μ, air molar mass, 0.02896kg/mol; and g, gravitational acceleration, 9.807m/s^2. This is an approximation of the hypsometric formula [2].

The rotor motors, propellers, and battery are briefly mentioned, although there is only minimal information available. The rotor motors are generic brushed DC motors, which are coreless and provide fast accelerations and angular velocities up to approximately 21000rev/min [4, 16]. The propellers have a length of 45mm and are made from plastic to mitigate the possibility of damages occurring during a collision or crash. The battery is lithium-polymer (LiPo), supplies 3.7V, and has a capacity of 250mA.hr with a discharge rate of approximately 15C for a continuous flight time of approximately 7min, depending on the manoeuvres performed - there is also an internal protection circuit module to preserve the cycle life and prevent over-charging, under-charging, and shorting [4, 16, 17].

4.2 VISUAL SERVOING PARTS

To perform visual servoing and image processing, the mentioned Raspberry Pi Zero W 1.1 (subsequently referred to as Raspberry Pi) and Raspberry Pi Camera Module 2.1 (subsequently referred to as Pi Camera) will be used. The Raspberry Pi is a single-board computer with a mass of 8g, wireless LAN 802.11n, and Bluetooth 4.1, and features a 1.0GHz BCM2835 single-core CPU, 512MB of RAM, 40-pin GPIO header, micro-SD card slot for flash memory (a SanDisk 16GB micro-SD card will be used), mini-HDMI port for display, micro-USB port for power at 0.140A and 5.1V (with the Pi Camera, the required current may increase so a supply capable of 2.1A will be used), micro-USB port for peripherals, and CSI camera connector for communication with the Pi Camera [20].

The Pi Camera is designed to detect frequencies of light over the visible light range and has a mass of 3g, focal length of 3.04mm, focal ratio of 2.0, latitudinal field of view of 62.2^o, and longitudinal field of view of 48.8^o [21]. For the imaging sensor, the Pi Camera uses an 8Mpx Sony IMX219 with square pixels of 1.12µm and support for photograph resolutions up to 3280px by 2464px and video resolutions up to 1920px by 1080px with 30fr/s, 1280px by 720px at 60fr/s, or 640px by 480px at 90fr/s (or a lower resolution can be chosen, but the maximum frames rate is 90fr/s) [21]. The basic schematics and photographs of the Raspberry Pi and Pi Camera are seen in Figure 21.

Fortunately, the Pi Camera aims to have little to no noticeable fish-eye distortion. However, the Pi Camera does use a rolling shutter where frames are rapidly scanned vertically and, as a result, the frames may be somewhat susceptible to various distortion effects, which are especially pronounced when the relative speed of the target is very high and related to the magnitude of the seconds per frame captured by the camera. As mentioned, there are methods to model and compensate for distortions from rolling shutter, but these are expected to be unnecessary since it is not crucial for exceptional detail and the discrepancies are expected to be very minimal at the operating speeds.

The communication between the Raspberry Pi and Crazyflie will be facilitated by a Bitcraze Crazyradio PA (subsequently referred to as Crazyradio), which is a USB Type-A radio dongle, has a mass

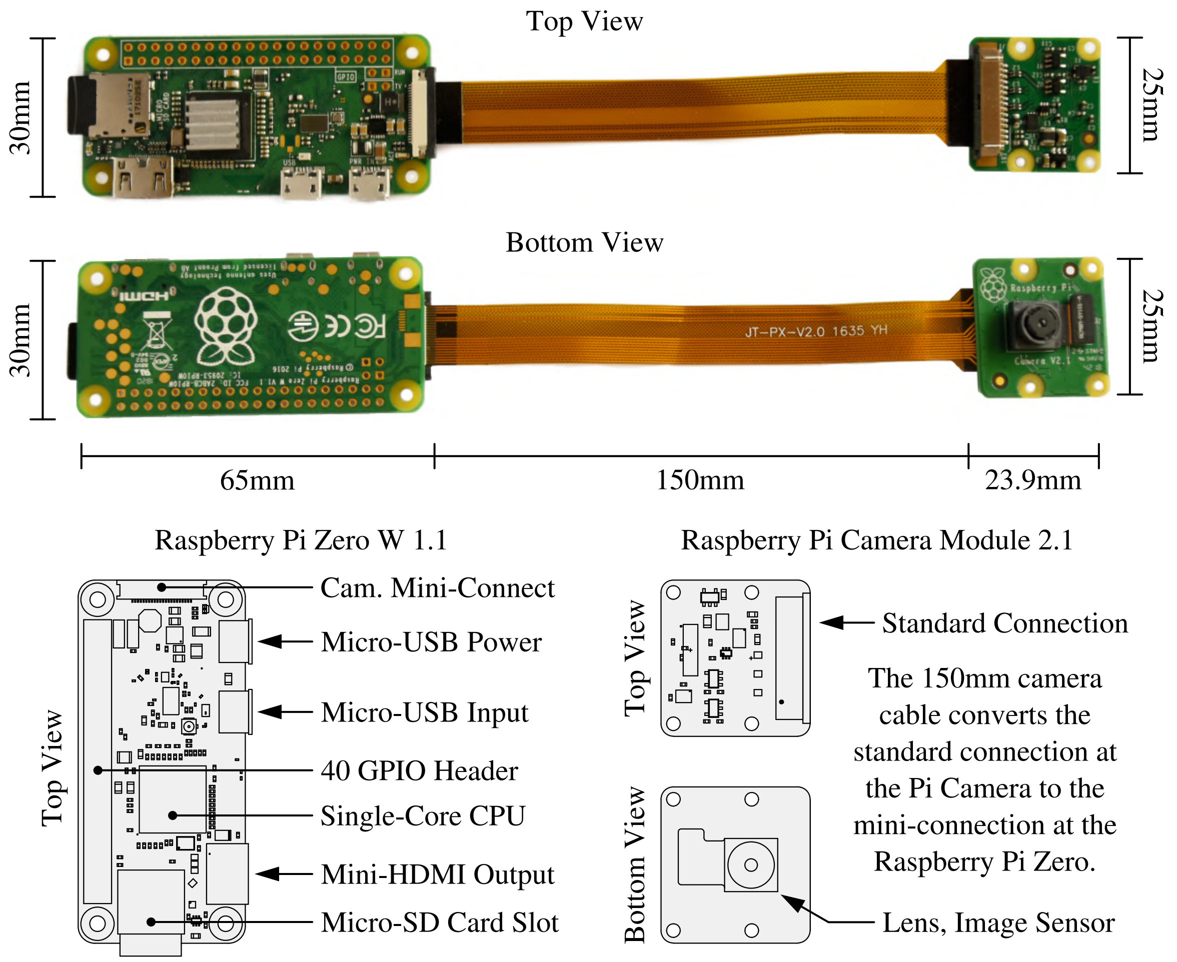


Figure 21: Schematics (bottom) and photographs (top) showing the top and bottom views of the Raspberry Pi Zero W 1.1 (left) and Raspberry Pi Camera Module 2.1 (right) [22, 23].

of 6g, and features a nRF24LU1+ radio microcontroller (16MHz, 2kB SRAM, and 32kB flash) with a 20dBm or 100mW low-noise amplifier for communication through the 2.4GHz ISM band radio at frequencies from 2.400GHz to 2.525GHz for 125 channels [16, 24]. The communication data rates can be set at 250kB/s, 1MB/s, or 2MB/s and the data sizes can be up to packets of 32B for low latency, while the ideal range can exceed 1km under normal conditions with a direct line-of-sight and without interference or obstacles [16, 24]. The minimum latency to send a packet is estimated to be about 2ms with 1ms for the USB serial communication and 1ms measured latency for the radio at 2MB/s without any retries [25]. Photographs of the Crazyradio are seen in Figure 22.

By default, the Crazyradio operates in primary transmitter mode (PTX), while the Crazyflie operates in primary receiver mode (PRX), where the Crazyradio sends data over the communication signal to the Crazyflie and the Crazyflie returns an acknowledgement packet which may also contain additional data for logging and monitoring [16]. So, for the control and communication, the high-level control setpoints will be generated on the Raspberry Pi and then wirelessly transmitted to the Crazyflie using the Crazyradio, where the firmware on the Crazyflie will interpret these setpoints and implement the corresponding low-level commands to control the rotors and overall flight.

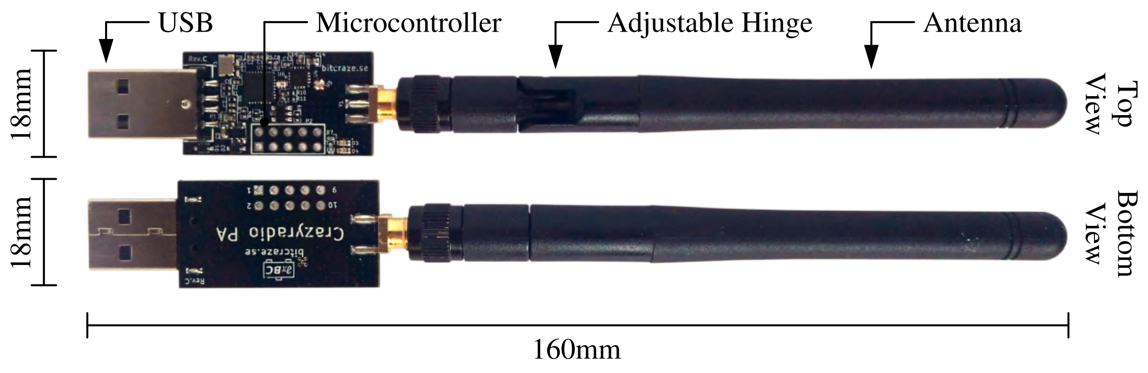


Figure 22: Photographs showing the top and bottom views of the Bitcraze Crazyradio PA.

4.3 MOVING TARGET PARTS

The moving target will be a simple two-wheeled design with a Dagu S4E EDU Controller (subsequently referred to as the Dagu Controller) as the motor driver and microcontroller which is compatible with the Arduino integrated development environment (IDE) with an Arduino Pro Mini 328 5V 16MHz bootloader [26]. This controller has an ATmega328P processor at 16MHz with 2kB of SRAM, 32kB of flash, 1kB of EEPROM, and will receive power from a battery pack with five Duracell 1.5V AA batteries with a maximum distribution of 2.5A to each motor pin [26].

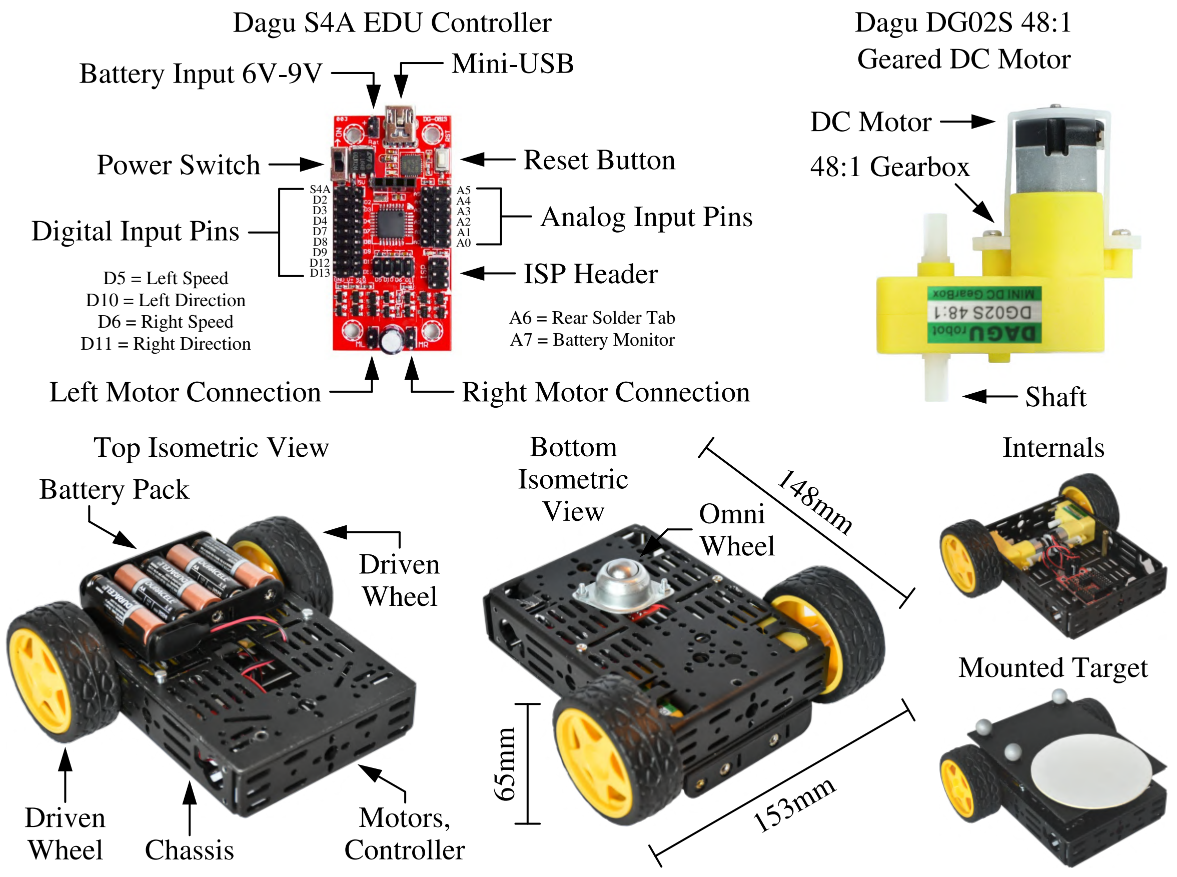


Figure 23: Photographs showing the Dagu S4A EDU Controller (top-left), Dagu DG02S 48:1 Geared DC Motor (top-right), and various isometric views of the target assembly (bottom).

Connecting to the two driven wheels of 65mm in diameter, there are two Dagu DO02S 48:1 Geared DC Motors (subsequently referred to as the Dagu Motors) requiring an input voltage between 3V and 6V with a no load current of 200mA, stall current of 1.5A at 3V or 3A at 6V, and maximum allowable torque of approximately 0.078N.m [27]. The built-in gearbox reduces the motor speed based on a ratio of 48:1, where the output angular speed can vary from 65rev/min at 3V unloaded to 190rev/min at 6V unloaded [27]. A generic omni-wheel will also be positioned centrally for stability.

For target detection with grayscale processing, a white disk of 93mm in diameter will be used as the marker, since it contrasts with the colour of the target and background for reliable detection and will not change shape as the orientation of the target changes. For target detection with colour processing, a red rectangle of 95mm by 74mm will be used as the marker, since it is unique in the environment. The Dagu Controller and Dagu Motors are shown in Figure 23 with the complete assembly, which is able to accomplish a suitable range of translational motion. The mass and quantity of each component are summarised in Table 1 for a total mass of approximately 528g.

Table 1: Quantity and mass of the components used to assemble the moving target.

	Quantity [unit]	Unit Mass [g]
Dagu S4A EDU Controller	1	14
Dagu DG02S 48:1 Geared DC Motor	2	32
Dagu 65mm Wheels With Rubber Tyres	2	40
Generic Metal Omni-Wheel	1	36
Dagu DG0-12 Metal Chassis	1	218
Duracell 1.5V AA Battery	5	25
Other Parts (Cables, Nuts, Bolts, Target, etc)	-	39
Total Assembly	-	576

To separately evaluate the yaw orientation tracking, a SG90 Micro Servo Motor will be used to set the desired orientation with a white rectangle of 140mm by 95mm as the marker, as seen in Figure 24. This servo motor can rotate between 0^o and 180^o with a torque up to 0.245N.m and mass of 14.7g, where control is issued through a PWM signal and a voltage between 4.8V and 6V is supplied.

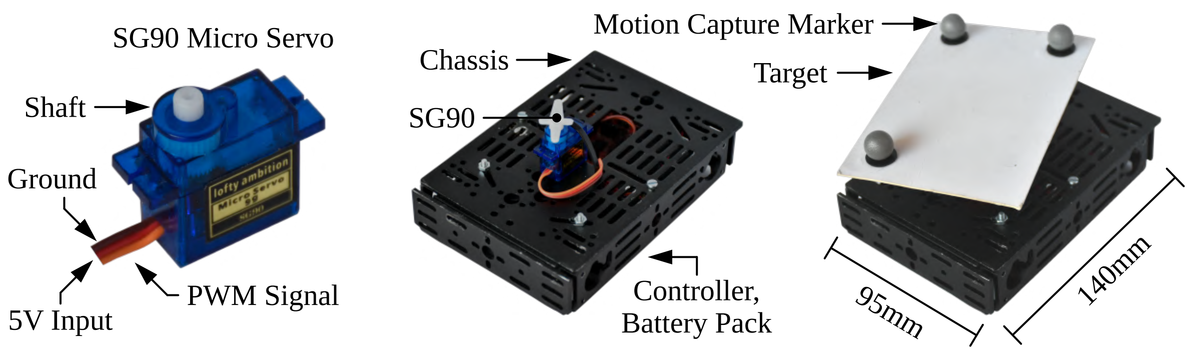


Figure 24: Photographs showing the SG90 Micro Servo motor and yawing target assembly.

4.4 MOTION CAPTURE FACILITIES

The motion capture system utilises four Qualisys Miqus M1 cameras. These cameras feature a 1Mpx image sensor with a resolution of 1216px by 800px, high-speed frame rate up to 250fr/s, latitudinal field of view of 58^{o}, longitudinal field of view of 40^{o}, maximum range of 10m using 16mm markers, and low latency for real-time applications with a camera latency of 2.9ms and system latency of 5ms (there is also an alternate mode for a latitudinal field of view of 41^{o} and longitudinal field of view of 27^{o}) [28]. The body of the camera has dimensions of 140mm by 84mm by 84mm with 102 NIR LEDs and infra-red strobe at 850nm [28]. Overall, the system allows for an accuracy anticipated to be on the order of millimetres. The setup in the motion capture facilities is shown in Figure 25, where the operating area for the quadrotor to use is approximately 2420mm by 2420mm with a height up to about 1500mm and black mats have been placed on the floor to cover this area.

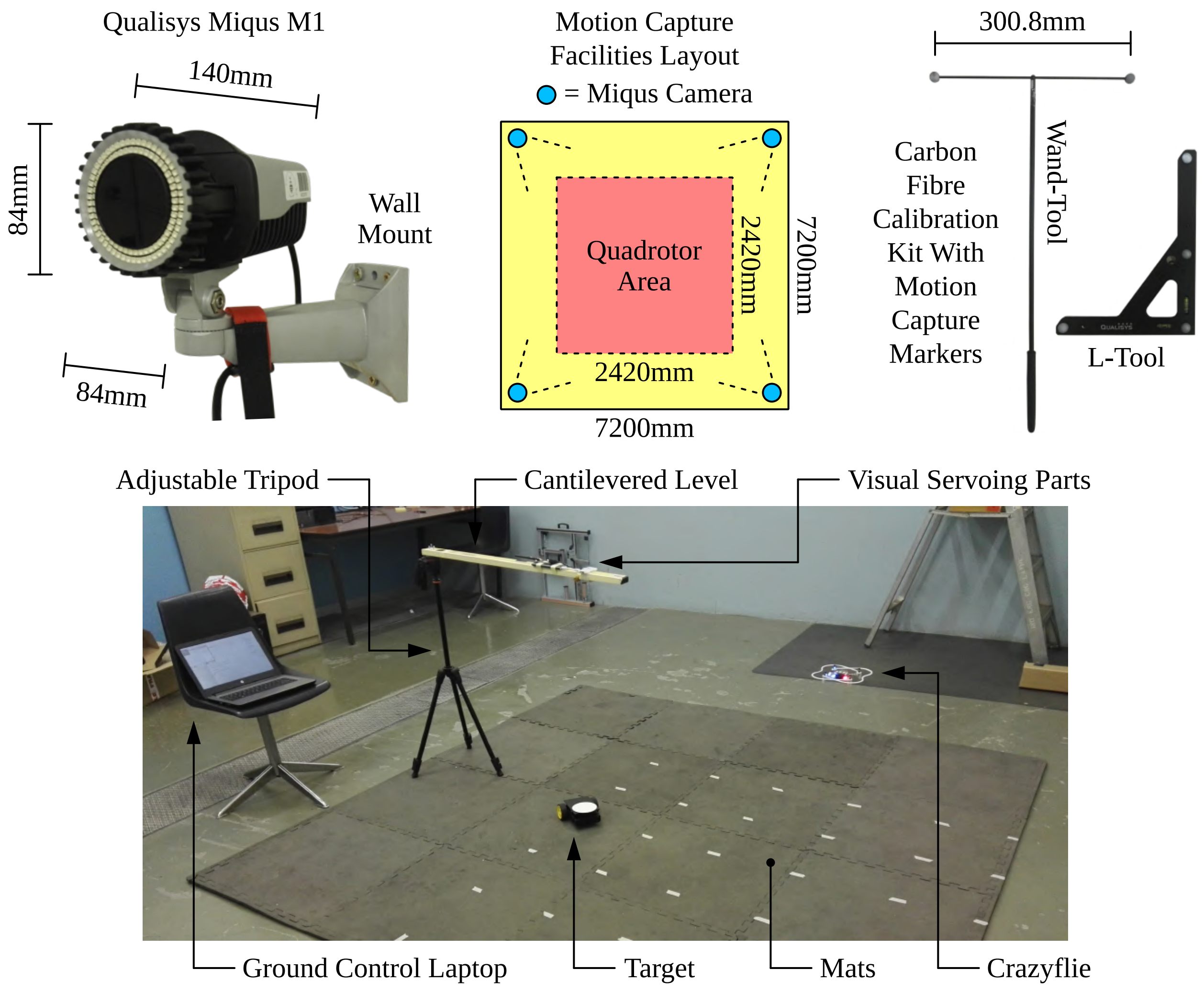


Figure 25: Photographs showing a Qualisys Miqus M1 (top-left), basic layout of the motion capture facilities (top-middle), carbon fibre calibration kit for the initial calibration of the motion capture volume (top-right), and arrangement of the apparatus within the motion capture facilities (bottom).

4.5 GROUND CONTROL LAPTOP

A laptop will be used to act as a ground control station for remotely accessing the Raspberry Pi, monitoring the target detection and tracking, and running the Qualisys Track Manager (QTM) software for the motion capture system. To facilitate connection to the Raspbian operating system on the Raspberry Pi, a local wireless LAN network will be created by the ground control laptop. In this case, an HP Notebook 14-an013nr will be used with an AMD E3-7110 processor incorporating four threads and four cores running at 1.8GHz, but any modern laptop would likely be a sufficient substitute.

4.6 SOFTWARE

The software to be used for visual servoing, controlling the quadrotor, and controlling the target is open-source with free availability and primarily includes Python, OpenCV, Raspbian, and Arduino. The logos associated with these software packages are illustrated in Figure 26 for visual distinction.

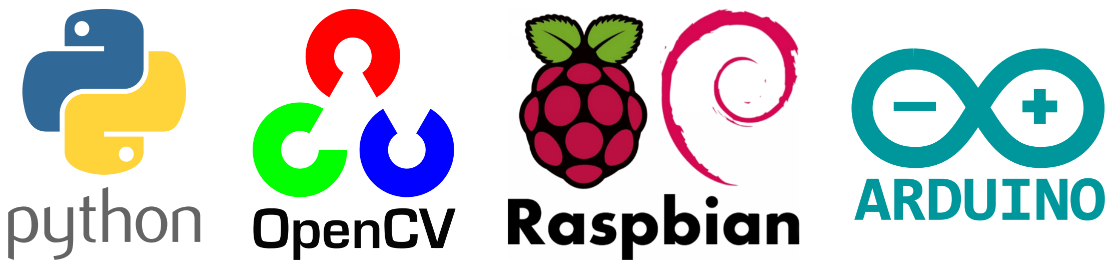


Figure 26: Open-source software used for visual servoing, quadrotor control, and target control.

4.6.1 CRAZYFLIE

The low-level firmware of the Crazyflie is written in C and C++, but there is a Crazyflie Python application programming interface (API) library `cflib` with high-level bindings which can send setpoints. With access through Python 3.7.3, the following classes and functions will be used:

- `Crazyflie`: Contains the intrinsic commands and callbacks used by the higher-level functions in other classes, where the same design as in the firmware is used for a one-to-one mapping.
- `SyncCrazyflie`: Wrapper around the `Crazyflie` class to handle its asynchronous nature and convert it for use as blocking functions in scripts performing tasks as sequences of events.
- `SyncLogger`: Provide synchronous access to log current variables. The variables which can be logged include estimated position, velocity, acceleration, angular orientation, and other information associated with the current state of the quadrotor as measured from the on-board sensors.
- `LogConfig`: Create a configuration in which to log variables.
  - `add_variable`: Add a variable to be logged with its group and name.
  - `start`: Begin the logging of the added variables.
  - `stop`: End the logging of the added variables.

- `Commander`: Send control setpoints to the Crazyflie. The setpoints need to be sent as often as possible, where the Crazyflie has a continuous backend or watchdog timer which will reset the roll and pitch after 0.5s and stop the motors after 1s if no setpoint is received. A minimum limit at which setpoints can be generated and sent is recommended at 10Hz.
  - `send_setpoint`: Send a setpoint in terms of roll, pitch, and yaw angles and thrust.
  - `send_velocity_world_setpoint`: Send a setpoint in terms of the desired absolute velocities in the x, y, and z directions with a specific yaw rate.
  - `send_hover_setpoint`: Send a setpoint in terms of desired absolute velocities in the x and y directions with a specific yaw rate, while a constant altitude is maintained.
  - `send_position_setpoint`: Send a setpoint in terms of the desired absolute position with x, y, and z coordinates and a specific yaw angle. This actually uses a velocity setpoint over a time period since there is no direct sense of absolute position, so the error in the estimated position may accumulate over time. The Crazyflie attempts to execute this as fast as possible with a maximum velocity up to 1m/s as limited by default - this can easily be changed by setting the `posCtlPid.xyVelMax` and `posCtlPid.zVelMax` parameters.
- `Extpos`: Send the current position of the Crazyflie measured using external sensors, which will be directly forwarded to the position estimator of the Crazyflie.
  - `send_extpos`: Send the current position of the Crazyflie in terms of x, y, and z coordinates.
- `MotionCommander`: Send control setpoints to the Crazyflie. This class is more orientated towards blocking functions, where a command is executed and the script waits until the motion is complete before continuing, rather than just sending a setpoint and continuing immediately. So, it is not an applicable option for real-time operations requiring variable adjustments.

If desired, further information can be found from the Bitcraze GitHub repository: `https://github.com/bitcraze/crazyflie-lib-python`. A graphical client is also available which allows for flashing firmware and interfacing with the Crazyflie using a remote controller for manual control.

4.6.2 OPENCV

OpenCV is a library of programming functions aimed at real-time computer vision. The Python implementation of OpenCV will be used through the OpenCV Python 3.2 library, which is an API with bindings for interfacing with the actual functions written in C++ (the complete version of OpenCV is actually written in C++, but this will be difficult to integrate with the Crazyflie Python API library and the extra functions are not really required for appropriate target detection and tracking). The library includes various functions for image processing as basically summarised:

- `line`, `rectangle`, `circle`, `polylines`, and `putText`: Draw shapes or text on the image.
- `cvtColor`: Convert between two colour modes. This will typically be used to convert from the default blue, green, and red (BGR) mode to grayscale with the argument `COLOR_BGR2GRAY` or to

a hue, lightness, and saturation (HLS) mode with the argument `COLOR_BGR2HLS`. (It should be emphasised that BGR is the default colour mode - not the common RGB mode).

- `blur`, `GaussianBlur`, `medianBlur`, or `bilatitudinalFilter`: Apply a blur distortion to the image. This will typically be used to provide smoothing and remove noise.
- `threshold`: Apply a threshold or mask to the image. This is typically used to generate a binary image using the argument `THRESHOLD_BINARY` or `THRESHOLD_BINARY_INV`.
- `adaptiveThreshold`: Apply an adaptive threshold or mask. This is typically used to consider illumination factors from the local neighbourhood of a pixel before applying the threshold.
- `inRange`: Apply a threshold or mask to the image based on the colours within minimum and maximum bounds (usually with HLS or HVS). This is typically used to generate a binary image.
- `erode`, `dilation`, or `morphologyEx`: Apply a morphology transformation. This is typically used to further reduce noise, where the latter will either apply an erosion transformation (remove white noise) followed by a dilation transformation (restore the original areas without noise) with `MORPH_OPEN` or apply a dilation transformation (connect white regions) followed by an erosion transformation (restore the original areas without disconnections) with `MORPH_CLOSE`.
- `Laplacian`, `Sobel`, or `Scharr`: Find the gradient or derivative intensities in a direction. This is typically used to highlight edges using a high-pass filter based on feature changes.
- `Canny`: Perform edge detection. This is an existing algorithm with multiple stages including noise reduction with a Gaussian filter, gradient intensity finding with the Sobel method in both latitudinal and longitudinal directions, non-maximum suppression with the removal of pixels which are not of the maximum value in the local neighbourhood in the gradient directions, and hysteresis thresholding to classify pixels as edges to be kept or non-edges to be discarded. Since a self-derived algorithm is required, this function will not be used, but it provides an initial guide.
- `findContours`: Find the contours (as mentioned, a contour is a continuous curve joining the points which have the same colour or intensity along a boundary). This is typically used for basic shape analysis and target detection with `RETR_TREE` to retrieve all contours and reconstruct a full hierarchy of nested contours and `CHAIN_APROX_SIMPLE` to only store the end points of straight lines for decreased processing. (The contours can be drawn with `findContours`).
- `boundingRect`: Find the top-left coordinate for a bounding rectangle and the width and height of this bounding rectangle applied to a contour. This is typically used for monitoring by drawing the found bounding rectangle around a contour. (There is no consideration for rotation).
- `minAreaRect`: Find the centroid, dimensions (height and width), and orientation between -90^0 and 0^o for a contour. This is typically used when the orientation of a contour is needed.
- `matchTemplate`: Search for and find the location of a provided template image in a larger image. This is typically used when an almost identical match is possible, where there is no scaling, colour, or distortion changes of the template image within the larger image.

By default, OpenCV uses the connected camera to capture an image at a resolution of 640px by 480px, but this can be changed along with other default parameters using `set`, such as the frame rate, codec, file format, brightness, contrast, saturation, hue, gain, exposure, and white balance - it should be noted that the camera must inherently support changing these parameters. There are also more advanced functions available which allow for camera calibration, optical flow, augmented reality, epipolar geometry, and various machine learning techniques, but these are mostly only applicable for specialised applications where greater computational effort is available [13].

(It should be noted that the OpenCV Python API library requires Numpy for backend processing, which is an open-source Python library for performing advanced calculations and matrix operations).

4.6.3 RASPBIAN

Raspbian Buster July 2019 is an operating system based on Debian 10 with the Linux Kernel 4.19. The operating system has been optimised for best performance on the Raspberry Pi and will be used to run the control script for visual servoing while issuing setpoints to control the Crazyflie through the installation of Python 3.7.3 and the required libraries. It also allows for interfacing with the ground control laptop using the wireless LAN 802.11n of the Raspberry Pi and local network created by the ground control laptop, where a remote desktop protocol (RDP) or secure shell (SSH) session is used for remote login through a graphical user interface (GUI) or the command line respectively.

4.6.4 ARDUINO

The Arduino IDE offers a means to program the target using C++ with special rules of code structuring, and it compiles the script when uploading to the controller (any programming language with a compiler to produce binary machine code could actually be used, but it will be most convenient to use the Arduino IDE). Since the Dagu Controller is compatible with the Arduino IDE, the motors can then be controlled using PWM signals to produce voltages between 0V and 5V corresponding to duty cycles linearly represented by values between 0 (0%) and 255 (100%). The following Arduino commands will primarily be used to control the motors and corresponding motion of the target:

- `pinMode`: Configure a pin to behave as either an `INPUT` (high-impedance state) or `OUTPUT` (low-impedance state). This is used to initially set the direction pin of each motor as an output.
- `digitalWrite`: Write a `HIGH` (5V) or `LOW` (0V) value to a digital pin. This is applied on the direction pin for the respective motor and used to set the direction in which the motor must turn.
- `analogWrite`: Write an analog value to a pin for a duty cycle represented between 0 (0%) and 255 (100%). This is applied to the signal pin for the respective motor and used to set the speed in terms of a PWM value at which the motor must turn based on a voltage between 0V and 5V.
- `servo.write`: Instruct the servo motor to rotate to the passed angle between 0^o and 180^o (`servo` is the name assigned to the pin on which the servo motor is attached).

5 METHODOLOGY

After preparatory development with the apparatus to gain an improved understanding, the proposed methodology can be divided into parts including camera compensation and calibration; initial Crazyflie considerations; target detection through processing the images from the camera; target tracking through issuing command setpoints to autonomously move the Crazyflie based on the location of the target; overall communication between the Raspberry Pi, Crazyflie, ground control laptop, and motion capture system; and motions of the target with different paths. Importantly, it is also required to monitor and store the position of the target and Crazyflie in real-time during execution using the motion capture system which will be examined to validate success or failure. A complete risk assessment for the operators, supervisors, and nearby bystanders has also been included in Appendix D.

5.1 INITIAL CAMERA CONSIDERATIONS

Firstly, it is necessary to confirm there is no significant fish-eye distortion, blurring, or effects from rolling shutter. This will be done by taking a series of sample images of a chessboard template and measuring whether the straight lines in the template appear straight in the images, where the measurement will be conducted by digitally overlaying straight lines over the captured image to observe any deviations. This also serves as a means to ensure the camera is not faulty.

Before the Raspberry Pi and Pi Camera can be used for target detection and tracking, it is necessary to evaluate the real-time performance for the image processing. This will be assessed by recording a short clip of approximately 15s to 30s with resolutions of 160px by 120px, 320px by 240px, and 640px by 480px to determine which resolutions provide a sufficiently high frames rate with less than 100ms between frames, so at least 10 setpoints can be generated per second. For a realistic result, practical cases will be considered to represent realistic use where there is no image processing, a minimal amount of image processing with only primary information for basic monitoring, and a high level of image processing with the additional extraction of secondary information for complete monitoring. Both grayscale and colour processing will be compared to evaluate the most effective and efficient method. For an accurate result, the average of three tests for each case with each resolution will be compared. The developed control script is available in Appendix B, where only OpenCV was employed which is valid since the tests are being compared directly against each other.

Warranting a lack of distortion and successful real-time performance, the pinhole camera model needs to be validated for the purpose of determining the desired position for the Crazyflie. So, the camera needs to be calibrated which will be based on a relationship for the current altitude in terms of the focal length of the camera, actual target area, and virtual target area, as found through manipulating the pinhole camera model relationships in Equation 16 for the disk target. Because the focal length of the camera and actual target area will be constant, this reveals an experimental correlation coefficient unique to the camera which can then be used in calculations with only the virtual target area.

Thus, a calibration will be performed where the camera is moved between an altitude of approximately 300mm to 800mm in increments of 20mm while the virtual target area is detected and recorded, as demonstrated in Figure 27. At each altitude increment, the target will be positioned in the field of view at the centre, top-left, top-right, bottom-left, and bottom-right regions to obtain an average value of the virtual target area and a gauge of the likely error and uncertainty in the measurements. This allows for a trend to be extracted from the recorded data, which can also be used to estimate the altitude of the camera. Considering the rectangle target, the same relationship can be derived from the pinhole camera model, as demonstrated in Equation 17, which presents the option for confirmation with reiteration of the calibration. The proportional relationships are also simplified in Equation 18.

$$z = f\frac{x}{u} = f\frac{y}{v} \longrightarrow z = f\sqrt{\frac{\pi x^2/4}{\pi u^2/4}} = f\sqrt{\frac{\pi y^2/4}{\pi v^2/4}} = f\sqrt{\frac{A_{xy}}{A_{uv}}} \equiv kA_{uv}^{-0.5} \text{ where } k = f\sqrt{A_{xy}} \tag{16}$$

$$z = f\frac{x}{u} = f\frac{y}{v} \longrightarrow z = f\sqrt{\frac{xy}{uv}} = f\sqrt{\frac{A_{xy}}{A_{uv}}} \equiv kA_{uv}^{-0.5} \text{ where } k = f\sqrt{A_{xy}} \tag{17}$$

$$\therefore\ z \propto \frac{1}{u} \text{ and } z \propto \frac{1}{v} \text{ and } z^2 \propto \frac{1}{A_{uv}} \text{ or } z \propto \frac{1}{\sqrt{A_{uv}}} \tag{18}$$

Where z, altitude, m; f, focal length, px; x, actual latitude distance, m; u, virtual image latitude distance, px; y, actual longitude distance, m; v, virtual image longitude distance, px; A_{xy}, actual area, m^2; A_{uv}, virtual image area, px^2; and k, camera correlation coefficient, m.px.

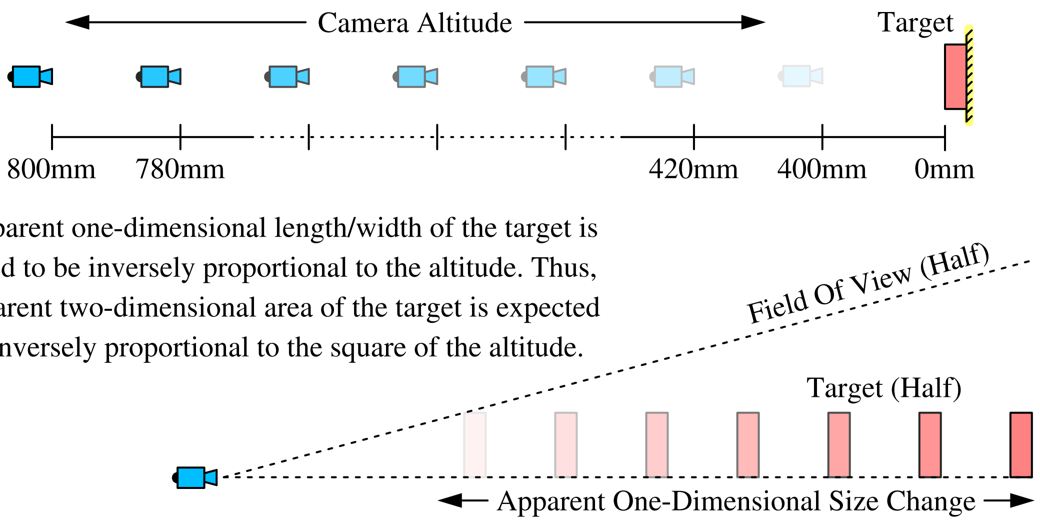

Camera Altitude
Target
800mm
780mm
420mm
400mm
0mm
The apparent one-dimensional length/width of the target is expected to be inversely proportional to the altitude. Thus, the apparent two-dimensional area of the target is expected to be inversely proportional to the square of the altitude.
Field Of View (Half)
Target (Half)
Apparent One-Dimensional Size Change

Figure 27: Method of calibrating the camera and determining the focal length.

5.2 INITIAL QUADROTOR CONSIDERATIONS

By default, the Crazyflie is set to operate in cross-mode which is acceptable given that there are no obvious reasons to change to plus-mode. To avoid effects from flying too near the ground, the Crazyflie should maintain an altitude above 300mm while also remaining within the range of the motion capture facilities. The natural behaviour of the Crazyflie while attempting to hover also needs to be analysed with observations into the drift experienced without any trimming being performed, so this can then be corrected with appropriate trimming to establish balanced behaviour if necessary.

5.3 TARGET DETECTION

A direct form of position-based visual servoing will be used, where both the current altitude of the camera and position of the target will be estimated through a reconstruction of the three-dimensional locations. Unfortunately, this technique will slightly suffer from the camera calibration errors.

So, the necessary information for visual servoing through target detection and tracking can be found from the Pi Camera viewing the target. The target detection can be performed through either grayscale or colour processing, where various filters are applied to the captured image to isolate the target - as mentioned, the most effective and efficient method will be initial evaluated and then used in the final tests for the best real-time performance. After considering and testing with the available techniques in OpenCV through trials, the following developed algorithm will be performed on each captured frame with the aim of reliable target detection while maintaining minimal computational effort:

A. Fundamental concept for grayscale processing (either A or B is performed):

1. Once the BGR original colour image has been captured, it will be converted to a grayscale image to reduce the values for each pixel and computational effort required during processing.
2. Using Equation 19, a binary threshold will be applied to the grayscale image with a threshold value, where a pixel above this value will become white and a pixel below this value will become black. The value will be tuned to remove the background while maintaining the target as white.

$$p(g) = \begin{cases} g_{new} & \text{if } g > g_{thr} \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Where p, pixel; g, grayscale value; g_{new}, new grayscale value, 255; g_{thr}, grayscale threshold value.

B. Fundamental concept for colour processing (either A or B is performed):

1. Once the original BGR colour image has been captured, it will be converted to a HLS colour image to provide easier isolation of the range for the distinct colour of the target.
2. Using Equation 20, a binary threshold will be applied to the HLS colour image with minimum and maximum bounds of HLS values to create the isolation range, where a pixel within this range will become white and a pixel outside of this range will become black. The bounds act as a tolerance since there may be distortions in the captured colours due to environment lighting, and the values will be tuned to remove the background while maintaining the target as white.

$$p(h,l,s) = \begin{cases} g_{new} & \text{if } h_{min} < h < h_{max},\ l_{min} < l < l_{max},\ s_{min} < s < s_{max} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

Where p, pixel; h, hue value; l, lightness value; s, saturation value; g_{new}, new grayscale value, 255;

h_{min}, minimum hue value; h_{max}, maximum hue value; l_{min}, minimum lightness value; l_{max}, maximum lightness value; and s_{min}, minimum saturation value; s_{max}, maximum saturation value.

C. Common target detection strategy (performed after either A or B is completed):

3. The noise in the threshold image will be reduced to a satisfactory level using a morphology transformation with a kernel size tuned for an open transformation with an erosion transformation followed by a dilation transformation. This will ensure the target is the largest white region in the image (unless there is a remarkably rare exception, but this will be accounted for and mitigated by checking whether the corresponding pixel area is within the expected range).
4. The largest contour in the image will then be found based on the maximum pixel area in terms of pixels. Assuming this contour is the target as is practically assured, the location of the centroid of the target can be found in terms of pixels to complete the target detection (a box can also be drawn around the contour to identify the target for monitoring confirmation).
5. If yaw orientation tracking is also desired, the angle of the target relative to the forward direction needs to be distinguished by fitting a rectangle with the minimum area in which the contour of the target can be enclosed. The respective angle of this rectangle can then be found from the corners of the rectangle, where this angle will provide the orientation of the target.

Implementing a blur before applying the threshold was initially considered, however preparatory tests indicated that this was unnecessary as long as a morphology transformation was performed, as is preferred for better noise reduction - in other words, a blur offered no perceivable advantages and will only result in greater computational effort. It was also possible to apply an adaptive threshold, but this is expected to be unnecessary given the stark contrast between the background and target.

5.4 TARGET TRACKING

Since the features of the white disk on the target will be constant for every orientation, the current altitude of the camera can be estimated based on the area of the target in terms of pixels in the captured frame, due to the target remaining at a constant altitude on the ground and relative altitude changes coming from the altitude of the camera. Thus, once the area of the target is known from the target detection, the desired altitude for the Crazyflie can be related with the found relationship from the camera calibration in the form of Equation 16 (instead of being used for calibration with a known altitude, the relationship will be known and the associated altitude will be calculated).

Using the pinhole camera model, the actual position of the target (which is the desired position of the Crazyflie) can also be estimated based on the location of the centroid of the target in terms of pixels relative to a chosen coordinate system within the captured frame. Thus, once the centroid of the target is known from the target detection, the desired position of the Crazyflie can be estimated based on Equation 21 for the latitude coordinate and Equation 22 for the longitude coordinate (the previously found altitude and experimentally calibrated focal length will actually be used instead of the ratio of the areas from the expanded theoretical model). These methods are illustrated in Figure 28.

$$x = u\,\frac{z}{f} = u\,\frac{k}{f\sqrt{A_{uv}}} = u\sqrt{\frac{A_{xy}}{A_{uv}}} \tag{21}$$

$$y = v\,\frac{z}{f} = v\,\frac{k}{f\sqrt{A_{uv}}} = v\sqrt{\frac{A_{xy}}{A_{uv}}} \tag{22}$$

Where x, actual latitude coordinate, m; u, virtual image latitude coordinate, px; z, altitude, m; f, focal length, px; y, actual longitude coordinate, m; and v, virtual image longitude coordinate, px.

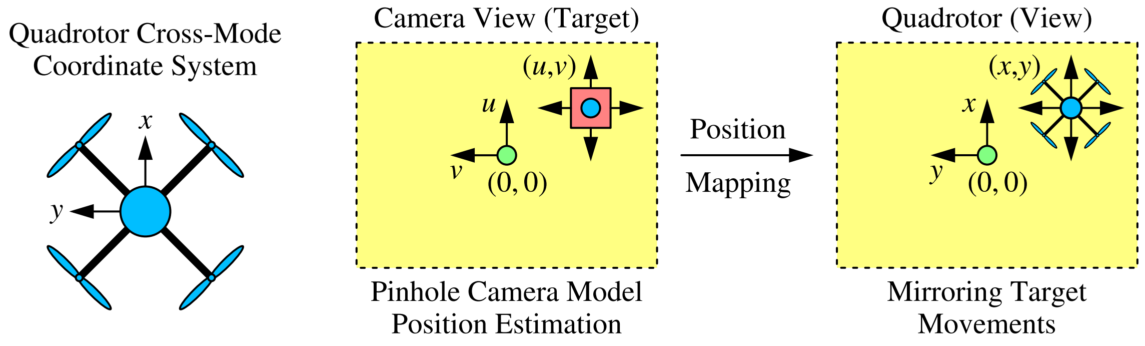

Quadrotor Cross-Mode Coordinate System
x
y
Camera View (Target)
(u,v)
u
v
(0, 0)
Pinhole Camera Model Position Estimation
Position Mapping
Quadrotor (View)
(x,y)
x
y
(0, 0)
Mirroring Target Movements

Figure 28: Tracking of the target with the corresponding movement of the Crazyflie, where the desired altitude will also be varied by manually changing the relative altitude of the camera.

To achieve successful results, the Crazyflie needs to know its starting location. So, for a coordinate system with the origin chosen to be at the centre of the image (coordinates of $(u, v) = (0, 0)$ are assigned to the centre pixel, which corresponds to position coordinates of $(x, y) = (0, 0)$), it will be required a conversion of the initial coordinate system through manipulation with Equation 23 and Equation 24, such that the coordinate system is moved and rotated from the default location in the top-left corner. This coordinate system conversion is detailed in Figure 29.

$$u = -v_i + \frac{r_W}{2} \tag{23}$$

$$v = -u_i + \frac{r_H}{2} \tag{24}$$

Where u, new latitude coordinate, px; u_i, initial latitude coordinate, px; r_W, latitude resolution, px; v, new longitude coordinate, px; v_i, initial longitude coordinate, px; and r_H, longitude resolution, px.

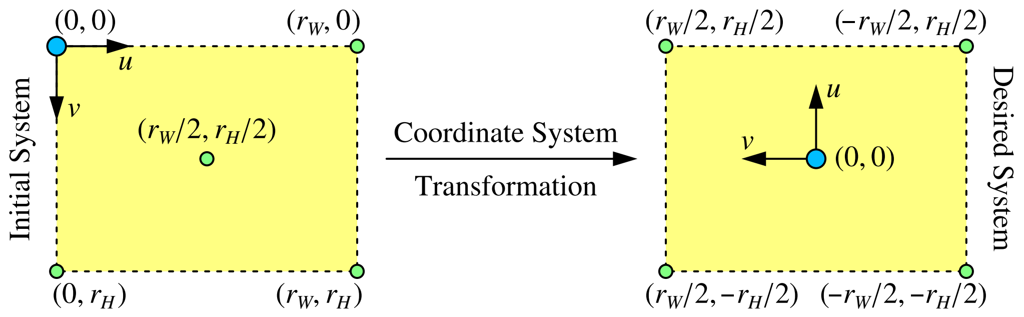
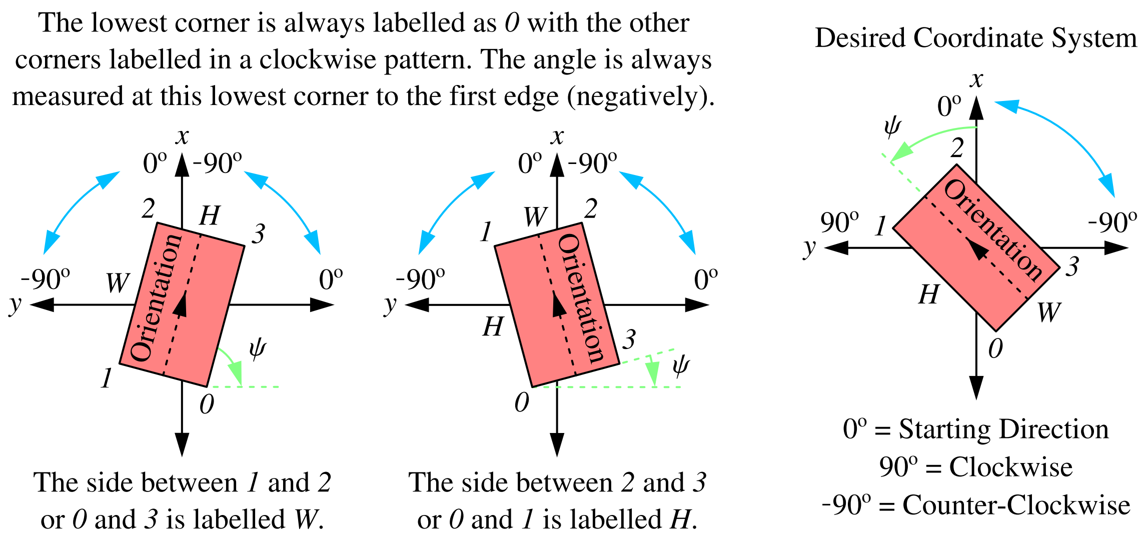


Figure 29: Coordinate system transformation to convert the initial system into the desired system.

A coordinate system also needs to be established for the yaw orientation tracking. As seen in Figure 30, the direct output from OpenCV is uncertain and it is not possible to establish the orientation since the target could be aligned in either of the two forward quadrants. So, relative to the rectangle target, it is devised that the returned output will be the angle measured between 0° and -90° from a horizontal projected at the lowest corner to the first edge. The orientation relative to the two forward quadrants can then be evaluated based on the labelling of the corners and which of the edges are labelled as the width and height. Thus, the yaw orientation can be found between -90° and 90° using the conditions of Equation 25 where the forward direction has a yaw orientation of 0°.

$$\psi = \begin{cases} -\psi_i & \text{if } W < H \\ -\psi_i - 90^{\circ} & \text{otherwise} \end{cases} \tag{25}$$

Where ψ, yaw orientation, °; ψ_i, initial yaw orientation, °; W, width, px; and H, height, px.



The lowest corner is always labelled as *0* with the other corners labelled in a clockwise pattern. The angle is always measured at this lowest corner to the first edge (negatively).
x
0°
-90°
y
2
H
3
W
1
0
Orientation
ψ
The side between *1* and *2* or *0* and *3* is labelled *W*.
The side between *2* and *3* or *0* and *1* is labelled *H*.
Desired Coordinate System
90°
0° = Starting Direction
90° = Clockwise
-90° = Counter-Clockwise

Figure 30: Explanation of the initial yaw orientation (left) and desired coordinate system (right). (It should be emphasised that this is only the coordinate system for the Crazyflie, and the target will use a conventional coordinate system between 0° and 180°as presented in Figure 32).

Subsequently, the sequence in which setpoints will be generated for target tracking is summarised:

1. Following the target detection, the found centroid of the target contour will be transformed into the desired coordinate system with the centre of the camera view as the origin.
2. The pinhole camera model will be used to convert the found area and transformed centroid of the target contour into position coordinates for the desired altitude and position of the Crazyflie.
3. The desired altitude and position of the Crazyflie will then be sent to the Crazyflie as a setpoint for the Crazyflie to move to this location, relative to its initial location as the origin.
4. This process repeats with each frame captured by the camera for as long as the target is moving.

5.5 CONTROL AND COMMUNICATION

To clearly describe the overall processes of target tracking, the flow diagram in Figure 31 is developed. In this diagram, the general idea for performing target tracking and open-loop control of the Crazyflie is depicted which is fully implemented in the final control script presented in Appendix B.

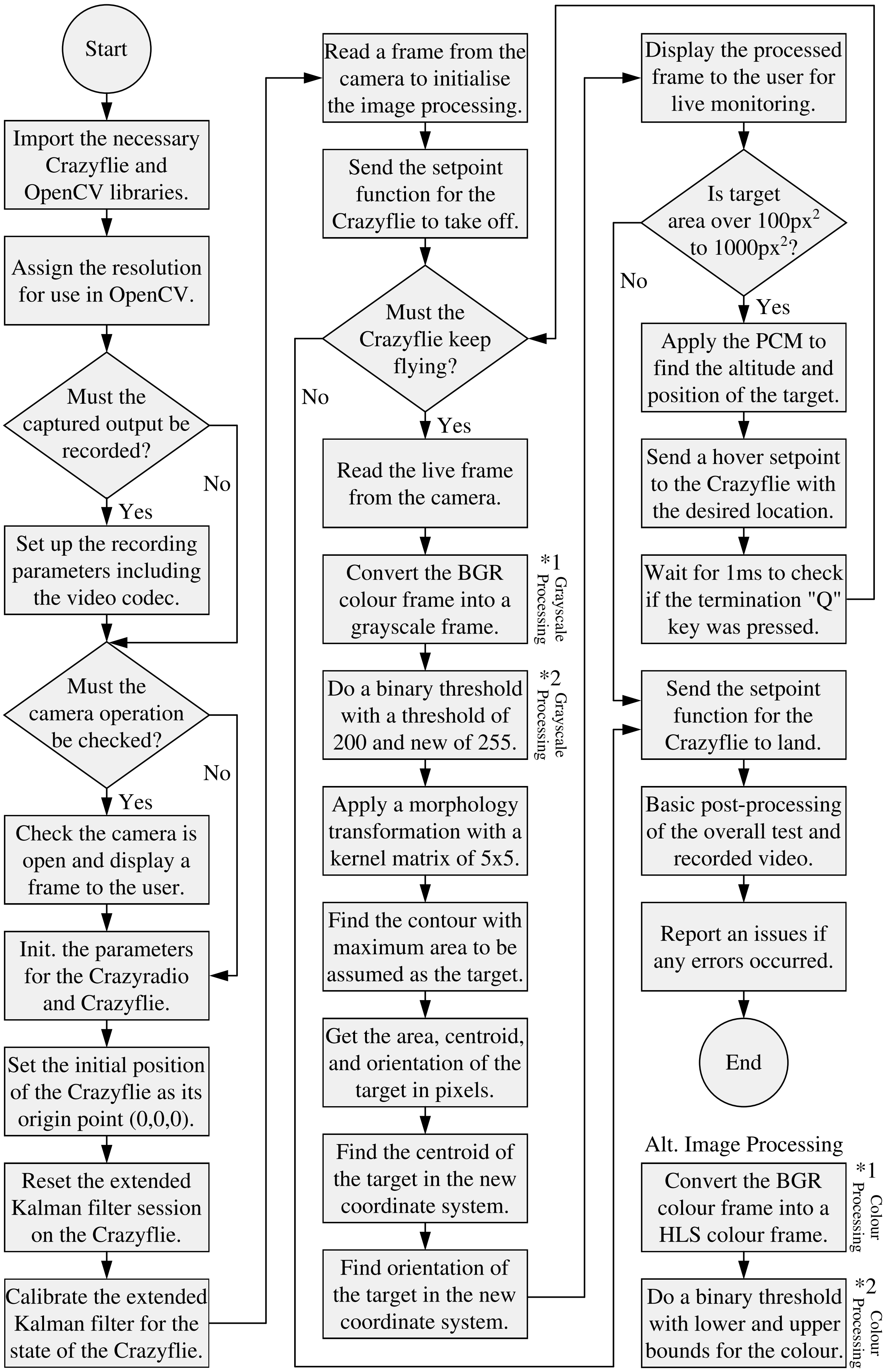

Start
Import the necessary Crazyflie and OpenCV libraries.
Assign the resolution for use in OpenCV.
Must the captured output be recorded?
No
Yes
Set up the recording parameters including the video codec.
Must the camera operation be checked?
No
Yes
Check the camera is open and display a frame to the user.
Init. the parameters for the Crazyradio and Crazyflie.
Set the initial position of the Crazyflie as its origin point (0,0,0).
Reset the extended Kalman filter session on the Crazyflie.
Calibrate the extended Kalman filter for the state of the Crazyflie.
Read a frame from the camera to initialise the image processing.
Send the setpoint function for the Crazyflie to take off.
Must the Crazyflie keep flying?
No
Yes
Read the live frame from the camera.
Convert the BGR colour frame into a grayscale frame.
*1 Grayscale Processing
Do a binary threshold with a threshold of 200 and new of 255.
*2 Grayscale Processing
Apply a morphology transformation with a kernel matrix of 5x5.
Find the contour with maximum area to be assumed as the target.
Get the area, centroid, and orientation of the target in pixels.
Find the centroid of the target in the new coordinate system.
Find orientation of the target in the new coordinate system.
Display the processed frame to the user for live monitoring.
Is target area over 100px^2 to 1000px^2?
No
Yes
Apply the PCM to find the altitude and position of the target.
Send a hover setpoint to the Crazyflie with the desired location.
Wait for 1ms to check if the termination "Q" key was pressed.
Send the setpoint function for the Crazyflie to land.
Basic post-processing of the overall test and recorded video.
Report an issues if any errors occurred.
End
Alt. Image Processing
Convert the BGR colour frame into a HLS colour frame.
*1 Colour Processing
Do a binary threshold with lower and upper bounds for the colour.
*2 Colour Processing

Figure 31: High-level flow diagram of the control flow for the moving target detection and tracking with regards to three degrees of translational freedom and a degree of rotational freedom as yaw orientation. (The suitable HLS colour bounds for the dark red are (150, 40, 40) and (210, 250, 250)).

For communication between the Crazyflie and Raspberry Pi with the Crazyradio, a radio frequency of 2.480GHz will be used with a data rate of 2MB/s and radio address of 0xE7E7E7E7E8 (change from the default radio address of 0xE7E7E7E7E7 to avoid interference from other Crazyflies). Because the control scripts will be running on the Raspberry Pi, a RDP is used to control the Raspberry Pi from the ground control laptop, such that the control scripts can be executed and terminated when desired.

5.6 TARGET MOTION

Throughout the tracking, the target will remain on the ground and can move in any direction on this plane at any moment. The target can also be seen to move from a starting point to reach a destination point along a path which mimics movement according to independent intentions without influence from the camera or Crazyflie - in other words, the target is unaware of the tracking and will not

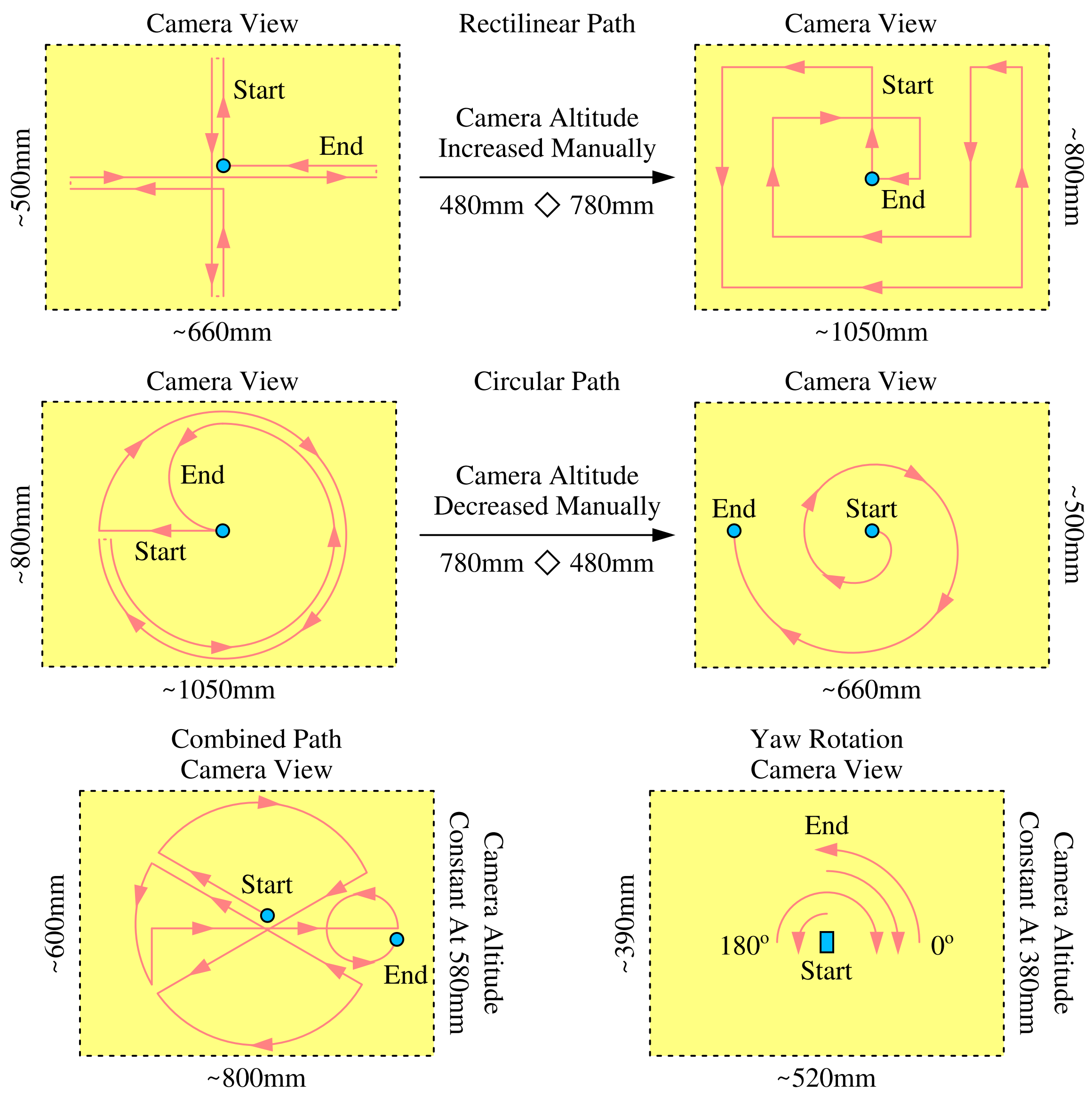


Figure 32: Paths on which the target will aim to move to test rectilinear (top), circular (middle), combined (bottom-left), and yaw rotation (bottom-right) motions. (The actual movement may differ).

intentionally try to assist or avoid the tracking. Furthermore, it is also assumed that the environment is clear from static and dynamic obstacles for both the target and Crazyflie.

Initially, the strength of the PWM signals for each motor will be calibrated in preparatory development. To evaluate different motion characteristics, three tests will be performed and aimed at validating rectilinear motion with primary movements along latitudinal and longitudinal lines, circular motion with primary movements across latitudinal and longitudinal lines, and a combination of rectilinear and circular motions in a seemingly random manner. Throughout the motions, the target will move at various speeds up to 0.20m/s as well as being stationary for sporadic periods of time, where the Crazyflie will be required to hover at a fixed position above the target. Moreover, an additional test will be performed for the yaw orientation, where the target is rotated both clockwise and counterclockwise between -90^{o} and 90^{o} at about 10^{o}/s. The planned path for these tests are annotated in Figure 32 - although the actual paths may slightly differ due to inconsistencies between the motors of the target, which is not an issue as these paths are essentially arbitrary and the Crazyflie must track the target regardless of the path. Each test will be performed two times to ensure repeatable results.

The altitude of the camera will also be altered for each of the tests and, for the rectilinear and circular motion tests, the target will actually stop at a midpoint and the altitude of the camera will be increased or decreased manually with the tripod. Over all of the tests, the variety of altitudes to be examined specifically include 380mm, 480mm, 580mm, and 780mm with the corresponding tests related in Figure 32. The respective fields of view from the camera at each altitude are measured to be approximately 520mm by 390mm, 660mm by 500mm, 800mm by 600mm, and 1050mm by 800mm.

5.7 DATA MONITORING

As mentioned in Figure 31, the current view from the camera will be displayed by the Raspberry Pi which is then streamed to the ground control laptop for live monitoring during the tests. It is also possible to immediately terminate the execution of the control script and land the Crazyflie, which is included for safety so a test can be cancelled if unexpected or unsafe behaviour is observed.

To validate the tests as successes or failures, the motion capture system will store the position of the target and Crazyflie in real-time at 100fr/s, such that any relative deviation between the current position of the target (which is the desired position of the Crazyflie) and actual position of the Crazyflie can be exactly quantified. Using the motion capture system is also more accurate than logging variables measured on-board the Crazyflie. The QTM will mostly handle running of the motion capture system, apart from starting the capture and defining which markers form rigid bodies for the target and Crazyflie, but it is necessary to initially calibrate the cameras with the following procedure:

1. Ensure there are no markers or reflective surfaces in view of the cameras. If there are reflective surfaces outside of the operating area which cannot be moved or covered, apply an auto-mask to digitally block these areas in the views of the camera.

2. Position the calibration L-tool in the centre of the operating area, where the corner will define the origin, the long axis will form the *x*-axis, and the short axis will form the *y*-axis.
3. Select the 300mm carbon fibre calibration kit under Project Options with a wand-tool length of 300.8mm and begin the calibration in QTM for a time of about 20s to 30s.
4. While the calibration is running, proceed to wave the wand-tool completely around the perimeter, inside of the operating area, and through different orientations until the calibration is complete.
5. Ensure the results of the calibration are successful, the resulting residuals for each camera are similar, and the standard deviation for the markers on the calibration tools is low.

6 DATA ANALYSIS AND RESULTS

By implementing the methodology, the appropriate data is to be obtained in a primitive form. This data can then be processed with an analysis for meaningful results. Ultimately, this allows for the experimental application to be judged against the objectives to form conclusions. There are also minor justifications submitted for certain decisions, but these are further discussed in Section 7.

6.1 CAMERA OBSERVATIONS

Firstly, the fish-eye distortion is evaluated using the template similar to that on a chessboard. From a photograph of this pattern shown in Figure 33, it is seen that the distortion present in the image from the Pi Camera is inconsequential with insignificant distortion towards the edges. Thus, it can be initially assumed that the pinhole camera model is valid, especially near the central region of the image, and it is not necessary to implement compensation for distortion.

From Figure 33 and monitoring while the target was moving, it was evident that there was occasionally slight blurring of the captured image with the Pi Camera being unable to auto-focus. However, due to the methods for image processing, this is not concerning as the effects have no meaningful impact on the target detection since adequate noise reduction is applied.

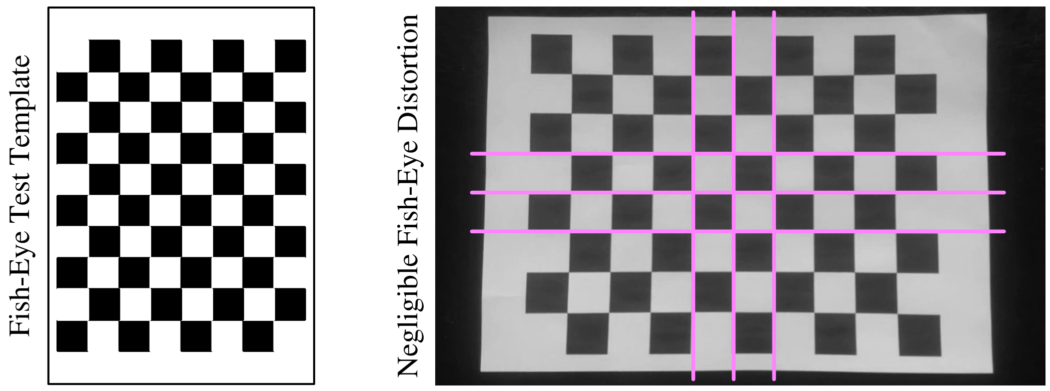


Figure 33: Photograph and evaluation of the fish-eye distortion in the Pi Camera. (The image was aligned with the pattern as best as manually possible but this may not be absolutely perfect).

6.2 IMAGE PROCESSING

Using the Raspberry Pi and Pi Camera, the collected data from the preliminary image processing to determine the most suitable resolution for acceptable real-time performance is shown in Table 2, and example frames illustrating the available detail at each resolution with grayscale and colour processing are seen in Figure 34. Based on the desired criteria, it is clear that a resolution of 160px by 120px and grayscale processing attains the best performance with an average frame rate around 30.1fr/s and time of 33ms between frames, although colour processing would still be sufficient if desired. The higher resolutions result in unacceptable frame rates and a noticeable lag was also observed between the displayed frame and real-time arrangement. This enables setpoints to be generated at about 30.1Hz which should allow for smooth flight without instability - although this may be slightly decreased due to the lightweight but noteworthy processing to control the Crazyflie.

Table 2: Evaluation of the most suitable resolution for acceptable real-time performance, comparing various levels of no image processing, grayscale processing, and colour processing with resolutions of 160px by 120px (left), 320px by 240px (middle), and 640px by 480px (right).

1. 160px By 120px

	Record Time [s]	Total Frames [fr]	Average FPS [fr/s]	Average SPF [s/fr]
A	18.9	588	31.1	0.032
	22.9	707	30.9	0.032
	24.7	772	31.2	0.032
B	27.3	823	30.1	0.033
	25.0	753	30.1	0.033
	25.5	766	30.0	0.033
C	22.3	280	12.6	0.080
	17.5	217	12.4	0.081
	28.1	355	12.6	0.079
D	21.6	542	25.1	0.040
	22.6	561	24.8	0.040
	23.8	585	24.6	0.041
E	22.0	249	11.3	0.088
	23.5	267	11.4	0.088
	24.7	275	11.1	0.090

2. 320px By 240px

	Record Time [s]	Total Frames [fr]	Average FPS [fr/s]	Average SPF [s/fr]
A	20.3	217	10.7	0.094
	21.2	233	11.0	0.091
	23.6	261	11.1	0.090
B	17.9	149	8.33	0.120
	22.4	191	8.53	0.117
	25.3	214	8.46	0.118
C	17.8	60	3.37	0.296
	18.5	65	3.52	0.284
	27.7	96	3.47	0.288
D	17.5	116	6.63	0.151
	14.5	94	6.49	0.154
	27.2	181	6.66	0.150
E	28.5	92	3.23	0.310
	22.2	72	3.25	0.308
	28.4	93	3.28	0.305

3. 640px By 480px

	Record Time [s]	Total Frames [fr]	Average FPS [fr/s]	Average SPF [s/fr]
A	18.7	72	3.85	0.260
	19.3	73	3.78	0.264
	15.5	59	3.81	0.263
B	27.6	107	3.88	0.258
	21.8	81	3.72	0.269
	24.3	93	3.83	0.261
C	38.3	41	1.07	0.934
	34.9	39	1.12	0.894
	30.1	33	1.10	0.912
D	26.8	81	3.02	0.331
	35.4	105	2.97	0.337
	23.2	67	2.89	0.346
E	14.2	14	0.99	1.013
	29.4	28	0.95	1.049
	22.7	21	0.93	1.080

A = None, B = Min. Gray Process, C = Max. Gray Process, D = Min. Colour Process, E = Max. Colour Process.

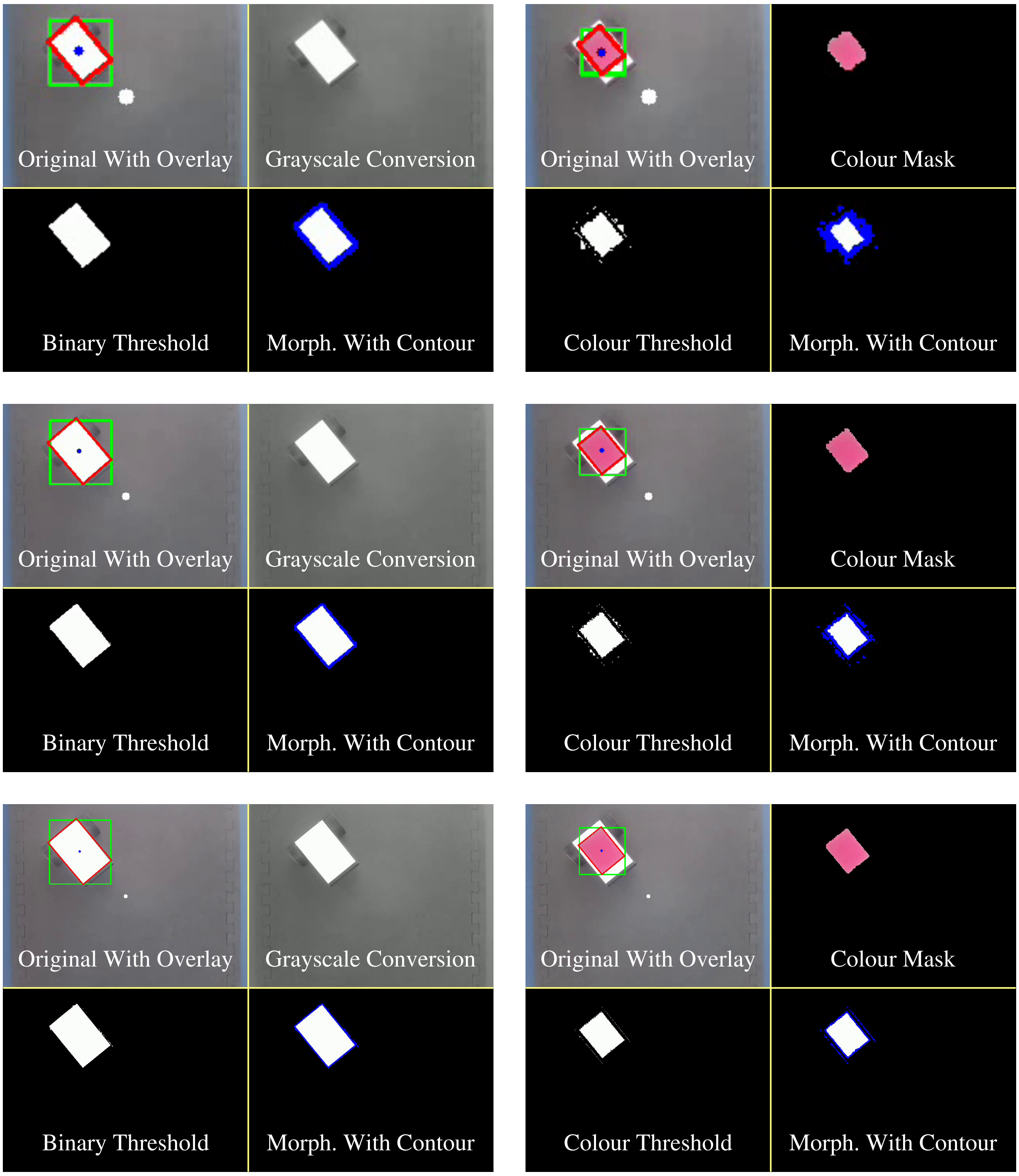


Figure 34: Example frames and processing captured at 160px by 120px (top), 320px by 240px (middle), and 640px by 480px (bottom), with an overlay of the target detection (as a graphical convenience for monitoring) found through grayscale processing (left) and colour processing (right).

While the camera calibration was being completed, the current area of the target in the captured frame was printed to the shell. Unexpectedly, it was gathered from this that the execution drastically slowed down with a sharply dropped frame rate and lag of approximately 2s to 3s between the displayed frame and real-time arrangement, even though the resolution remained the same. By removing the commands to print to the shell for confirmation, it was concluded that printing to the shell indeed caused these issues and is not a viable option for real-time monitoring of the camera view.

The rasterisation of the target was also considered for the disk and rectangle targets. The comparison for a resolution of 160px by 120px is seen in Figure 35, where it was observed that the pixelated appearance of the rectangle target was greatly altered depending upon the orientation, but there was almost no change in the pixelated appearance of the disk target for any orientation.

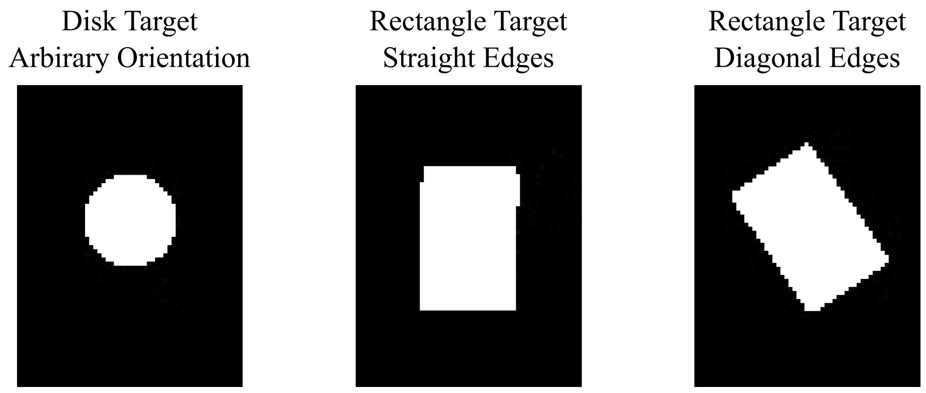


Figure 35: Rasterisation at 160px by 120px for the disk (left) and rectangle (middle and right) targets. (These images have been cropped from the appearance of the targets during the altitude calibration).

6.3 CAMERA CALIBRATION

For the camera calibration, the altitude of the camera was increased between 300mm and 780mm corresponding to a decrease in the area of the target, with example frames captured during the process shown in Figure 36. The recorded data is plotted in Figure 37 which allows for the trend relationship to be evaluated and used for the Crazyflie to maintain an altitude. By manipulating Equation 16 to form Equation 26, the focal length in terms of pixels can be determined using the actual area of 7088.2mm^2 for the disk target. To achieve confirmation, the rectangle target of 13300mm^2 was also used for the same calibration methodology with the recorded data plotted in Figure 38.

$$f = \frac{k}{\sqrt{A_{xy}}} \tag{26}$$

Where f, focal length, px; k, camera correlation coefficient, m.px; and A_{xy}, actual area, m^2.

From Equation 16 and Equation 17, it is evident that it is theoretically expected for a power relationship with a correlation coefficient and exponent of -0.5. From the recorded data, the disk calibration resulted in a trend relationship given by Equation 27 with an exponent of -0.484. and the rectangle calibration resulted in a trend relationship given by Equation 28 with an exponent of -0.492. With the correlation coefficient and Equation 26, the focal length in terms of pixels is then found to be 111px for the disk calibration and 116px for the rectangle calibration, which is an actual percentage difference of 4.41% and magnitude percentage difference of 0.931% using Equation 29 and Equation 30.

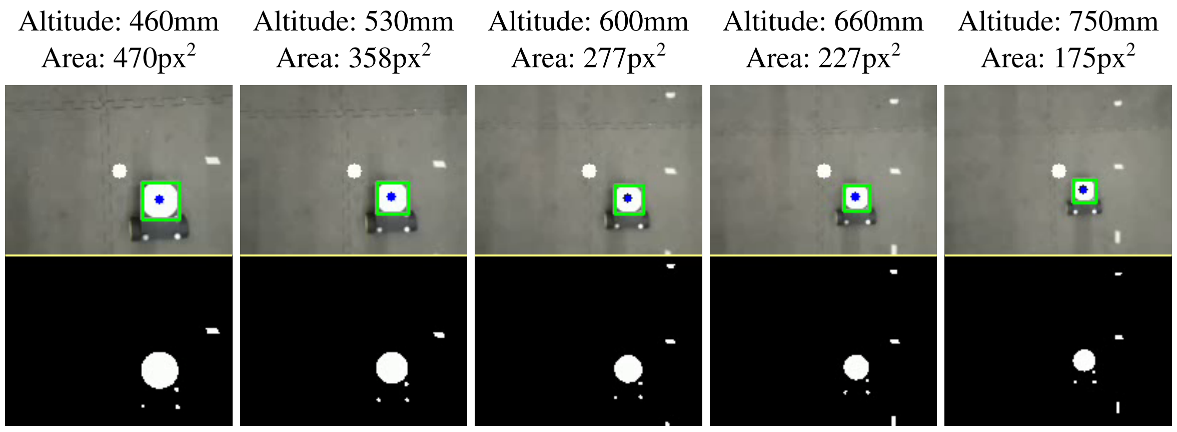


Figure 36: Example frames captured at 160px by 120px using grayscale processing at altitudes between 460mm and 750mm for the camera calibration using a disk as the target with a radius of 95mm, with an overlay of the target detection (as a graphical convenience for monitoring).

$$z = 9.128A_{uv}^{-0.484} \tag{27}$$

$$z = 13.47A_{uv}^{-0.492} \tag{28}$$

$$\%p_a = \left|\frac{a-b}{(a+b)/2}\right| \times 100\% \tag{29}$$

$$\%p_m = \left|\frac{\log(a)-\log(b)}{(\log(a)+\log(b))/2}\right| \times 100\% = \left|\frac{\log(a/b)}{\log(ab)/2}\right| \times 100\% \tag{30}$$

Where z, altitude, m; A_{uv}, virtual image area, px^2; $\%p_a$, actual percentage difference; a, first arbitrary variable; b, second arbitrary variable; and $\%p_m$, magnitude percentage difference.

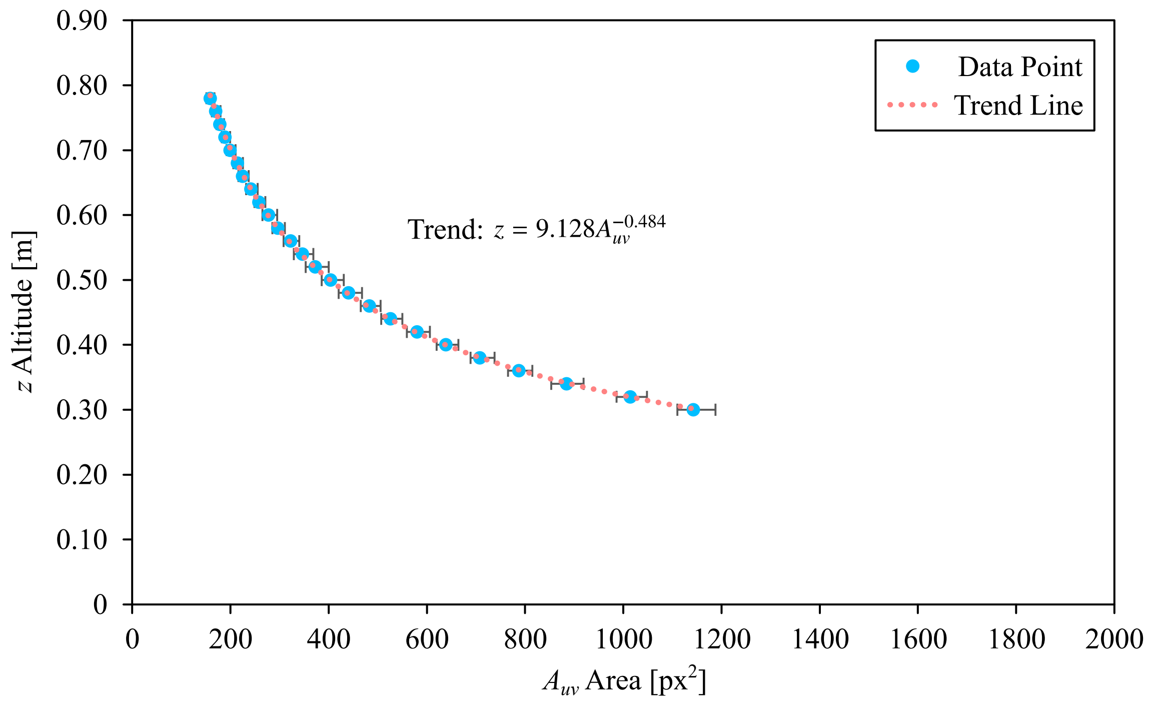


Figure 37: Altitude calibration using a disk as the target with a radius of 95mm. (The uncertainty in each altitude measurement was about ±5mm which is not distinguishable on the plot).

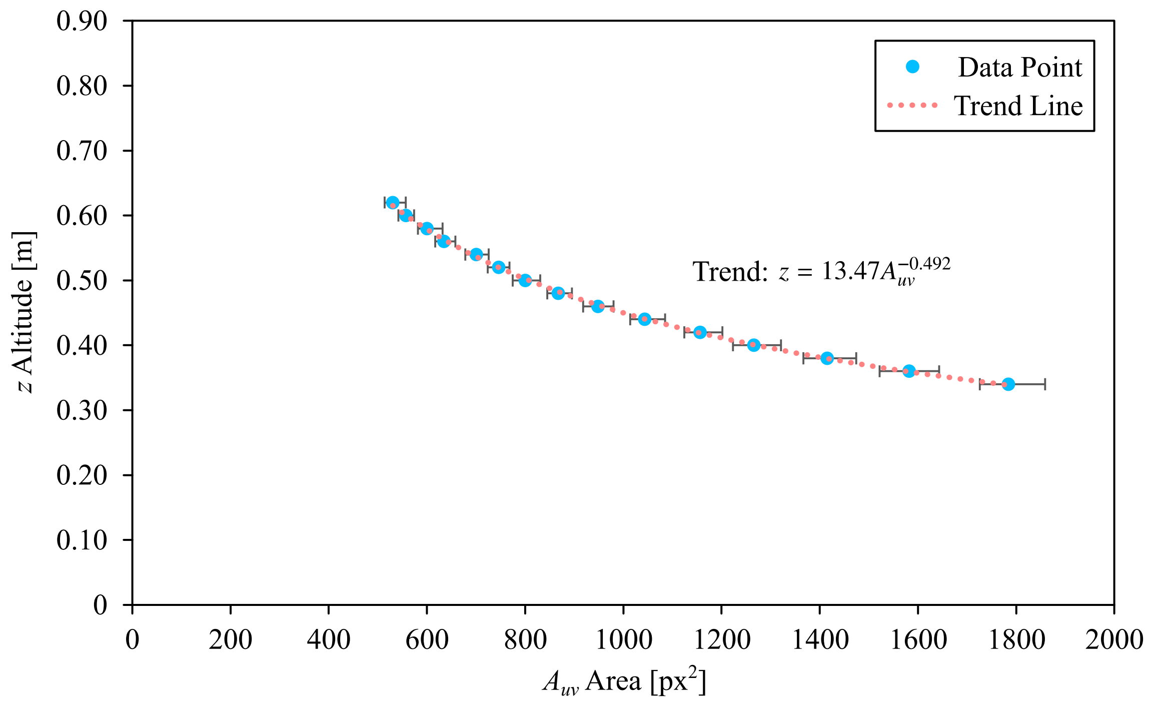


Figure 38: Altitude calibration using a rectangle as the target with edges of 140mm by 95mm. (The uncertainty in each altitude measurement was about ±5mm which is not distinguishable on the plot).

6.4 QUADROTOR OBSERVATIONS

Considering ground effects as the increased lift and decreased drag experienced when flying close to a fixed surface, there were no noticeable ground effects once the Crazyflie had taken off and reached the desired altitudes above 300mm. For an aircraft not designed to normally operate with ground effects, these ground effects can be identified by abnormal flight control when flying near the ground, but this was not present in a perceivable form as the Crazyflie was tracking the target.

The Flow deck was able to successfully stabilise the Crazyflie and hover at a fixed position without the need for roll or pitch trimming. However, during take off, there were instabilities where the behaviour of the Crazyflie was partially uncontrolled and erratic, and this had to be corrected by accounting for the slightly altered origin after take off. There were also indications of very slow and minor drift in the yaw orientation over extended periods of time, but this was seen to be almost undetectable for short flight times and did not affect the performance of the Crazyflie.

6.5 VISUAL SERVOING

While the tests were being performed, the primitive data was collected in the form of the three-dimensional position and three-dimensional rotation of both the target and Crazyflie. To extract meaning from these data, it is necessary to analyse it with regards to lag time in position, target detection robustness, relative deviation fluctuations, and ability to track the target at varying speeds.

6.5.1 PRIMITIVE DATA

From the motion capture system, the basic three-dimensional position of the target and Crazyflie was recorded and is shown in Figure 39 to Figure 41 for each trial in each of the rectilinear, circular, and combined tests respectively. The position of the target is directly presented relative to its starting position, while the position of the Crazyflie is re-evaluated relative to its initial position after taking off due to the mentioned irregularities. Unfortunately, there was a slight delay in the first trial of the rectilinear test when manually raising the tripod which contributed to an extended total time, but there was no meaningful impact on the results. (As mentioned, the exact uncertainty of the motion capture is not precisely known but it is anticipated to be on the order of millimetres).

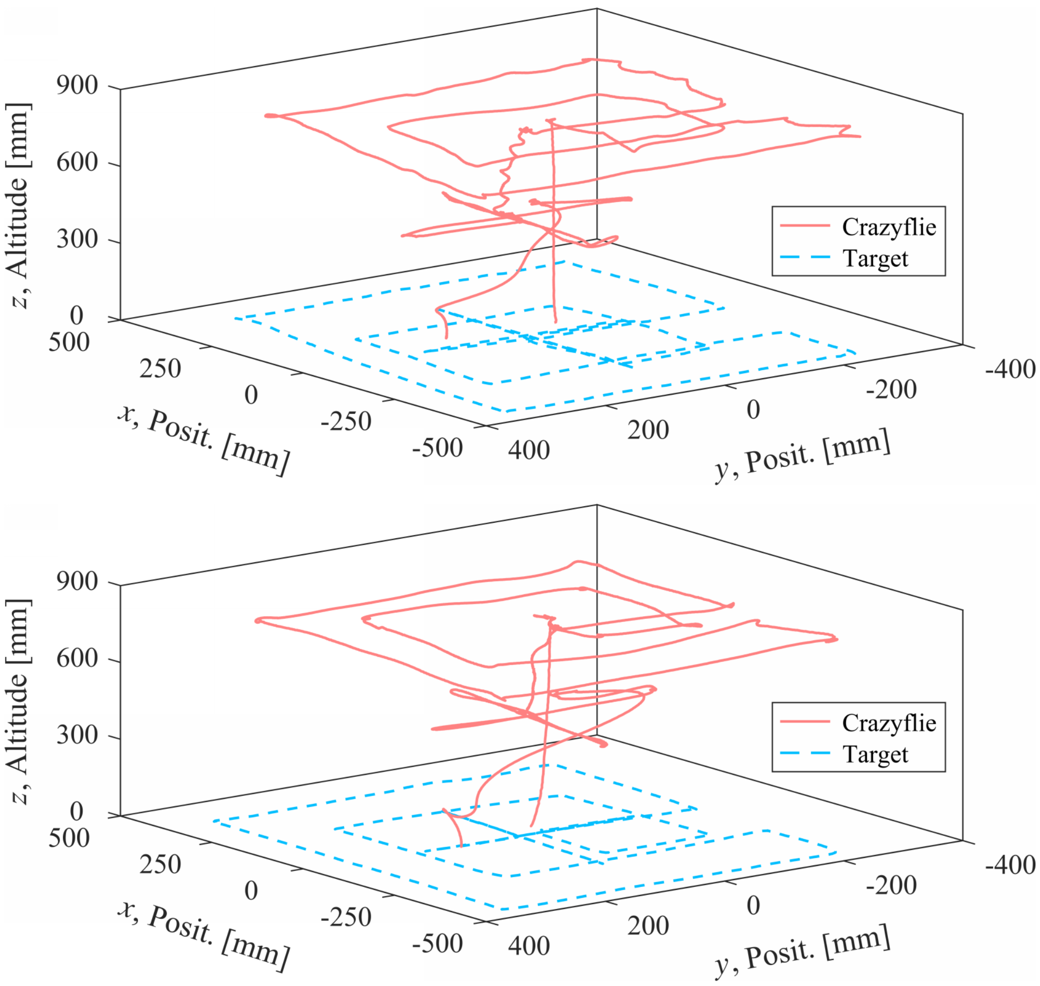


Figure 39: Position of the markers in the first (top) and second (bottom) trials of the rectilinear test.

The trials for the yaw orientation tests were also completed fairly successfully. The primary results are given in Section 6.5.2 with the analysis, since the three-dimensional position of the target and Crazyflie is mostly irrelevant when considering the yaw orientation, but it was observed that the Crazyflie would slightly tremor with periods of minor instability as it tried to maintain a fixed position while only yawing (as expected, the target simply rotated without translational motion).

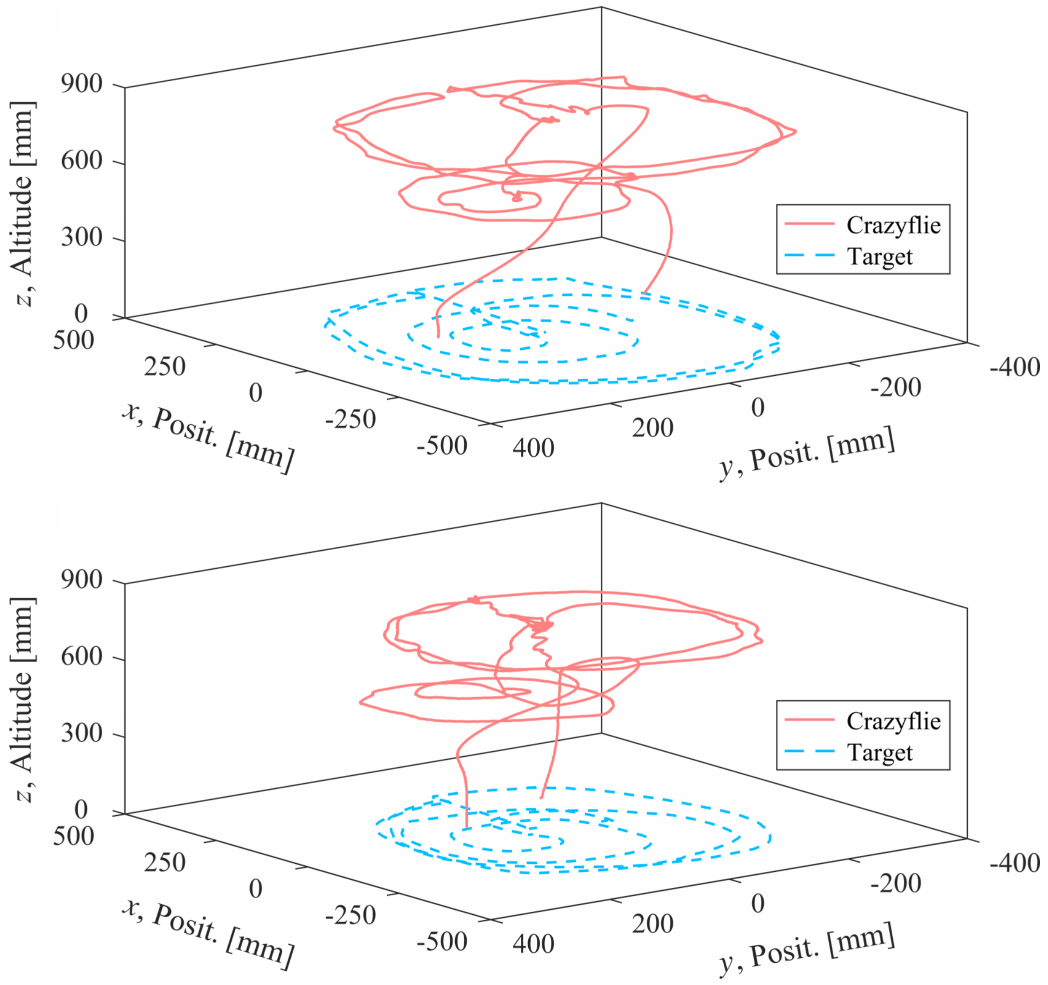


Figure 40: Position of the markers in the first (top) and second (bottom) trials of the circular test.

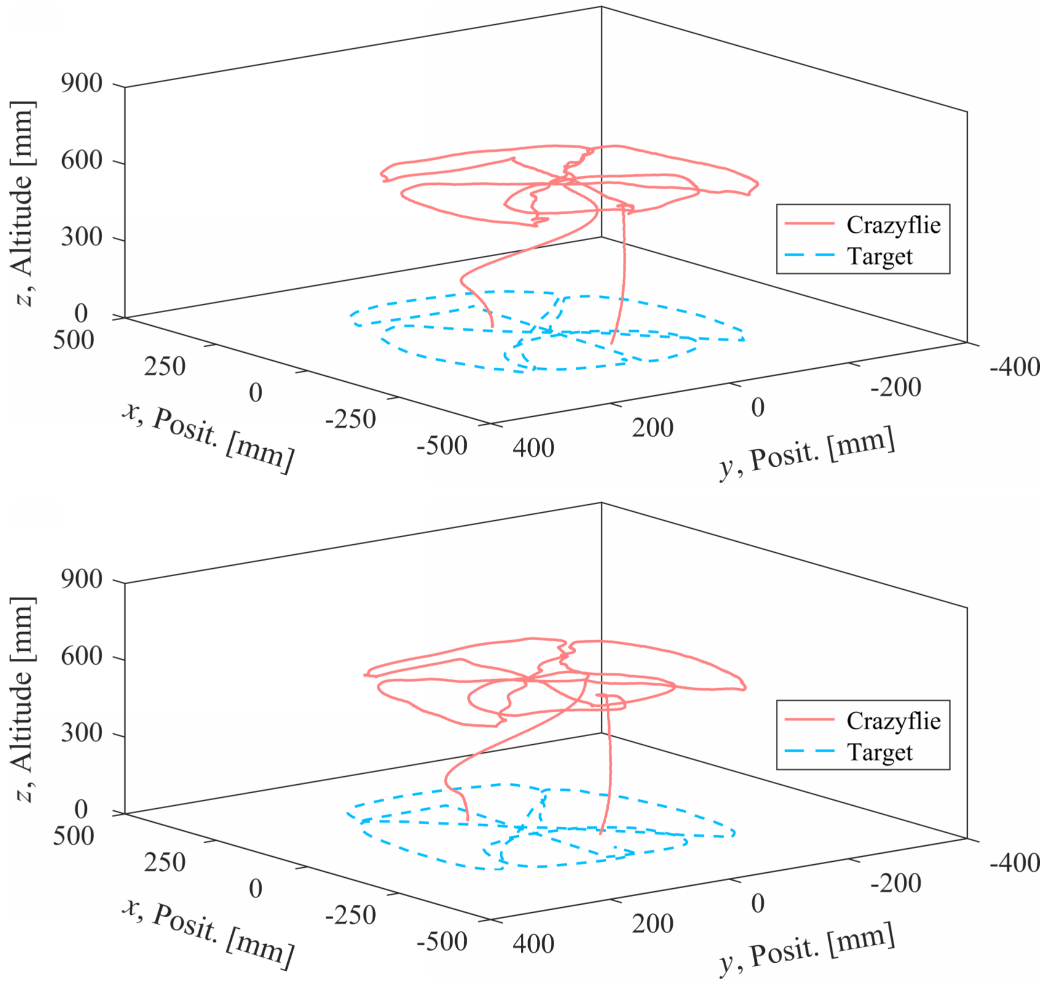


Figure 41: Position of the markers in the first (top) and second (bottom) trials of the combined test.

6.5.2 RESULTS ANALYSIS

For the translation tests, the relevant results for each trial are divided into the latitude and longitude position of the target and Crazyflie, magnitude of the deviation between the relative position of the target and Crazyflie as determined with Equation 31, and altitude of the Crazyflie. These factors are plotted against time and analysed in Figure 42 to Figure 47 for each of the translation tests.

$$D_{xy} = \sqrt{(x_t - x_q)^2 + (y_t - y_q)^2} \tag{31}$$

Where D_{xy}, latitude and longitude position deviation, m; x_t, target latitude position, m; x_q, quadrotor latitude position, m; y_t, target longitude position, m; and y_q, quadrotor longitude position, m.

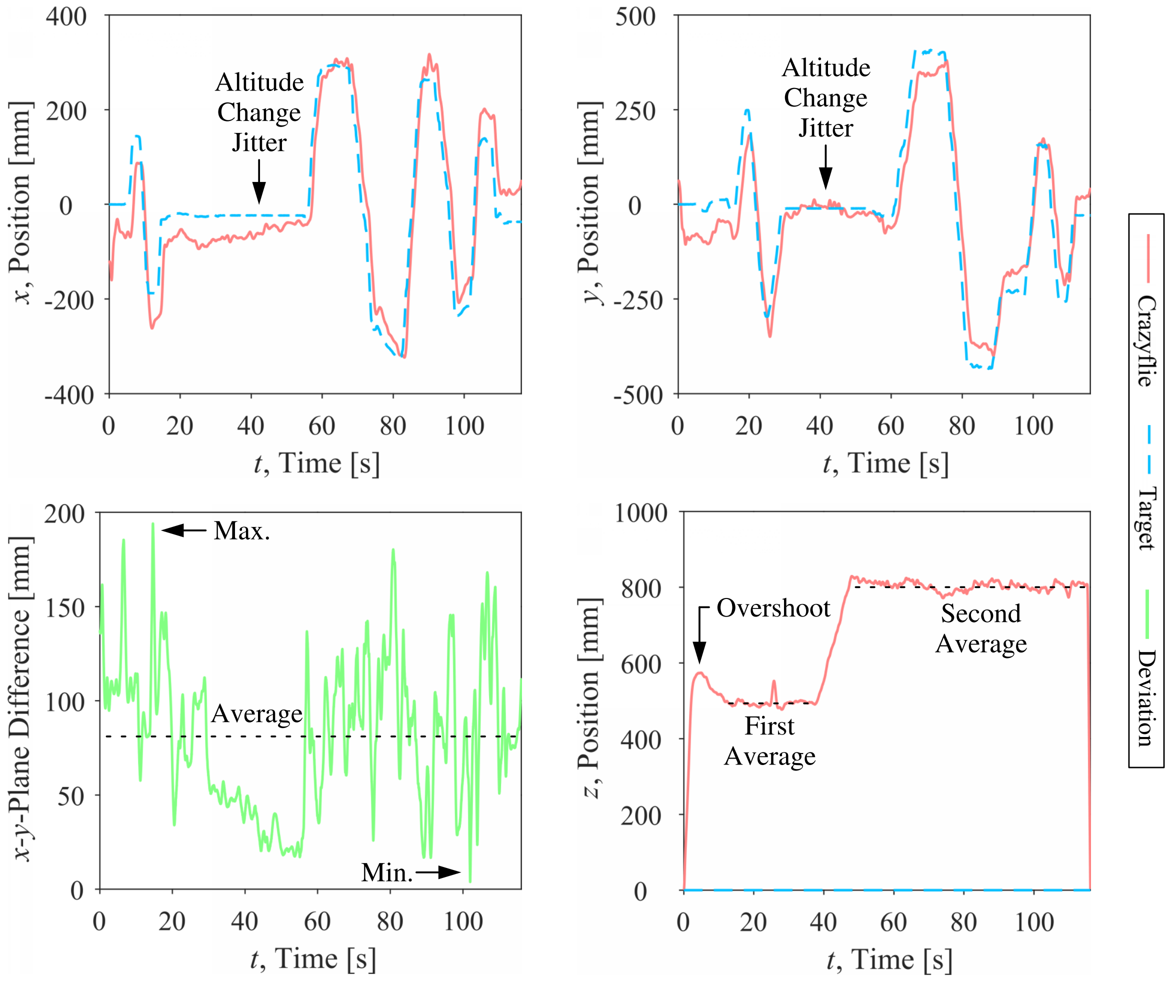


Figure 42: Relevant results against time throughout the first trial of the rectilinear motion test.

When the altitude was being changed in the rectilinear and circular motion tests, the Crazyflie jittered irregularly in the latitude and longitude positions due to the manual adjustment of the camera, where the camera could not be kept at a constant position during this adjustment. However, once the succeeding altitude was reached, the Crazyflie was able to continue to operate without a distinct overshoot or any perceivable repercussions. There was also a slight lag throughout the tests between the actions

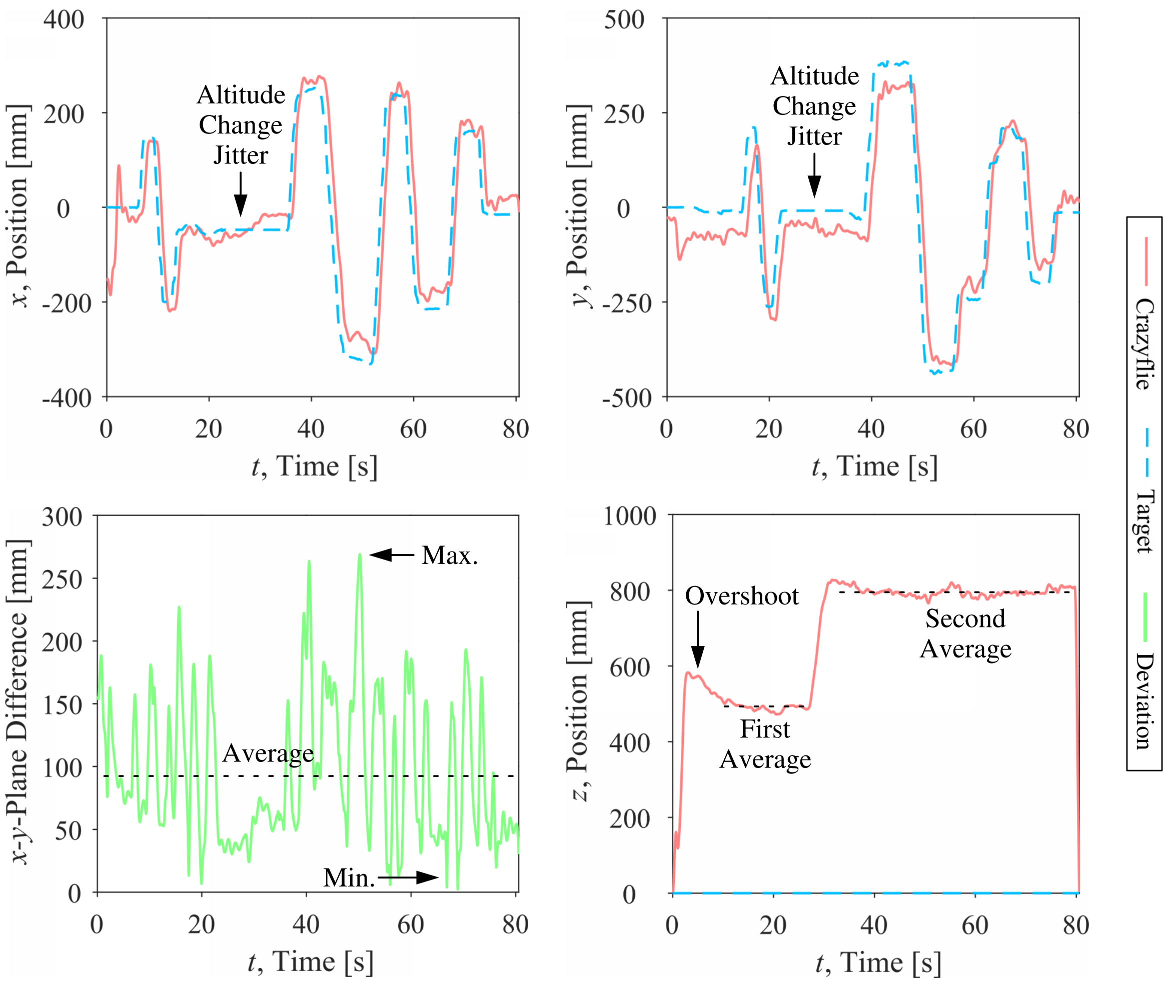


Figure 43: Relevant results against time throughout the second trial of the rectilinear motion test.

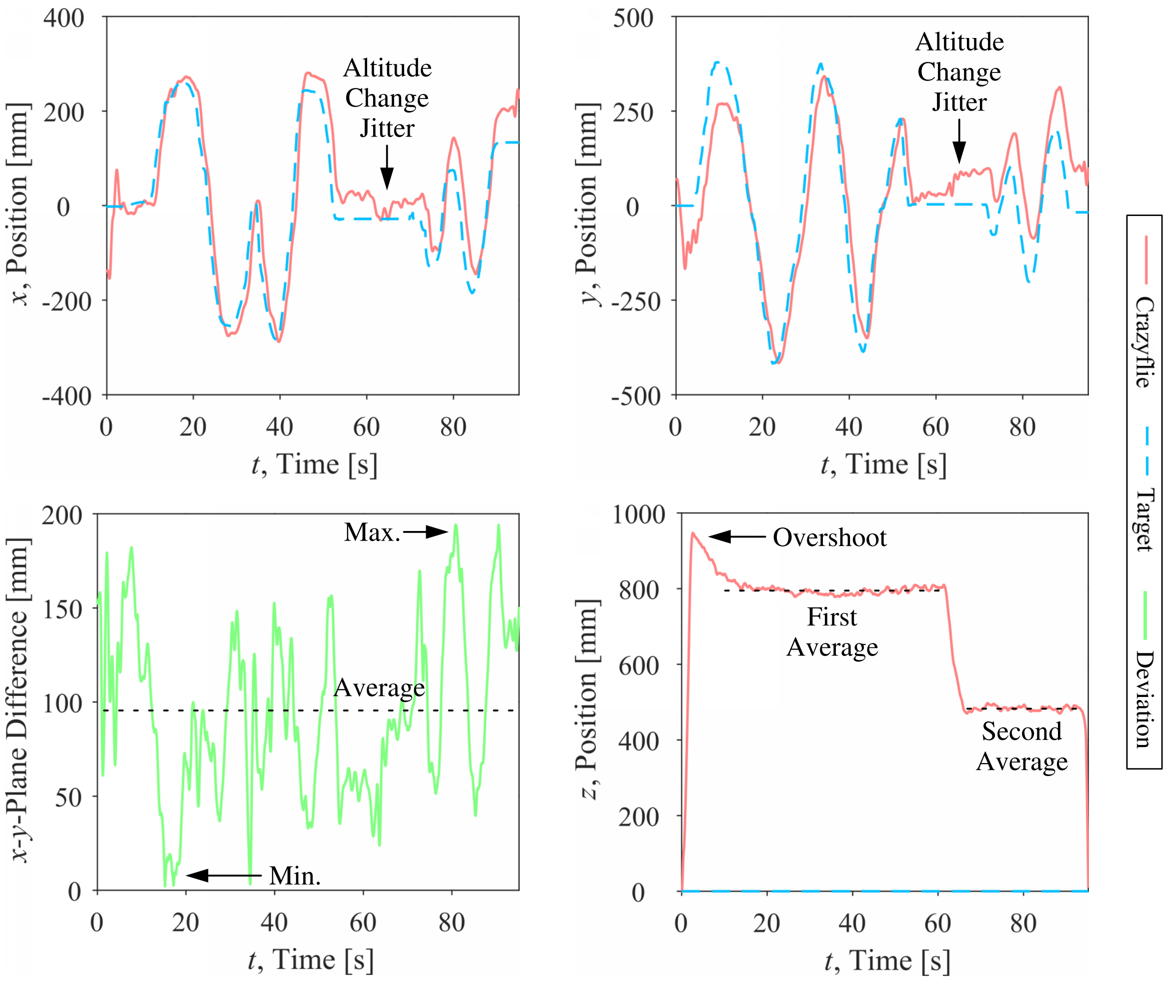


Figure 44: Relevant results against time throughout the first trial of the circular motion test.

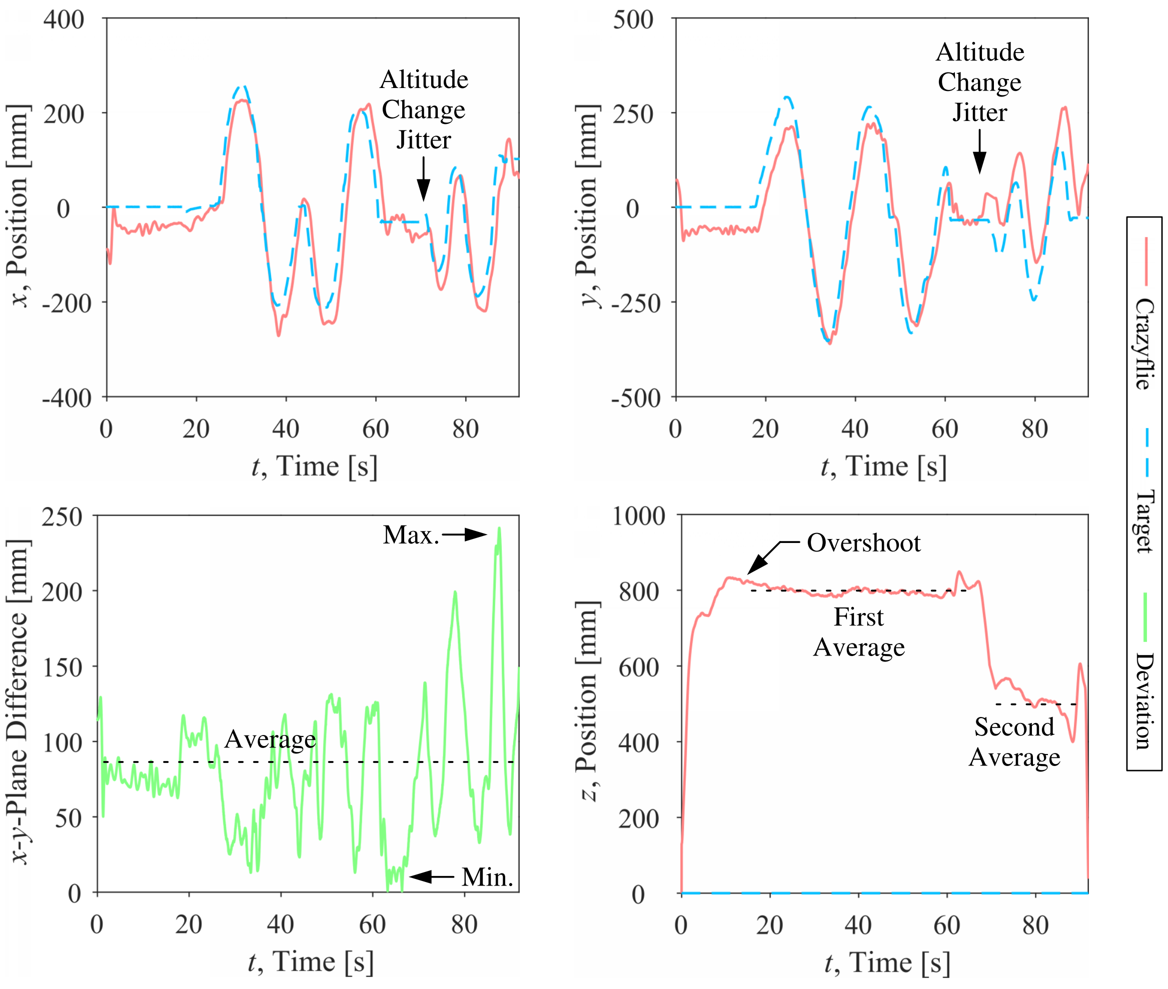


Figure 45: Relevant results against time throughout the second trial of the circular motion test.

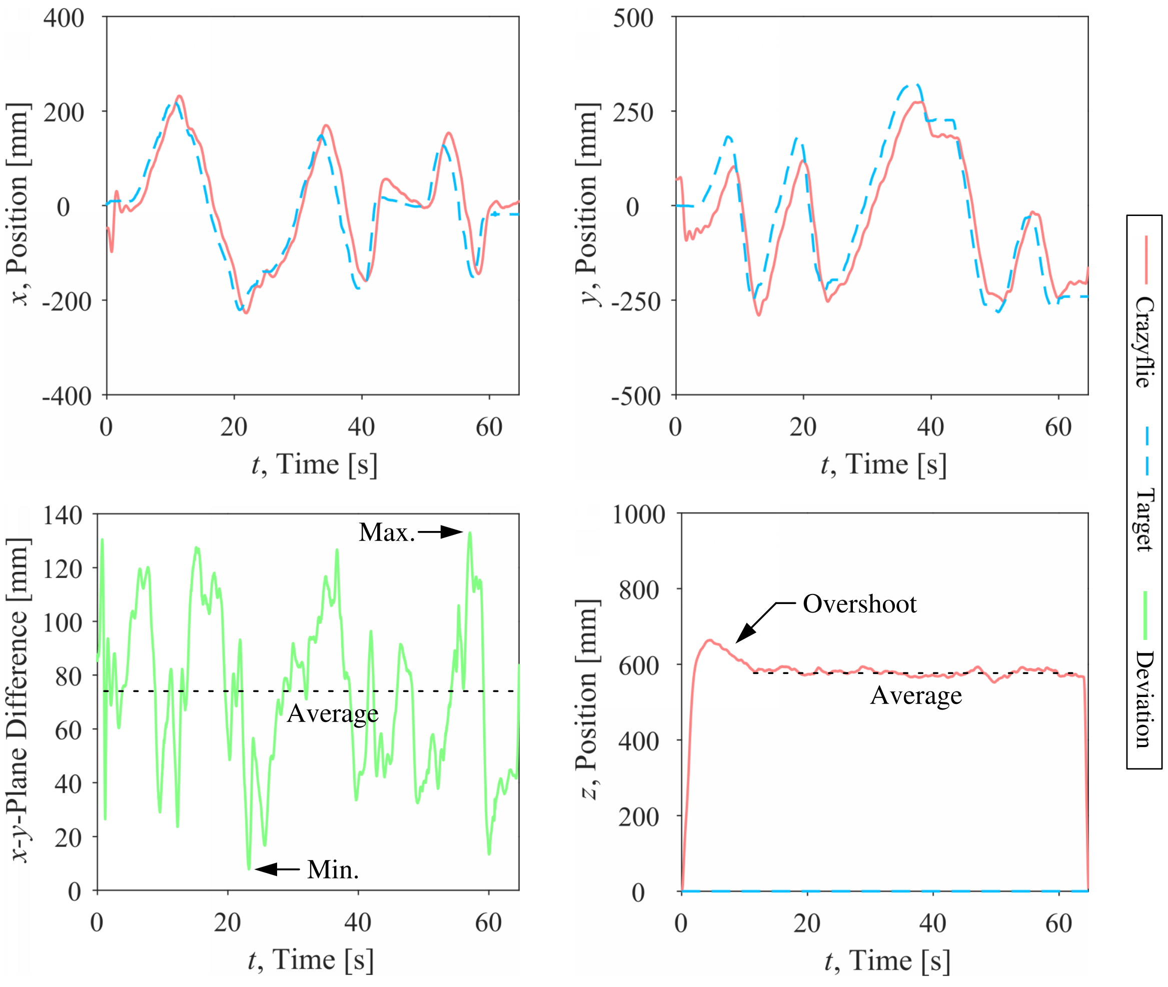


Figure 46: Relevant results against time throughout the first trial of the combined motion test.

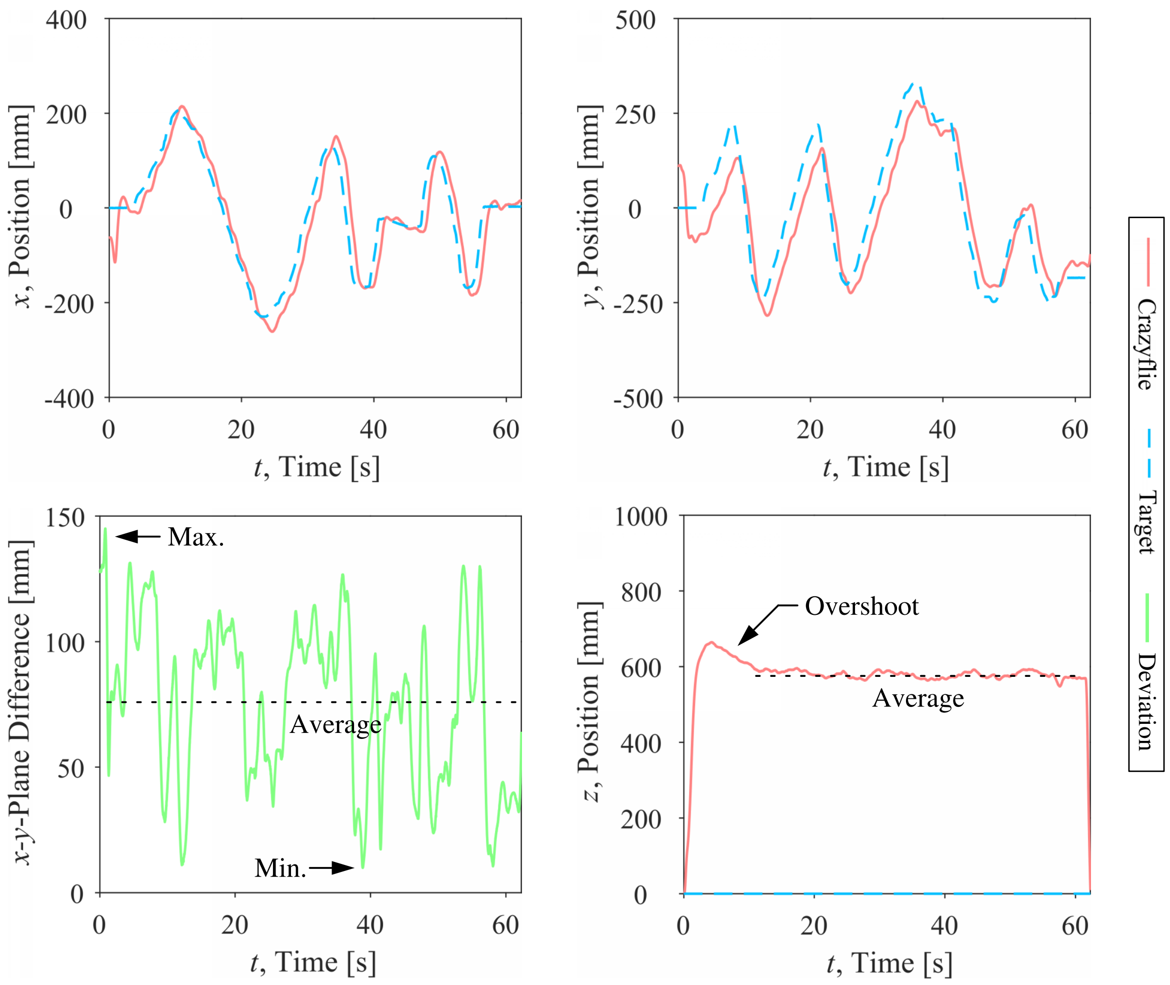


Figure 47: Relevant results against time throughout the second trial of the combined motion test.

of the target and response from the Crazyflie, especially when the target moved at higher speeds - the speed of the target ranged from stationary up to approximately 0.215m/s, which can be represented in terms of pixels using the pinhole camera model with Equation 32. This lag, along with the resulting position deviation between the relative position of the target and Crazyflie, are outlined in Table 3 with a comparison of the position deviation if the average lag is compensated (for discussion, the average values are more relevant since the magnitude of the deviation is being considered).

$$m = f\,\frac{n}{z} \longrightarrow \dot{m} = f\,\frac{\dot{n}}{z} \tag{32}$$

Where m, arbitrary virtual image distance, px; f, focal length, px; n, arbitrary actual distance, m; z, altitude, m; $\dot{m}$, arbitrary virtual image velocity, px/s; and $\dot{n}$, arbitrary, actual velocity, m/s.

For the yaw orientation test, the relevant results for each trial are divided into the angle of the yaw orientation for the target and Crazyflie, deviation between the relative angle of the yaw orientation for the target and Crazyflie as determined with Equation 33, and altitude of the Crazyflie. These factors are plotted against time and analysed in Figure 48 and Figure 49. A lag between the Crazyflie and target was still observed, although the average values were particularly less prominent. As outlined

Table 3: Outline of the lag and position deviation of the Crazyflie in the x-y-plane for the translation tests, with comparison for the deviation after adjustment accounting for the average lag.

		Total Time [s]	Lag Time [s]			Original D_{xy} [mm]			Adjusted D_{xy} [mm]		
			Minimum	Average	Maximum	Minimum	Average	Maximum	Minimum	Average	Maximum
Rectilinear	Trial 1	116.1	0.18	0.96	1.60	3.82	81.2	194	9.50	58.9	132
	Trial 2	80.60	0.46	1.02	1.79	2.24	95.9	269	2.94	56.0	136
Circular	Trial 1	95.15	0.39	0.78	1.23	2.06	94.6	194	8.77	67.2	138
	Trial 2	91.88	0.22	0.86	1.80	0.659	80.2	241	0.482	63.8	135
Combined	Trial 1	64.65	0.19	0.74	2.18	8.02	73.3	133	1.15	39.0	129
	Trial 2	62.29	0.31	0.67	2.61	10.2	75.1	145	2.14	39.3	117
Overall Averages		-	0.292	0.838	1.87	4.50	83.4	196	4.16	54.0	131

in Table 4, this lag results in an angular deviation between the yaw orientation of the target and Crazyflie (for discussion, the minimum and maximum values are more relevant since the direction of the deviation is included). The position deviation in the translation tests and angular deviation in these yaw orientation tests while compensating for the average lag are seen in Figure 50.

$$D_\psi = \psi_t - \psi_q \tag{33}$$

Where D_ψ, yaw orientation deviation, ^{o}; ψ_t, target yaw angle, ^{o}; and ψ_q, quadrotor yaw angle, ^{o}.

Table 4: Outline of the lag and angular deviation of the Crazyflie for the yaw orientation in the rotation tests, with comparison for the deviation after adjustment accounting for the average lag.

		Total Time [s]	Lag Time [s]			Original D_ψ [^{o}]			Adjusted D_ψ [^{o}]		
			Minimum	Average	Maximum	Minimum	Average	Maximum	Minimum	Average	Maximum
Yaw	Trail 1	69.36	0.34	0.45	0.88	-9.54	0.185	9.87	-4.51	0.136	4.46
	Trail 2	67.06	0.12	0.38	0.64	-8.32	-1.08	8.95	-4.76	-1.04	3.96
Overall Averages		-	0.23	0.415	0.76	-8.94	-0.45	9.42	-4.64	-0.45	4.21

In each of the tests, there may be partial effects on the minimum and maximum values from the target suddenly doubling-back which artificially fluctuates the lag and position deviation. This should be minimal during the rotation tests due to the low average lag, but it may be noticeable on the translation tests although the stationary periods before doubling-back should reduce these effect.

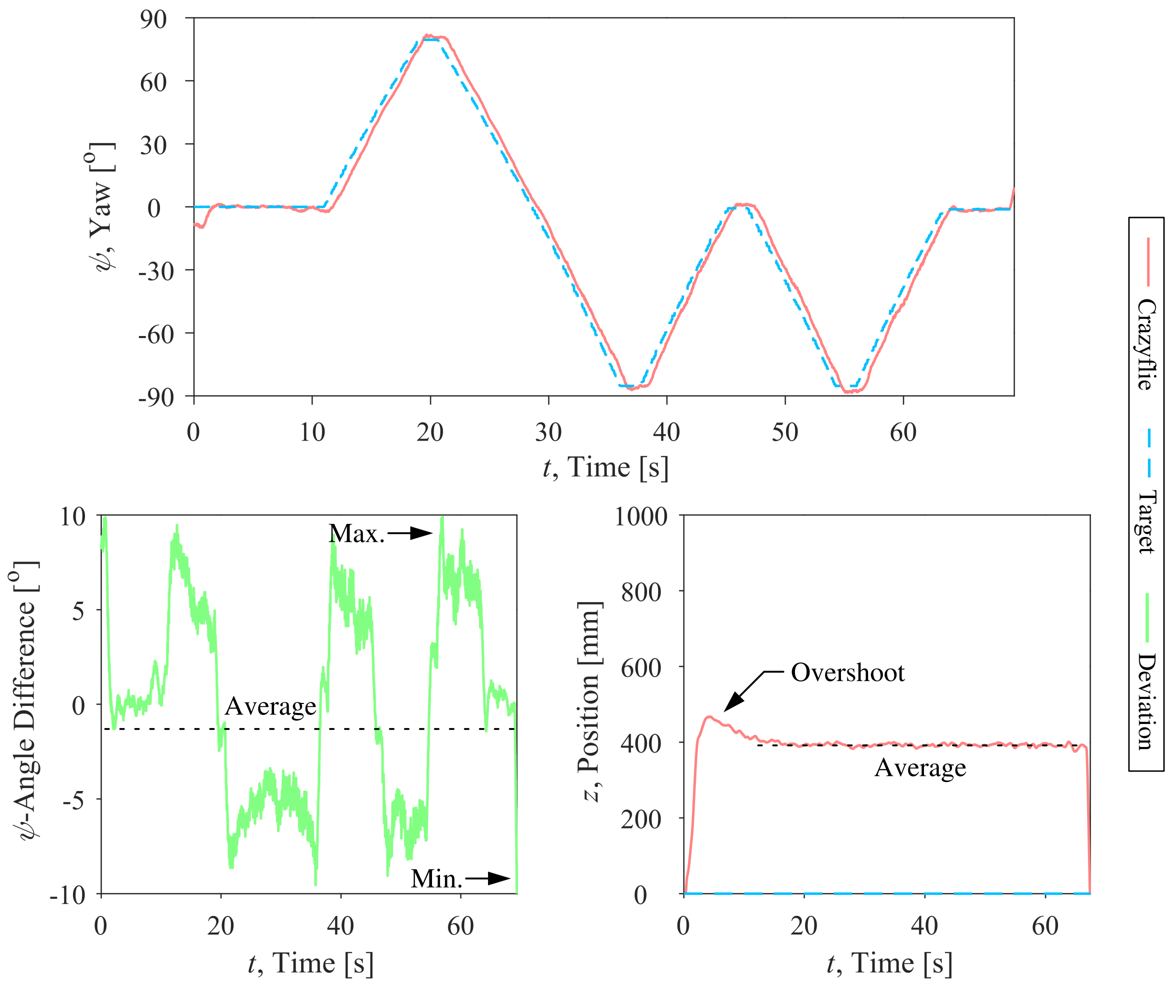


Figure 48: Relevant results against time throughout the first trial of the yaw orientation test.

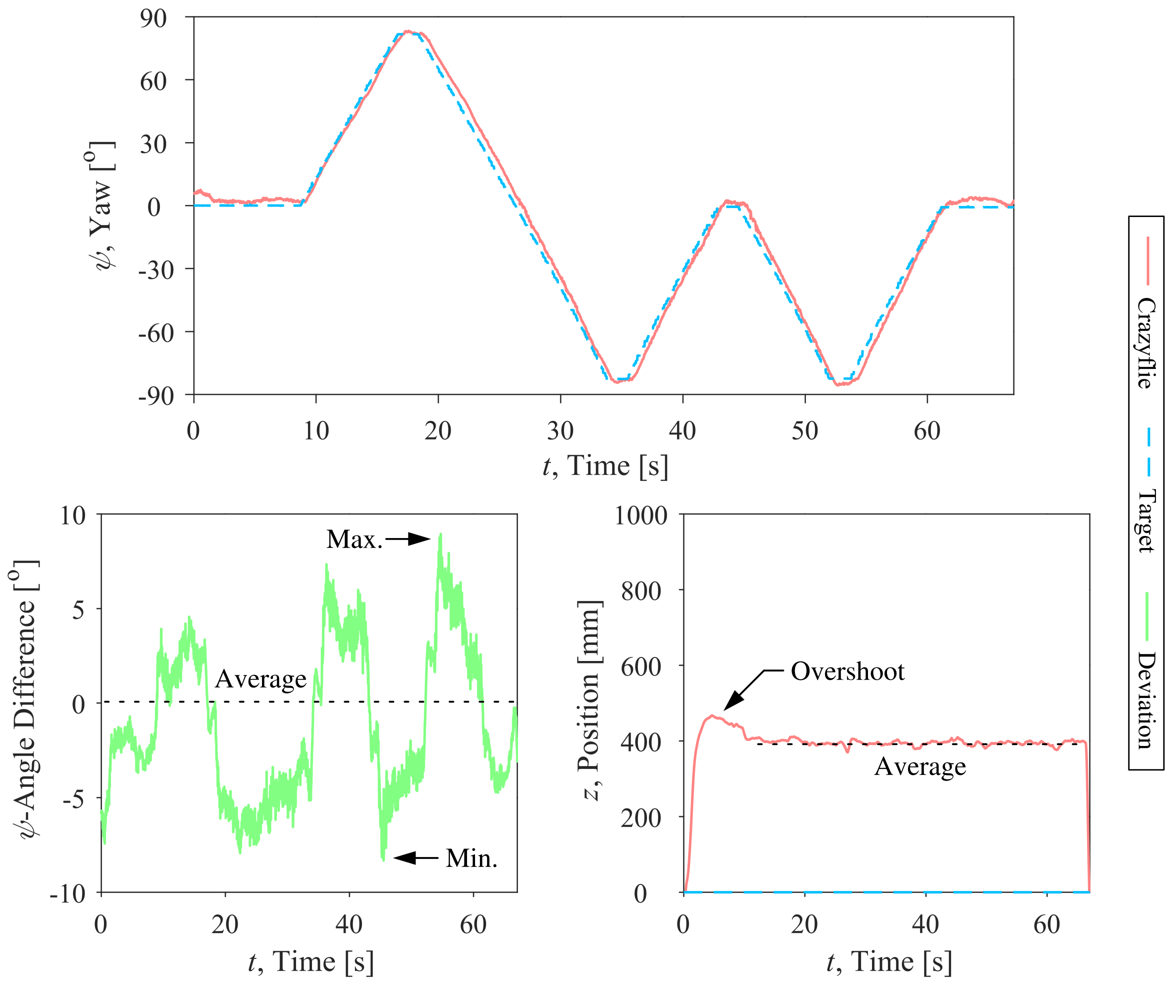


Figure 49: Relevant results against time throughout the second trial of the yaw orientation test.

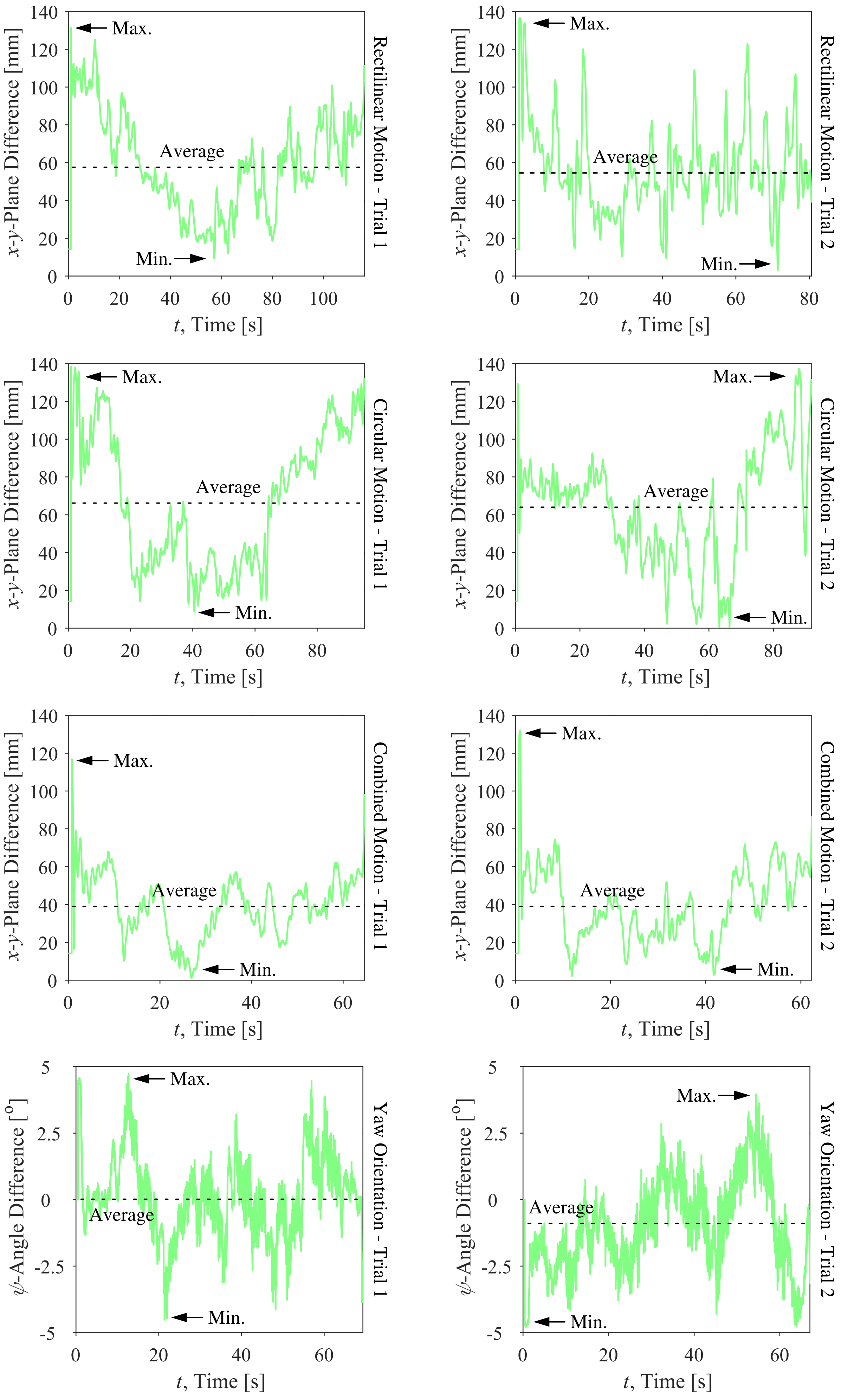


Figure 50: Deviations with adjustments for average lag in the translation and rotation tests.

To isolate the altitude behaviour, the average altitude at each part of the flight was shown on the altitude plots for each trial. As mentioned, the initial behaviour during take off is partially unstable with repeated overshooting, where the actual and magnitude percentage differences in the overshoot can be compared, with Equation 29 and Equation 30 respectively, using the average altitude over the following part of the flight. The compiled altitude results are included in Table 5.

Table 5: Outline of the altitude of the Crazyflie for each of the translation and rotation tests.

		Initial Overshoot [mm]	First Minimum [mm]	First Average [mm]	First Maximum [mm]	Second Minimum [mm]	Second Average [mm]	Second Maximum [mm]	Actual Overshoot [%]	Mag. Overshoot [%]
Rectilinear	Trial 1	574	477	495	551	772	799	824	14.7	2.38
	Trial 2	583	473	490	511	767	793	820	17.2	2.79
Circular	Trial 1	947	778	794	814	470	484	497	17.6	2.64
	Trial 2	833	780	793	808	471	510	539	4.98	0.75
Combined	Trial 1	663	552	576	593	-	-	-	14.1	2.22
	Trial 2	664	564	574	596	-	-	-	14.5	2.29
Yaw	Trail 1	467	375	393	404	-	-	-	17.3	2.90
	Trail 2	468	370	393	411	-	-	-	17.3	2.91
Overall Averages		-	-	-	-	-	-	-	14.7	2.36

7 DISCUSSION

Using inexpensive and low-end hardware in the form of a Raspberry Pi and Pi Camera for target detection with computer vision techniques in OpenCV, it was possible to implement position-based visual servoing through end-point open-loop control, where a Crazyflie effectively tracked the relative position of a target in a mirrored fashion. Specifically, as is required for the aim of the research, the Raspberry Pi is definitely considered to have marginal computational effort capabilities, since it only utilises a basic 1.0GHz single-core processor and 512MB of RAM, as compared to modern hardware where common consumer devices have multiple threads and cores with processing speeds over 1.8GHz and access to over 2GB of RAM - this is further highlighted by the cost of a Raspberry Pi which is only around R200 [29]. The results are discussed for direct relation to the objectives.

7.1 CAMERA AND PROCESSING FACTORS

Examining the first significant finding in Table 2, the image processing tests at different resolutions are clear and, in a sense, restrictive when considering the apparatus and computationally limited hardware in general. As a consequence, it would be absolutely infeasible to use computer vision techniques to control the Crazyflie in real-time with the default resolution of 640px by 480px in OpenCV, since a setpoint is only generated every 263ms (frequency of only 3.80Hz) on average with minimum grayscale processing; and, with a resolution of 320px by 240px, a setpoint can be generated every 118ms (frequency of 8.47Hz) on average with minimum grayscale processing, but this may be impractical as it is below the recommendation for setpoints to be generated at 10Hz as a minimum limit. By reducing the resolution to 160px by 120px, it was possible to achieve a generation of setpoints every 33ms (frequency of 30.1Hz) on average with minimum grayscale processing, while colour processing and maximum processing were also satisfactory if desired. The relationship between the resolution and time for each setpoint to be generated appears to be non-linear which is expected since the number of pixels in the image is decreasing by a factor of four as the resolution is halved in each dimension, but additional in-between resolutions should be investigated to find a reliable relationship, if this relationship is desired to be found (which was unnecessary for the aim of this research).

To compare the grayscale and colour processing, it is evident that the minimum grayscale processing obtains a higher frame rate over the minimum colour processing with actual percentage differences of 19.3%, 24.7%, and 25.6% (magnitude percentage differences of 2.93%, 6.17%, and 10.6%) for resolutions of 160px by 120px, 320px by 240px, and 640px by 480px respectively. Similar results are evident for the maximum processing with actual percentage differences of 10.1%, 6.55%, and 15.4% (magnitude percentage differences of 2.22%, 2.70%, and 5.26%) for resolutions of 160px by 120px, 320px by 240px, and 640px by 480px respectively. Furthermore, there was severely more noise observed around the target in the frames captured with colour processing, which results in a less accurate interpretation of the target in colour processing compared to grayscale processing.

The Pi Camera experienced no detrimental fish-eye distortion or effects from rolling shutter, and it was calibrated to find the relationship between the virtual target area and current altitude of the camera with the experimental trend relationships for the disk and rectangle targets expressed in Equation 27 and Equation 28 respectively. By averaging the 111px and 116px experimental focal lengths from the disk and rectangle targets respectively, the focal length in terms of pixels is estimated to be 113.5px. In addition, these remarkable correlations between the experimental trend relationships and with the theoretical predictions further reinforce and validate the legitimacy of the pinhole camera model. (It should be noted that this calibration is unique to the tested resolution of 160px by 120px, where higher resolutions will have a greater virtual target area due to the increased number of pixels).

Unfortunately, the calibration for the rectangle target suffered greater uncertainty than the calibration for the disk target, where the variation for the rectangle target increased from -16px to -58px for the minimum bound and from 27px to 75px for the maximum bound, and the variation for the disk target increased from -8px to -32px for the minimum bound and from 8px to 46px for the maximum bound (and this was over a larger range of altitudes). The power nature of this increase (observed in Figure 37 and Figure 38) is also expected as it mimics the power nature of the calibration. The reason for the greater uncertainty with the rectangle target is proposed to be due to the possibility for the orientation to change, which, with pronounced effects from the low resolution, results in poor rasterisation along the edges when they are diagonal compared to when they are vertical or horizontal. Conversely, the disk target appears almost identical for all orientations, since it is continuously symmetric and proportional and, thus, its uncertainty will be mostly independent from orientation.

7.2 VISUAL SERVOING PERFORMANCE

With the grayscale processing and calibration of the Pi Camera to use the pinhole camera model, the real-time target detection and tracking was implemented with the Crazyflie mirroring the motion of the target while maintaining the altitude of the Pi Camera. The take off of the Crazyflie was occasionally erratic with large variations in the latitude and longitude positions as it ascended. There was also initial overshooting before reaching the desired altitude, where the actual percentage difference in the overshoot averaged at 14.70% over all of the tests with a fairly consistent trend among the tests, which is expected since this is an independent event only associated with the take off before the target begins any motion. Nevertheless, this behaviour did not affect the operation once it had reached the desired altitude and, accordingly, the main substance of the results were not affected.

During the target detection and tracking, the Crazyflie lagged slightly behind the target in terms of position with an average of 0.838s for the translation tests and 0.415s for the rotation tests. A lag is expected since each frame is being processed for 0.33s on average and then the generated setpoint needs to be sent wirelessly over the Crazyradio to the Crazyflie which then needs to be processed internally. The reason for the difference in lag between the translation tests and rotation tests is due to the slower speed of the target in terms of pixels in the rotation tests compared to the translation tests. Elaborating, the maximum speed of the target was 0.215m/s in the translation tests which converts to 50.8px/s at an altitude of 480mm or 31.3px/s at an altitude of 780mm; and the maximum speed of the target was 10^o/s in the rotation tests which converts to 24.8px/s at an altitude of 380mm. Notably, the speed of the target in both cases is appropriately less than the 1.41m/s maximum speed of the Crazyflie, so the lag was not influenced by the target moving faster than the Crazyflie is capable.

Due to the lag between the Crazyflie and target, there were substantial deviations in position or yaw orientation experienced. The maximum position deviation reached 269mm with an average deviation of 83.4mm considering the translation tests, and the minimum and maximum yaw orientation deviations reached -9.54^o and 9.87^o respectively considering the rotation tests. However, for a fair conclusion, the deviations in position and yaw orientation should be compared after the lag has been compensated. As a result, the maximum position deviation is reduced to 138mm with an average deviation of 54.0mm, and the minimum and maximum yaw orientation deviations are also diminished considerably to -4.76^o and 4.46^o respectively. While viewing the plots of the position of the target and Crazyflie in Figure 42 to Figure 49 for subsequent trials, it is also evident that there is repeatable performance from the control script with consistent performance across each of the tests.

With regards to altitude, the Crazyflie maintained average altitudes of 373mm, 496mm, 558mm, and 787mm, for the expected altitudes of 380mm, 480mm, 580mm, and 780mm respectively (although the expected altitudes were not measured during operation). During each of the tests, the Crazyflie performed well in sustaining these average altitude with minor variances, where the typical minimum bound was only as low as 3.75% below the average value and the typical maximum bound was only as high as 3.98% above the average value based on an average of actual percentage differences.

7.3 OTHER FINDINGS AND REMARKS

There are multiple sources causing minor uncertainty in the results, such as the slight inaccuracy in measuring the target dimensions at ±1mm, discretization error and noise from the low resolution of the captured frames, altitude measurement error in calibrating the Pi Camera at ±5mm, use of underlying velocity setpoints by the Crazyflie to accomplish position setpoints, and reconstruction of the target position which consisted of error accumulation from its interdependent sources. The primary contributor to uncertainty is the measurement accuracy of the motion capture system, where the exact uncertainty is not precisely known but it is on the order of millimetres. For consolation, the important results are several orders of magnitude above this scale and can reasonably be assumed to be mostly insensitive to discrepancies presented by the motion capture system.

Although there were many sources of uncertainty, the control script was found to be reasonably robust, as long as the target is the largest contour within the captured frame for grayscale processing or no similar colours appeared within the captured frame for colour processing. With very minimal modification to its current form, the control script can also be suitable for an implementation of autonomous landing, where the desired location for landing is marked by the target and detected through target detection with the Crazyflie then moving to this location to land - in essence, this is already being performed since the Crazyflie is landing at the final location of the target.

Thus, this overall implementation serves as a fairly successful proof of concept for target detection and tracking in three-dimensions using a single camera and hardware with marginal computational effort. With expansion, the possible uses include indoor sports arenas, storage warehouses, manufacturing or assembly workshops, and other applications in buildings requiring target tracking, surveillance, or monitoring - although it is recommended for the quadrotor to frequently return to the initial origin to recalibrate and account for any accumulation of error in position. Finally, despite a one-to-one mapping being pursued where the Crazyflie directly mirrored the target, it is also easily adaptable to use a different mapping, such as one-to-two, where the Crazyflie moves twice the distance of the target, or inverse mapping, where the Crazyflie moves in the opposite direction to the target.

8 CONCLUSIONS

An investigation into position-based visual servoing through end-point open-loop control was conducted for estimation of the three-dimensional position and yaw orientation of a moving target using a single camera, where a Crazyflie then tracked this position and yaw orientation autonomously. Additionally, inexpensive and low-end hardware with marginal computational effort was used in the form of a Raspberry Pi and Pi Camera for lightweight image processing with OpenCV. From the implementation, the following conclusions can be summarised in relation to the objectives:

- The pinhole camera model is valid and can be successfully implemented using the Pi Camera at a resolution of 160px by 120px which results in a focal length of approximately 113.5px. There were no major distortions experienced and compensation was not required.
- Comparing the resolutions of 160px by 120px, 320px by 240px, and 640px by 480px, it was found that the most satisfactory resolution was 160px by 120px with the capability to generate setpoints at a frequency of 30.1Hz. Moreover, when considering the minimum processing required, it is evident that the grayscale processing allows for an increased frame rate compared to colour processing by an average actual percentage difference of 23.2% across the resolutions of 160px by 120px, 320px by 240px, and 640px by 480px (this equivalently corresponds with an average actual percentage difference of 23.2% in the number of setpoints generated), while also eliminating more background noise for a more accurate interpretation of the target.
- Through calibrating the Pi Camera based on the pinhole camera model, the three-dimensional position of the target relative to the camera can be successfully described through computer vision techniques for target detection with the centroid of the target providing latitude and longitude coordinates and the area of the target providing the altitude of the camera. The yaw orientation can also be estimated based on the area of the target when using a disproportioned target. End-point open-loop visual servoing can then be executed, such that the Crazyflie is controlled to remotely mirror the translational and rotational motion of the target and maintain the altitude of the camera by generating setpoints on the Raspberry Pi with transmission over the Crazyradio.
- The real-time effectiveness of the end-point open-loop visual servoing exhibited an average lag of 0.732s, average position deviations of 83.4mm, minimum yaw orientation deviation of -9.54^{o}, and maximum yaw orientation deviation of 9.87^{o}. However, if the average lag is compensated in each test, the average position deviation reduces to only 54.0mm, while the minimum and maximum yaw orientation deviations reduce to only -4.76^{o} and 4.46^{o} respectively. An altitude could also be maintained fairly successfully, where the Crazyflie only drifted by 3.75% downwards or 3.98% upwards on average before returning to the correct altitude.

9 RECOMMENDATIONS

The following points are recommended for further research and development:

- Although it did not affect the results and conclusions, it may be helpful to revise the take off function in the control script for smoother behaviour if further testing is to be performed.
- The tests should be repeated in a larger environment to gauge the capability for expansion. This will involve calibrating the camera for different altitudes and possibly using a larger target.

- Additional tests should also be performed in a variety of uncontrolled environments which contain minor obstacles, such as outdoors with effects on the flight control from inconsistent wind and effects on the target detection from a greater variation in background colours and textures.
- The control script can be adapted to function with multiple targets and quadrotors, where each target is uniquely identified and assigned to a specific quadrotor to follow the motion.
- The target could move at higher speeds to highlight the effect this will have on the lag and whether the computational effort of the Raspberry Pi has a direct influence in this situation.
- For faster execution, the commands for the Crazyflie and image processing in OpenCV could be rewritten and performed natively in C and C++, which are generally considered to require less computational time than Python since C and C++ are compiled while Python is interpreted.
- An infra-red camera, as is traditional for motion capture systems, can be investigated for use in darker environments or when there is noise from changing lighting. The Raspberry Pi NoIR Camera Module 2.1 is a suitable infra-red camera and is compatible with the current setup.
- Using a larger quadrotor, the camera can be mounted and end-point closed-loop could be performed with the quadrotor directly tracking the target. This was partially developed to complement the completed research, as documented in Appendix E, but it was not possible to get the quadrotor to fly safely after several attempts and a reasonable amount of effort.
- Finally, there is an opportunity to research artificial intelligence and deep learning algorithms, such as a convolutional neural network, for more controlled and accurate results compared to the pinhole camera model, although this will likely require greater computational effort.

REFERENCES

[1] M. Rabah et al. 'Autonomous Moving Target-Tracking For A UAV Quadcopter Based On Fuzzy-PI'. In: *Institute of Electrical and Electronics Engineers (IEEE) Access* Vol. 7 (Apr. 2019), pp. 38407–38419. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2906345. URL: https://ieeexplore.ieee.org/document/8672092.

[2] O. Dunkley. *Visual Inertial Control Of A Nano-Quadrotor*. Technical University Munich, Sept. 2014. URL: https://vision.in.tum.de/_media/spezial/bib/dunkley14msc.pdf.

[3] R. Mahony, V. Kumar and P. Corke. 'Multirotor Aerial Vehicles: Modelling, Estimation, And Control'. In: *Robotics And Automation Magazine, Institute of Electrical and Electronics Engineers (IEEE)* Vol. 19. (3) (Sept. 2012), pp. 20–32. ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2206474. URL: https://ieeexplore.ieee.org/document/6289431.

[4] F. Poirier et al. *Control Of Crazyflie Quadcopter*. University of Manchester, Manchester, Sept. 2018. URL: https://www.ensta-bretagne.fr/jaulin/rapport2018_poirier.pdf.

[5] M. G. Popova. *Visual Servoing For A Quadrotor UAV In Target Tracking*. University of Toronto, Nov. 2015. URL: https://tspace.library.utoronto.ca/handle/1807/70520.

[6] C. Karlsson. *Vision Based Control And Landing Of Micro Aerial Vehicles*. Karlstad University, June 2019. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-73225.

[7] A. Kendall, N. Salvapantula and K. Stol. 'On-Board Object Tracking Control Of A Quadcopter With Monocular Vision'. In: *International Conference on Unmanned Aircraft Systems (ICUAS), Institute of Electrical and Electronics Engineers (IEEE)* (Apr. 2014), pp. 404–411. DOI: 10.1109/ICUAS.2014.6842280. URL: https://ieeexplore.ieee.org/document/6842280 or https://alexgkendall.com/media/papers/ICUAS_Kendall_2014.pdf.

[8] J. Förster. *System Identification Of The Crazyflie 2.0 Nano Quadrocopter*. Swiss Federal Institute Of Technology, Zurich, Aug. 2015. DOI: 10.3929/ethz-b-000214143. URL: https://www.research-collection.ethz.ch/handle/20.500.11850/214143.

[9] L. Adrjanowicz, M. Kubanek and J. Bobulski. 'Single Camera Based Location Estimation With Dissimilarity Measurement'. In: *2013 6th International Conference On Human System Interactions (HSI)* (June 2013), pp. 241–246. DOI: 10.1109/HSI.2013.6577830. URL: https://ieeexplore.ieee.org/document/6577830.

[10] B. Mack, C. Noe and T. Rice. *Formation Control Of Crazyflies*. Bradley University, Peoria, May 2018. URL: http://cegt201.bradley.edu/projects/proj2018/crazy/resources/bu_crazyflie_final_report.pdf and http://cegt201.bradley.edu/projects/proj2018/crazy/deliverables/deliverables.html.

[11] L. O. G. Lobo E Silva. *Attitude Control Of A Quadrotor*. Ministerio Da Defesa Instituto Militar De Engenharia (IME), Rio de Janeiro, Sept. 2016. URL: http://mecatrime.org/wp-content/uploads/2017/05/Luis_mestrado_dissertacao_final.pdf.

[12] C.-K. Liang, L.-W. Chang and H. H. Chen. 'Analysis And Compensation Of Rolling Shutter Effect'. In: *Institute of Electrical and Electronics Engineers (IEEE) Transactions On Image Processing* Vol. 17. (8) (Aug. 2008), pp. 1323–1330. DOI: 10.1109/TIP.2008.925384. URL: https://ieeexplore.ieee.org/document/4549748.

[13] OpenCV and Doxygen. *OpenCV 4.1.1 Documentation Modules - Python Tutorials*. Open Source Computer Vision. Online. 26th July 2019. URL: https://docs.opencv.org/4.1.1/d6/d00/tutorial_py_root.html (visited 18th Sept. 2019).

[14] K. V. Patil. *Object Tracking Using A Quadcopter*. Indian Institude of Technology (IIT), Bombay, July 2015. URL: https://www.ee.iitb.ac.in/student/~kalpesh.patil/material/navstik_report.pdf.

[15] J. Pedro. *MECN4029 - Mechatronics II - Lecture Series And Notes*. University of The Witwatersrand, Johannesburg, 2019.

[16] W. Hönig and N. Ayanian. 'Flying Multiple UAVs Using ROS'. In: *Robot Operating System (ROS): The Complete Reference*. Ed. by A. Koubaa. Vol. 2. Springer International Publishing, Cham, May 2017, pp. 83–118. ISBN: 978-3-319-54927-9. DOI: 10.1007/978-3-319-54927-9_3. URL: https://link.springer.com/chapter/10.1007%2F978-3-319-54927-9_3 or http://act.usc.edu/publications/Hoenig_Springer_ROS2017.pdf.

[17] K. Richardsson et al. *Crazyflie 2.1*. Bitcraze. Online. 2019. URL: https://store.bitcraze.io/products/crazyflie-2-1 (visited 16th Sept. 2019).

[18] T. Antonsson, M. Eliasson and A. Taffanel. *Crazyflie 2.1 Control Board Schematic*. Bitcraze, Dec. 2018. URL: https://wiki.bitcraze.io/_media/projects:crazyflie2:hardware:cf2.1_component_placement.pdf.

[19] K. Richardsson et al. *Flow Deck V2*. Bitcraze. Online. 2019. URL: https://store.bitcraze.io/collections/decks/products/flow-deck-v2 (visited 12 Oct. 2019).

[20] Raspberry Pi Foundation. *Raspberry Pi Zero W*. Raspberry Pi. Online. 2019. URL: https://www.raspberrypi.org/products/raspberry-pi-zero-w/ (visited 16th Sept. 2019).

[21] J. Hughes, L. Lynch, B. Nuttall, and Raspberry Pi Foundation. *Documentation - Hardware - Camera Module*. Raspberry Pi. Online. 24th April 2018. URL: https://www.raspberrypi.org/documentation/hardware/camera/ (visited 16th Sept. 2019).

[22] R. Thornton. *Raspberry Pi Zero W V1.3 - Mechanical Drawings*. Raspberry Pi Foundation, July 2019. URL: https://www.raspberrypi.org/documentation/hardware/raspberrypi/mechanical/rpi_MECH_Zero_1p3.pdf.

[23] M. Stimson and J. Adams. *Raspberry Pi Camera Module V2.1 - Mechanical Drawings*. Raspberry Pi Foundation, Nov. 2015. URL: https://www.raspberrypi.org/documentation/hardware/camera/mechanical/rpi_MECH_Camera2_2p1.pdf.

[24] K. Richardsson et al. *Crazyradio PA USB*. Bitcraze. Online. 2019. URL: https://store.bitcraze.io/collections/kits/products/crazyradio-pa (visited 16th Sept. 2019).

[25] A. Taffanel. *Crazyflie Communication*. Bitcraze. Online. Jan. 2015. URL: https://www.bitcraze.io/2015/01/crazyflie-2-0-radio-communication/ (visited 22nd Oct. 2019).

[26] DaguRobot Electronics. *S4A EDU Controller (Scratch For Arduino) ROHS*. Online. 2019. URL: http://www.dagurobot.com/S4A_EDU_controller (visited 28th Sept. 2019).

[27] DaguRobot Electronics. *DAGU Off Set Gear Motor Accessories 1:48 (1 Pair) 9mm Long Output Axis ROHS*. Online. 2019. URL: http://www.dagurobot.com/motor_acessorios_1:48_9MM_long_output_axis (visited 28th Sept. 2019).

[28] Qualisys. *Miqus Specifications, Capture More With Less*. Qualisys, Motion Capture Systems, Oct. 2018. URL: https://cdn-content.qualisys.com/2018/10/PI_Miqus.pdf.

[29] PiShop. *Raspberry Pi Zero W 1.1*. Online. 2019. URL: https://www.pishop.co.za/store/raspberry-pi/raspberry-pi-zero-wireless (visited 23rd Oct. 2019).

[30] T. Antonsson, M. Eliasson and A. Taffanel. *BigQuad Expansion Deck*. Bitcraze Wiki. Online. 11th December 2018. URL: https://wiki.bitcraze.io/projects:crazyflie2:expansionboards:bigquad (visited 14th Sept. 2019).

[31] T. Antonsson, M. Eliasson and A. Taffanel. *BigQuad Expansion Deck Schematic*. Bitcraze, Dec. 2018. URL: https://wiki.bitcraze.io/_media/projects:crazyflie2:expansionboards:bigquad-revc-top.png.

A ADDITIONAL RESOURCES

For supplement to the presented data, online resources containing additional data which are not of an appropriate form for presentation in a report are available at: `https://drive.google.com/open?id=1wsVvasJ_MfNI8SphguZh88tsmAFaG_MC` (access is limited to accounts linked to the University of the Witwatersrand unless permission is requested). Since the Bitcraze Crazyflie 2.1, Bitcraze Flow V2 expansion deck, Bitcraze Crazyradio PA, Raspberry Pi Zero W 1.1, and Raspberry Pi Camera Module 2.1 are open-source, the electronic schematics for the hardware are freely available and have been compiled with these other online resources if reference is desired.

B CONTROL SCRIPTS

Firstly, the installation of the required packages is as follows for Raspbian (and other similar Debian-based linux distributions with the Apt 1.8.2 package manager) before utilising the Crazyflie Python API library, Crazyflie graphical client, and OpenCV Python API library. (The Raspberry Pi must be booted with the Crazyradio inserted otherwise it will not be detected correctly).

```
sudo apt install python3
sudo apt install python3-pip
sudo apt install python3-libusb1
sudo apt install python3-pyqt5
pip3 install numpy # sudo apt install python3-numpy
pip3 install opencv-python # sudo apt install python3-opencv
pip3 install cflib
pip3 install PyUSB
pip3 install appdirs
pip3 install PyYAML
```

The subsequent Python script with OpenCV was used to evaluate the real-time performance for image processing by comparing resolutions of 160px by 120px, 320px by 240px, and 640px by 480px with both grayscale and colour processing. The presented form of the code is for the maximum grayscale processing - for the other processing tests, the relevant lines marked with ## must be uncommented. As discussed, a resolution around 160px by 120px was only suitable with the Raspberry Pi.

```
# ********************************************************************************
# Date: 2019-09-30. Author: Edward Rycroft. Description: This program is used to
> evaluate the grayscale and colour image processing at different resolutions.
# ********************************************************************************

# --------------------------------------------------------------------------------
# Import Files
# ------------------------------------------------------------------------------------

# Import the necessary libraries (note the changes of names).
import time
import numpy as np
import cv2 as cv

# ------------------------------------------------------------------------------------
# Target Detection Functions
# ------------------------------------------------------------------------------------

# Function to perform grayscale image processing.
def processing_gray (frame):

    # Convert the frame to grayscale, apply a threshold or mask with the limit
    > tuned for the target, and apply an open morphology transformation.
    mode = cv.cvtColor (frame, cv.COLOR_BGR2GRAY)
    #mode = cv.blur (mode, (5, 5))
    ret, threshold = cv.threshold (mode, 150, 255, cv.THRESH_BINARY)
    kernel = np.ones ((5, 5), np.uint8)
    morphology = cv.morphologyEx (threshold, cv.MORPH_OPEN, kernel)

    # Return the relevant variables to store.
    return frame, mode, threshold, morphology

# Function to perform colour image processing.
def processing_colour (frame):

    # Convert the frame to hue-lightness-saturation, apply a threshold or mask with
    > the limits tuned for the target, and apply an open morphology transformation.
    mode = cv.cvtColor (frame, cv.COLOR_BGR2HLS)
    #mode = cv.blur (mode, (5, 5))
    lower_colour = np.array ([150, 40, 40])
    upper_colour = np.array ([210, 250, 250])
    threshold = cv.inRange (mode, lower_colour, upper_colour)
    kernel = np.ones ((5, 5), np.uint8)
    morphology = cv.morphologyEx (threshold, cv.MORPH_OPEN, kernel)

    # Return the relevant variables to store.
    return frame, mode, threshold, morphology

# Function to detect the centroid of the target.
def target_detection (frame, processed):

    # Find and identify the contours within the current frame.
    image, contours, hierarchy = cv.findContours (threshold, cv.RETR_TREE,
    > cv.CHAIN_APPROX_SIMPLE) # This line should be used for OpenCV Python 3.X.
    #contours, hierarchy = cv.findContours (processed, cv.RETR_TREE,
    > cv.CHAIN_APPROX_SIMPLE) # This line should be used for OpenCV Python 4.X.
    #temporary = cv2.findContours (des, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE) #
    > With the line below, this line could be used for OpenCV Python 3.X or 4.X.
    #contours = temporary [0] if len (temporary) == 2 else temporary [1]

    # Locate the largest contour and find the corresponding properties. Note that
    > the image moments are simply weighted averages based on the pixel intensities.
    if len (contours) != 0:
        maximum = max (contours, key = cv.contourArea)
        moment = cv.moments (maximum)
        centroid_x = int (moment ["m10"] / moment ["m00"])
        centroid_y = int (moment ["m01"] / moment ["m00"])
        x, y, w, h = cv.boundingRect (maximum)
        angled = cv.minAreaRect (maximum)
        box = np.int0 (cv.boxPoints (angled))
        cv.rectangle (frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv.drawContours (frame, [box], 0, (0, 0, 255), 2)
        cv.circle (frame, (centroid_x, centroid_y), 3, (255, 0, 0), 5)
    else:
        centroid_x = 0
        centroid_y = 0

    # Draw a coordinate system and timestamp to the current frame.
    time = str (np.around (time.time () - time_start, decimals = 1))
    cv.putText (frame, time, (10, 30), cv.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2)
    cv.circle (frame, (int (res_x / 2), int (res_y / 2)), 5, (255, 255, 255), 5)

    # Return the relevant variables to store.
    return frame, contours, centroid_x, centroid_y

# Function to record and display the basic output as a video.
def record_minimum (frame, record):

    # Record (optional) and display the basic output.
    if record == "Y":
        video.write (frame)
    cv.imshow ("Original Output", frame)

# Function to record and display the complete output as a video.
def record_maximum (frame, mode, threshold, morphology, record, contours):

    # Combine the processed frames into a single frame.
    if np.size (mode) == res_x * res_y:
        mode = cv.merge ([mode, mode, mode])
    elif np.size (mode) == res_x * res_y * 3:
        mode = cv.cvtColor (mode, cv.COLOR_HLS2BGR)
        mode = cv.bitwise_and (mode, mode, mask = morphology)
    threshold = cv.merge ([threshold, threshold, threshold])
    morphology = cv.merge ([morphology, morphology, morphology])
    if len (contours) != 0:
        cv.drawContours (morphology, contours, -1, (255, 0, 0), 2)
    output = np.zeros ((res_y * 2, res_x * 2, 3), dtype = "uint8")
    output [0:res_y, 0:res_x] = frame
    output [0:res_y, res_x:(res_x * 2)] = mode
    output [res_y:(res_y * 2), 0:res_x] = threshold
    output [res_y:(res_y * 2), res_x:(res_x * 2)] = morphology

    # Record (optional) and display the combined output.
    if record == "Y":
        video.write (output)
    cv.imshow ("Combined Output", output)

# ----------------------------------------------------------------------------------
# Define Variables
# ----------------------------------------------------------------------------------

# Set the variables for target detections.
print ("Setting the variables to be used.")
res_x = 160 # 320 # 640
res_y = 120 # 240 # 480
fps = 30

# Find basic requirements from the user.
record = input ("Must the source be recorded as a video? [Y/N] ")
if record == "Y":
    version = input ("Must basic [B] or combined [C] output be recorded? [B/C] ")
    if version != "B" and version != "C":
        print ("! The input for the version is invalid. Quitting the program.")
        raise SystemExit
elif record != "N":
    print ("! The input to record the source is invalid. Quitting the program.")
    raise SystemExit

# ----------------------------------------------------------------------------------
# Perform Checks
# ------------------------------------------------------------------------------------

# Identify the source for use and recording if desired.
print ("Identifying the source for use and recording if desired.")
source = cv.VideoCapture (0)
source.set (cv.CAP_PROP_FRAME_WIDTH, res_x) # Alternate identifier: 3.
source.set (cv.CAP_PROP_FRAME_HEIGHT, res_y) # Alternate identifier: 4.
if record == "Y":
    codec = cv.VideoWriter_fourcc (*"XVID")
    if version == "B":
        video = cv.VideoWriter ("Out.avi", codec, fps, (res_x, res_y), True)
    elif version == "C":
        video = cv.VideoWriter ("Out.avi", codec, fps, (res_x * 2, res_y * 2), True)

# Ensure the source is open and available.
active = True
strikes = 0
if source.isOpened == False:
    print ("! The source is not open. Trying to open the source.")
    source.Open ()
    if source.isOpened == False:
        print ("The source is not open.")
        active == False
elif source.isOpened == True:
    print ("The source is open and available.")

# Check that the source is operating correctly.
print ("Checking that the source is operating correctly.")
check = input ("Should the source be checked to be capturing correctly? [Y/N] ")
if check == "Y":
    correct, frame = source.read ()
    if correct == False:
        print ("A frame could not be read correctly.")
        correct = "N"
    elif correct == True:
        cv.imshow ("Image Test", frame)
        cv.waitKey (1)
        correct = input ("Is the capture displayed correctly? [Y/N] ")
    if correct == "N":
        print ("! The capture is not operating correctly.")
        active = False
    cv.destroyAllWindows ()

```

```
# ------------------------------------------------------------------------------------
# Main Project Loop
# ------------------------------------------------------------------------------------

# Begin capture of the video frames and image processing.
print ("Beginning capture of the video frames and image processing.")
print ("To terminate the capture, press [Q] in the video window.")
time_start = time.time ()
while active == True:

    # Load the current frame from the source.
    correct, frame = source.read ()

    # A: No processing with only basic video capture.
    ##record_minimum (frame, record)

    # B: Minimum grayscale processing with results video capture.
    ##frame, mode, threshold, morphology = processing_gray (frame)
    ##frame, contours, centroid_x, centroid_y = target_detection (frame, morphology)
    ##record_minimum (frame, record)

    # C: Maximum grayscale processing with complete video capture.
    frame, mode, threshold, morphology = processing_gray (frame)
    frame, contours, centroid_x, centroid_y = target_detection (frame, morphology)
    record_maximum (frame, mode, threshold, morphology, record, contours)

    # D: Minimum colour processing with results video capture.
    ##frame, mode, threshold, morphology = processing_colour (frame)
    ##frame, contours, centroid_x, centroid_y = target_detection (frame, morphology)
    ##record_minimum (frame, record)

    # E: Maximum colour processing with complete video capture.
    ##frame, mode, threshold, morphology = processing_colour (frame)
    ##frame, contours, centroid_x, centroid_y = target_detection (frame, morphology)
    ##record_maximum (frame, mode, threshold, morphology, record, contours)

    # Look for the designated key to terminate the capture.
    key = cv.waitKey (1)
    if key == ord ("Q"):
        active = False

time_end = time.time ()

# ------------------------------------------------------------------------------------
# Finishing Commands
# ---------------------------------------------------------------------------------------

# Terminate the capture and release the source.
print ("Terminating the capture and releasing the source.")
source.release ()
if record == "Y":
    print ("The output video is 'Out.avi' in the current directory.")
    video.release ()
cv.destroyAllWindows ()

# Review the details of the captured output.
if record == "Y":
    print ("Review of the details of the capturing process:")
    video = cv.VideoCapture ("Out.avi")
    time_record = (time_end - time_start)
    total_frames = int (video.get (cv.CAP_PROP_FRAME_COUNT))
    print ("   Horizontal Resolution:", res_x, "pixels.")
    print ("   Vertical Resolution:", res_y, "pixels.")
    print ("   Total Recording Time:", "{0:.2f}".format (time_record), "seconds.")
    print ("   Total Frames:", total_frames, "frames.")
    try:
        fps_avg = total_frames / time_record
        spf_avg = 1 / fps_avg
        print ("   Average FPS: ", "{0:.2f}".format (fps_avg), "frames per second.")
        print ("   Average SPF:", "{0:.4f}".format (spf_avg), "seconds per frame.")
    except ZeroDivisionError:
        print ("   ! The resolution passed for capture does not match the output.")
        print ("   ! The average frame rate values could not be determined.")
        fps_avg = 0
        spf_avg = 0

# ---------------------------------------------------------------------------------------
# End
# ---------------------------------------------------------------------------------------
```

For altitude calibration, the above script was modified with `print(cv.contourArea(maximum))` within the `target_detection` function to display the target area in pixels, while maximum grayscale processing monitored the target as it was positioned and the altitude of the camera was varied. As mentioned, this printing to the shell dramatically slowed execution with a noticeable lag.

The final script for target detection and tracking is presented below, where further optimisations have also been performed after the script for the resolution tests and altitude calibration to make the code more efficient and robust. An optimisation which was not exploited, as it was not necessary, is the idea to restrict the region in which to look for the target based on the detected location of the target in the previous frame, such that even less computational effort would be required as a result.

```
# ******************************************************************************
# Date: 2019-18-10. Author: Edward Rycroft. Description: This program is used to
> perform visual servoing through end-point open-loop control using a Raspberry Pi
> and Pi Camera for image processing with OpenCV and a Crazyflie to mirror the
> relative position of a moving target for effective target detection and tracking.
# ******************************************************************************

# ------------------------------------------------------------------------------
# Import Files
# ------------------------------------------------------------------------------

# Import the necessary libraries (note the changes of names).
import time
import logging
import numpy as np
import cv2 as cv
import cflib.crtp
from cflib.crazyflie import Crazyflie
from cflib.crazyflie.log import LogConfig
from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
from cflib.crazyflie.syncLogger import SyncLogger

# ------------------------------------------------------------------------------
# Target Detection And Tracking Functions
# ------------------------------------------------------------------------------

# Function to perform grayscale image processing.
def processing_gray (frame):

    # Convert the frame to grayscale, apply a threshold or mask with the limit
    > tuned for the target, and apply an open morphology transformation.
    mode = cv.cvtColor (frame, cv.COLOR_BGR2GRAY)
    #mode = cv.blur (mode, (5, 5))
    ret, threshold = cv.threshold (mode, 200, 255, cv.THRESH_BINARY)
    kernel = np.ones ((5, 5), np.uint8)
    morphology = cv.morphologyEx (threshold, cv.MORPH_OPEN, kernel)

    # Return the relevant variables to store.
    return frame, mode, threshold, morphology
```

```

# Function to perform colour image processing.
def processing_colour (frame):

    # Convert the frame to hue-lightness-saturation, apply a threshold or mask with
    > the limits tuned for the target, and apply an open morphology transformation.
    mode = cv.cvtColor (frame, cv.COLOR_BGR2HLS)
    #mode = cv.blur (mode, (5, 5))
    lower_colour = np.array ([150, 40, 40])
    upper_colour = np.array ([210, 250, 250])
    threshold = cv.inRange (mode, lower_colour, upper_colour)
    kernel = np.ones ((5, 5), np.uint8)
    morphology = cv.morphologyEx (threshold, cv.MORPH_OPEN, kernel)

    # Return the relevant variables to store.
    return frame, mode, threshold, morphology

# Function to detect the centroid and area of the target.
def target_detection (frame, morphology, time_start, res_x, res_y):

    # Find and identify the contours within the current frame.
    image, contours, hierarchy = cv.findContours (threshold, cv.RETR_TREE,
    > cv.CHAIN_APPROX_SIMPLE) # This line should be used for OpenCV Python 3.X.
    #contours, hierarchy = cv.findContours (processed, cv.RETR_TREE,
    > cv.CHAIN_APPROX_SIMPLE) # This line should be used for OpenCV Python 4.X.
    #temporary = cv2.findContours (des, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE) #
    > With the line below, this line could be used for OpenCV Python 3.X or 4.X.
    #contours = temporary [0] if len (temporary) == 2 else temporary [1]

    # Locate the largest contour and find the corresponding properties.
    try:
        maximum = max (contours, key = cv.contourArea)
        moment = cv.moments (maximum)
        centroid_x = int (moment ["m10"] / moment ["m00"])
        centroid_y = int (moment ["m01"] / moment ["m00"])
        area = cv.contourArea (maximum)
        x, y, w, h = cv.boundingRect (maximum)
        cv.rectangle (frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv.circle (frame, (centroid_x, centroid_y), 3, (255, 0, 0), cv.FILLED)
        area = cv.contourArea (maximum)
        transform_x = int (res_x / 2) - centroid_y
        transform_y = int (res_y / 2) - centroid_x
    except:
        area = 1
```

```
            transform_x = 0
            transform_y = 0

    # Draw a coordinate system and timestamp to the current frame.
    time_current = np.around (time.time () - time_start, decimals = 1)
    cv.putText (frame, str (time_current), (10, 30), cv.FONT_HERSHEY_SIMPLEX, 1,
    > (255, 255, 255), 2) # (10, 30) = Top-Left Corner
    cv.circle (frame, (int (res_x/2), int (res_y/2)), 5, (255, 255, 255), cv.FILLED)

    # Return the relevant variables to store.
    return frame, contours, transform_x, transform_y, area

# Function to detect the yaw orientation of the target.
def target_detection_orientation (frame, contours):

    # Locate the largest contour and find the corresponding properties. Note that
    > the image moments are simply weighted averages based on the pixel intensities.
    try:
        maximum = max (contours, key = cv.contourArea)
        x, y, w, h = cv.boundingRect (maximum)
        bounds = cv.minAreaRect (maximum)
        orientation = bounds [2]
        if bounds [1][0] < bounds [1][1]: # width < height
            orientation = - orientation
        else:
            orientation = - orientation - 90
        box = np.int0 (cv.boxPoints (bounds))
        cv.drawContours (frame, [box], 0, (0, 0, 255), 2)
    except:
        orientation = 0

    # Return the relevant variables to store.
    return frame, orientation

# Function to record and display the basic output as a video.
def monitor_minimum (frame, record):

    # Record (optional) and display the basic output.
    if record == "Y":
        video.write (frame)
    cv.imshow ("Original Output", frame)

# Function to record and display the complete output as a video.
def monitor_maximum (frame, mode, threshold, morphology, record, contours):

    # Combine the processed frames into a single frame.
    if np.size (mode) == res_x * res_y:
        mode = cv.merge ([mode, mode, mode])
    elif np.size (mode) == res_x * res_y * 3:
        mode = cv.cvtColor (mode, cv.COLOR_HLS2BGR)
        mode = cv.bitwise_and (mode, mode, mask = morphology)
    threshold = cv.merge ([threshold, threshold, threshold])
    morphology = cv.merge ([morphology, morphology, morphology])
    if len (contours) != 0:
        cv.drawContours (morphology, contours, -1, (255, 0, 0), 2) # cv.FILLED
    output = np.zeros ((res_y * 2, res_x * 2, 3), dtype = "uint8")
    output [0:res_y, 0:res_x] = frame
    output [0:res_y, res_x:(res_x * 2)] = mode
    output [res_y:(res_y * 2), 0:res_x] = threshold
    output [res_y:(res_y * 2), res_x:(res_x * 2)] = morphology

    # Record (optional) and display the combined output.
    if record == "Y":
        video.write (output)
    cv.imshow ("Combined Output", output)

# Function to print the characteristics of the target centroid.
def monitor_text (centroid_x, centroid_y, altitude, orientation):

    # Print the target characteristics to the shell.
    print ("Centroid X: ", centroid_x)
    print ("Centroid Y: ", centroid_y)
    print ("Altitude: ", "{0:.2f}".format (altitude))
    print ("Orientation: ", "{0:.2f}".format (orientation))
    print ("")

# ---------------------------------------------------------------------------------
# Crazyflie Control Functions
# ---------------------------------------------------------------------------------

# Function to update the position estimator after the extend Kalman filter is reset.
def wait_for_position_estimator (crazyflie):

    # Print an update to the shell for monitoring.
    print ("Waiting for estimator to calibrate the current state.")

    # Set up logging and add the variables to log for updating.
    log_config = LogConfig (name = "Kalman Variance", period_in_ms = 500)
    log_config.add_variable ("kalman.varPX", "float")
    log_config.add_variable ("kalman.varPY", "float")
    log_config.add_variable ("kalman.varPZ", "float")
    var_y_history = [1000] * 10
    var_x_history = [1000] * 10
    var_z_history = [1000] * 10

    # Define a threshold for maximum and minimum variations.
    threshold = 0.001

    # Begin logging the added variables.
    with SyncLogger (crazyflie, log_config) as logger:
        for log_entry in logger:

            # Append the current values to the stored history.
            data = log_entry [1]
            var_x_history.append (data ["kalman.varPX"])
            var_x_history.pop (0)
            var_y_history.append (data ["kalman.varPY"])
            var_y_history.pop (0)
            var_z_history.append (data ["kalman.varPZ"])
            var_z_history.pop (0)

            # Compare the maximum and minimum variations until within the threshold.
            minimum_x = min (var_x_history)
            maximum_x = max (var_x_history)
            minimum_y = min (var_y_history)
            maximum_y = max (var_y_history)
            minimum_z = min (var_z_history)
            maximum_z = max (var_z_history)
            if (maximum_x - minimum_x) < threshold and (maximum_y - minimum_y) <
            > threshold and (maximum_z - minimum_z) < threshold:
                break

# Function to set a custom initial position.
def set_initial_position (crazyflie, initial_x, initial_y, initial_z, initial_yaw):

    # Print an update to the shell for monitoring.
    print ("Setting the initial position to (", initial_x, ",", initial_y, ",",
    > initial_z, ") at ", initial_yaw, " radians.")

    # Setting the initial position.
    crazyflie.param.set_value ("kalman.initialX", initial_x)
    crazyflie.param.set_value ("kalman.initialY", initial_y)
    crazyflie.param.set_value ("kalman.initialZ", initial_z)
    crazyflie.param.set_value ("kalman.initialYaw", initial_yaw)

# Function to set the initial position as the origin.
def set_initial_position_origin (crazyflie):

    # Print an update to the shell for monitoring.
    print ("Setting the initial position as the origin (0,0,0) at 0 radians.")

    # Setting the initial position.
    crazyflie.param.set_value ("kalman.initialX", "0")
    crazyflie.param.set_value ("kalman.initialY", "0")
    crazyflie.param.set_value ("kalman.initialZ", "0")
    crazyflie.param.set_value ("kalman.initialYaw", "0")

# Function to reset the extended Kalman filer.
def reset_estimator (crazyflie):

    # Print an update to the shell for monitoring.
    print ("Resetting the extended Kalman filter for a clean state.")

    # Reset the extended Kalman filter and update the position estimator.
    crazyflie.param.set_value ("kalman.resetEstimation", "1")
    time.sleep (0.1)
    crazyflie.param.set_value ("kalman.resetEstimation", "0")
    wait_for_position_estimator (crazyflie)

# Function to vertically take off the Crazyflie.
def take_off (crazyflie, altitude):

    # Print an update to the shell for monitoring.
    print ("Crazyflie is taking off to the starting altitude.")

    # Define the desired time in which to move and the number of setpoints to send.
    action_time = 1
    setpoint_time = 0.05
    steps = int (action_time / setpoint_time)

    # Loop to send the required setpoint at the desired time intervals.
    for i in range (steps):
        crazyflie.commander.send_hover_setpoint (0, 0, (altitude / steps * i), 0)
        time.sleep (setpoint_time)

    # Ensure the Crazyflie has reached the desired altitude.
    crazyflie.commander.send_hover_setpoint (0, 0, 0, altitude)
    time.sleep (setpoint_time)

# Function to vertically land the Crazyflie.
def land (crazyflie, altitude):

    # Print an update to the shell for monitoring.
    print ("Crazyflie is landing from the current altitude.")

    # Define the desired time in which to move and the number of setpoints to send.
    action_time = 1
    setpoint_time = 0.05
    steps = int (action_time / setpoint_time)

    # Loop to send the required setpoint at the desired time intervals.
    for i in range (steps):
        crazyflie.commander.send_hover_setpoint (0, 0, (- altitude / steps * i), 0)
        time.sleep (setpoint_time)

    # Ensure the Crazyflie has landed and becomes stationary.
    crazyflie.commander.send_stop_setpoint ()
    time.sleep (setpoint_time)

# ---------------------------------------------------------------------------------
# Define Variables
# ---------------------------------------------------------------------------------

# Set the variables for target detection and tracking.
print ("Setting the variables to be used.")
res_x = 160
res_y = 120
fps = 30
time_start = 0

# Set the variables for Crazyflie communication control.
uri = "radio://0/80/2M/E7E7E7E7E8"
logging.basicConfig (level = logging.ERROR)
crazyflie = Crazyflie (rw_cache = "./cache")

# Set the variables for the initial setup before tracking.
altitude = 0.4
yaw = 0
#velocity_max = 0.4
focal_length = 113.5
```

```python

# Find basic requirements from the user.
record = input ("Must the source be recorded as a video? [Y/N] ")
if record == "Y":
    version = input ("Must basic [B] or combined [C] output be recorded? [B/C] ")
    if version != "B" and version != "C":
        print ("! The input for the recording version is invalid.")
        raise SystemExit
elif record != "N":
    print ("! The input for the recording of the source is invalid.")
    raise SystemExit

# ---------------------------------------------------------------------------------
# Perform Checks
# ---------------------------------------------------------------------------------

# Identify the source for use and recording if desired.
print ("Identifying the source for use and recording if desired.")
source = cv.VideoCapture (0)
source.set (cv.CAP_PROP_FRAME_WIDTH, res_x) # Alternate identifier: 3.
source.set (cv.CAP_PROP_FRAME_HEIGHT, res_y) # Alternate identifier: 4.
if record == "Y":
    codec = cv.VideoWriter_fourcc (*"XVID")
    if version == "B":
        video = cv.VideoWriter ("Out.avi", codec, fps, (res_x, res_y), True)
    elif version == "C":
        video = cv.VideoWriter ("Out.avi", codec, fps, (res_x * 2, res_y * 2), True)

# Ensure the source is open and available.
active = True
strikes = 0
if source.isOpened == False:
    print ("! The source is not open. Trying to open the source.")
    source.Open ()
    if source.isOpened == False:
        print ("The source is not open.")
        active == False
elif source.isOpened == True:
    print ("The source is open and available.")

# Check that the source is operating correctly.
print ("Checking that the source is operating correctly.")
check = input ("Should the source be checked to be capturing correctly? [Y/N] ")
if check == "Y":
    correct, frame = source.read ()
    if correct == False:
        print ("A frame could not be read correctly.")
        correct = "N"
    elif correct == True:
        cv.imshow ("Image Test", frame)
        cv.waitKey (1)
        correct = input ("Is the capture displayed correctly? [Y/N] ")
    if correct == "N":
        print ("! The capture is not operating correctly.")
        active = False
    cv.destroyAllWindows ()

# --------------------------------------------------------------------------------
# Main Project Loop
# --------------------------------------------------------------------------------

# Execute the functions to perform visual servoing and target tracking.
if __name__ == "__main__":

    # Initialize the low-level drivers (do not list the debug drivers).
    cflib.crtp.init_drivers (enable_debug_driver = False)

    # Initialise the SyncCrazyflie class for function blockings.
    with SyncCrazyflie (uri, cf = crazyflie) as crazyflie_sync:

        # Set the parameters for control and rest the extended Kalman filter.
        crazyflie = crazyflie_sync.cf
        set_initial_position_origin (crazyflie)
        reset_estimator (crazyflie)
        #crazyflie.param.set_value ("posCtlPid.xyVelMax", "velocity_max")
        #crazyflie.param.set_value ("posCtlPid.zVelMax", "velocity_max")
        #crazyflie.commander.set_client_xmode (True)

        # Load the current frame from the source and process the image. This is not
        > for target detection, but acts as a buffer to load the video window.
        correct, frame = source.read ()
        frame, mode, threshold, morphology = processing_gray (frame)
        frame, contours, centroid_x, centroid_y, area = target_detection (frame,
        > morphology, time_start, res_x, res_y)
        monitor_minimum (frame, record)
        cv.waitKey (200)

        # Record the current time as the start of the test.
        time_start = time.time ()

        # Take off to an altitude of approximately 0.4m.
        take_off (crazyflie, altitude)

        # Begin capture of the video frames and image processing.
        print ("Beginning capture of the video frames and image processing.")
        print ("To terminate the capture, press [Q] in the video window.")
        print ("Alternatively, press [ctrl+c] in the shell window.")
        try:
            while active == True:

                # Load the current frame from the source.
                correct, frame = source.read ()

                # Minimum grayscale processing with results video capture.
                frame, mode, threshold, morphology = processing_gray (frame)
                frame, contours, centroid_x, centroid_y, area = target_detection
                > (frame, morphology, time_start, res_x, res_y)
                ##frame, orientation = target_detection_orientation (frame,
                > contours) # Uncomment for yaw orientation tracking.
                monitor_minimum (frame, record)

                # Estimate the camera altitude and target centroid position.
                if area < 100 or area > 1000:
                    print ("The detected area is inconsistent with expectations.")
                    land (crazyflie, position_z)
                    crazyflie.commander.send_stop_setpoint ()
                    active = False
                    break
                position_z = 9.128 * area ** -0.484 # Disk Target
                ##position_z = 13.47 * area ** -0.492 # Rectangle Target
                position_x = centroid_x * (position_z / focal_length)
                position_y = centroid_y * (position_z / focal_length)
                ##yaw = orientation # Uncomment for yaw orientation tracking.

                # Based on the target, send a position setpoint to the Crazyflie.
                crazyflie.commander.send_position_setpoint (position_x, position_y,
                > position_z, yaw)

                # Display the target centroid position and camera altitude. This
                > dramatically slows down video capture due to printing to the shell.
                #monitor_text (centroid_x, centroid_y, position_z, orientation)

                # Look for the designated key to terminate the capture.
                key = cv.waitKey (1)
                if key == ord ("Q"):
                    print ("Termination key from [Q] was detected.")
                    land (crazyflie, position_z)
                    crazyflie.commander.send_stop_setpoint ()
                    active = False

        # Look for the designated key to terminate the capture.
        except KeyboardInterrupt:
            print ("KeyboardInterrupt from [ctrl+c] was detected.")
            land (crazyflie, altitude)
            crazyflie.commander.send_stop_setpoint ()
            active = False

    time_end = time.time ()

# ---------------------------------------------------------------------------------
# Finishing Commands
# ---------------------------------------------------------------------------------

# Terminate the capture and release the source.
print ("Terminating the capture and releasing the source.")
source.release ()
if record == "Y":
    print ("The output video is 'Out.avi' in the current directory.")
    video.release ()
cv.destroyAllWindows ()

# ---------------------------------------------------------------------------------
# End
# ---------------------------------------------------------------------------------
```

C ETHICS CONSIDERATIONS

For the completion of the project, there were no ethical considerations with regards to participants, as the requisite knowledge, information, and resources was acquired through self-derived means in the form of published research, online resources, and analytical calculations. Furthermore, the results of the experiment were not doctored or plagiarized in any form so that the scientific method was thoroughly and sufficiently followed. As evaluated during the project proposal by a member of the School Ethics Committee, there were no ethical risks and ethical clearance was not required.

D RISK ASSESSMENT

The following precautions must be followed throughout experimentation:

- At all times within the motion capture facilities, the operators, supervisors, and nearby bystanders must wear the required personal protective equipment (PPE) in the form of a laboratory dust coat, steel-toed boots, and eye protection to reduce risk in the event of a crash.
- There should be at least two operators or an operator and supervisor monitoring the Crazyflie.
- The environment must be clear of obstacles not involved in the tests with no debris or loose items in the vicinity which could be unpredictably lifted by the thrust of the Crazyflie.
- The ground control laptop must be charged or charging before testing or charging while testing to ensure it will not power off during a test while the Crazyflie is flying.
- The Raspberry Pi must also have a reliable power supply for the duration of testing.
- Before charging or testing, it must be ensured that the battery of the Crazyflie is not punctured or swollen. If the battery is damaged in any way, a supervisor must be notified immediately.
- There must be no interference with the connections between the ground control laptop, Raspberry Pi, and Crazyflie to ensure there will be no disconnections while the Crazyflie is flying.
- The state of the hardware should be visually checked before each test to ensure there are no damages and the installation is still correct. This is specifically applicable to the propellers, where damages may have occurred in previous tests, and the correct blades must be installed on the correct rotors for clockwise or counter-clockwise rotation. For the Crazyflie, A, A1, or A2 indicate clockwise rotation and B, B1, or B2 indicate counter-clockwise rotation.
- At the start of a test, the battery of the Crazyflie should always be fully charged. If so, it must also be ensured that the battery installation is correct and secure to the frame without metal or sharp parts contacting the battery and without any possibility of a short-circuit occurring.
- Once the Crazyflie is placed on solid ground and powered on, the status LEDs should be check for any errors detected while the firmware performs internal self-checks during start-up. For the Crazyflie, the LEDs labelled M1 and M4 will indicate the result of the self-checks, where the M4 LED rapidly blinks green five times if the self-checks are passed or the M1 LED rapidly blinks red five times then pauses and repeats again if the self-checks are failed. After the self-checks have passed, the M1 LED should blink twice per second if the sensors are calibrated or the M1 LED will blink once every two seconds if the calibration failed (for calibration, the Crazyflie must be still and the ground should be level). If the M4 LED is constantly red, the battery is low and the Crazyflie should not be flown. The M2 and M3 LEDs should be constantly blue and are not related to the self-checks (used to indicate orientation), unless in a different state for charging or firmware flashing. The motors will also initially rotate in sequence to indicate operation.
- The control scripts on the Raspberry Pi must be checked and free from errors and warnings before connecting to the Crazyflie. If unsure, the Crazyflie can be connected and the control

scripts can be run while there are no propellers attached to the rotors or while safely holding the Crazyflie to observe if there will be any unexpected behaviour (the Crazyflie should be gripped firmly at the bottom of the frame to avoid the propellers).

- When it is ready for a test, the Crazyflie should be placed on solid and level ground with the forward direction pointing away from the operators, supervisors, and nearby bystanders.
- When it begins flying for a test, the Crazyflie should start the motors, take off vertically, and hover at a fixed position to ensure correct, controlled, and stable operation before continuing.
- Once the test is complete and the Crazyflie has landed with inactivity, the Raspberry Pi should be disconnected and the battery of the Crazyflie should be removed (do not pull the battery cables).

The risks involved in the experiment have been identified and evaluated. This involved categorizing the severity, likelihood, and type of the risks, and proposing precautionary actions to further prevent the risks. The complete risk assessment is seen in Table 6 and the official standard operating procedure in the motion capture facilities for quadrotor experiments is seen in Figure 51.

Table 6: Evaluation of the potential risks involved in the experiment.

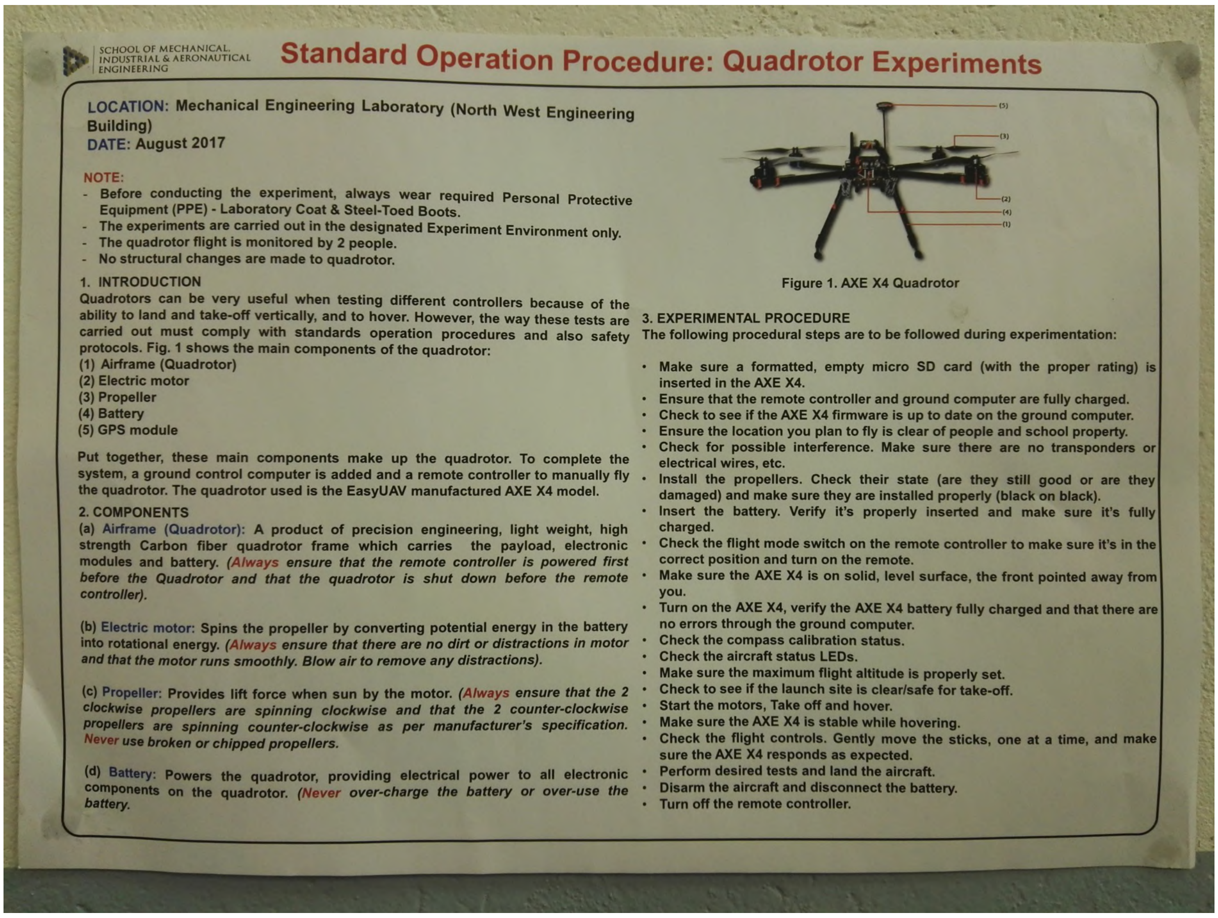
Activity: Quadrotor Visual Servoing For Moving Target Tracking

Venue: North West Engineering Laboratory, Robotics Lab

Risk Type	Risk Description	Severity	Likelihood	Risk Score	Action Type	Action
Mechanical	Due to a loss of control or unpredictable behaviour, the Crazyflie may collide with an operator or person in the vicinity of the experiment. This is especially concerning for the propellers possibly causing short-term injuries to eyes.	Minor	Possible	5	Procedure, PPE	Only necessary personnel should be allowed within the motion capture facilities and operating vicinity. Safety glasses may also be worn within the motion capture facilities.
Mechanical	Hair or loose fitting clothing may become caught or tangled in the motors and propellers of the Crazyflie.	Minor	Possible	5	PPE	Hair must be tied up and loose clothing must be sufficiently restricted.
Mechanical	If the Crazyflie crashes, parts can be dislodged and act as shrapnel.	Minor	Rare	3	PPE	Safety glasses may be worn within the motion capture facilities.
Electrical	Minor electric shocks may be experienced from handling the Crazyflie and Raspberry Pi, but the voltage and current of these shocks are very minimal.	Minor	Unlikely	4	Procedure	In manufacturing and grounding, it is unlikely for electrical shocks but the Crazyflie and Raspberry Pi must still be handled with care such that they are not damaged and become hazardous.
Chemical	The LiPo battery may erupt (fire, explosions, and toxic smoke) if exposed to high temperatures or penetrated, where the lithium will be exposed which is highly flammable and potentially explosive when mixed with air and may cause chemical burns. This may occur during use, charging, or storage.	Major	Possible	7	Modification	The battery must be housed correctly within the frame of the Crazyflie so exposure is minimised in the event that it unexpectedly erupts. The battery must be check before and after each use. A good quality and reliable battery must be used from the Crazyflie suppliers.
Ergonomic	The thrust of the Crazyflie may pick up dust, which may be breathed in by or enter the eyes of the operator or personnel in the operating vicinity.	Minor	Rare	3	Elimination	It should be ensured that the motion capture facilities and operating vicinity are sufficiently clean and dust-free.
Ergonomic	To place the target and Crazyflie on the floor, the operator is required to bend down multiple times	Minor	Rare	3	Procedure	The operator must correctly bend with their knees, instead of their back, to avoid straining muscles.
Ergonomic	The black mats in the motion capture facilities form an uneven surface with the ground, which lead to tripping.	Minor	Rare	3	Procedure	The operator must take care when moving around the motion capture facilities to avoid tripping.

Signed: Edward Rycroft

Date: 2019/10/28



SCHOOL OF MECHANICAL, INDUSTRIAL & AERONAUTICAL ENGINEERING

Standard Operation Procedure: Quadrotor Experiments

LOCATION: Mechanical Engineering Laboratory (North West Engineering Building)
DATE: August 2017

NOTE:
- Before conducting the experiment, always wear required Personal Protective Equipment (PPE) - Laboratory Coat & Steel-Toed Boots.
- The experiments are carried out in the designated Experiment Environment only.
- The quadrotor flight is monitored by 2 people.
- No structural changes are made to quadrotor.

1. INTRODUCTION
Quadrotors can be very useful when testing different controllers because of the ability to land and take-off vertically, and to hover. However, the way these tests are carried out must comply with standards operation procedures and also safety protocols. Fig. 1 shows the main components of the quadrotor:
(1) Airframe (Quadrotor)
(2) Electric motor
(3) Propeller
(4) Battery
(5) GPS module

Put together, these main components make up the quadrotor. To complete the system, a ground control computer is added and a remote controller to manually fly the quadrotor. The quadrotor used is the EasyUAV manufactured AXE X4 model.

2. COMPONENTS
(a) Airframe (Quadrotor): A product of precision engineering, light weight, high strength Carbon fiber quadrotor frame which carries the payload, electronic modules and battery. *(Always ensure that the remote controller is powered first before the Quadrotor and that the quadrotor is shut down before the remote controller).*

(b) Electric motor: Spins the propeller by converting potential energy in the battery into rotational energy. *(Always ensure that there are no dirt or distractions in motor and that the motor runs smoothly. Blow air to remove any distractions).*

(c) Propeller: Provides lift force when sun by the motor. *(Always ensure that the 2 clockwise propellers are spinning clockwise and that the 2 counter-clockwise propellers are spinning counter-clockwise as per manufacturer's specification. Never use broken or chipped propellers.*

(d) Battery: Powers the quadrotor, providing electrical power to all electronic components on the quadrotor. *(Never over-charge the battery or over-use the battery.*

Figure 1. AXE X4 Quadrotor

3. EXPERIMENTAL PROCEDURE
The following procedural steps are to be followed during experimentation:

- Make sure a formatted, empty micro SD card (with the proper rating) is inserted in the AXE X4.
- Ensure that the remote controller and ground computer are fully charged.
- Check to see if the AXE X4 firmware is up to date on the ground computer.
- Ensure the location you plan to fly is clear of people and school property.
- Check for possible interference. Make sure there are no transponders or electrical wires, etc.
- Install the propellers. Check their state (are they still good or are they damaged) and make sure they are installed properly (black on black).
- Insert the battery. Verify it's properly inserted and make sure it's fully charged.
- Check the flight mode switch on the remote controller to make sure it's in the correct position and turn on the remote.
- Make sure the AXE X4 is on solid, level surface, the front pointed away from you.
- Turn on the AXE X4, verify the AXE X4 battery fully charged and that there are no errors through the ground computer.
- Check the compass calibration status.
- Check the aircraft status LEDs.
- Make sure the maximum flight altitude is properly set.
- Check to see if the launch site is clear/safe for take-off.
- Start the motors, Take off and hover.
- Make sure the AXE X4 is stable while hovering.
- Check the flight controls. Gently move the sticks, one at a time, and make sure the AXE X4 responds as expected.
- Perform desired tests and land the aircraft.
- Disarm the aircraft and disconnect the battery.
- Turn off the remote controller.

Figure 51: Standard operating procedure in the motion capture facilities for quadrotor experiments.

E BIGQUAD DEVELOPMENT

To complement the end-point open-loop control, an investigation into end-point closed-loop control was initially performed using a larger quadrotor, on-board camera with the Pi Camera, and on-board processing with the Raspberry Pi. However, it was not possible for the quadrotor to fly safely due to the risk of damaging the exposed LiPo battery, so focus was shifted to more in-depth end-point open-loop control only. The partial development is included as a reference for future progress.

E.1 END-POINT CLOSED-LOOP CONTROL

For end-point closed-loop control, the basic arrangement for location estimation is essentially the same as for end-point open-loop control, except the camera would be mounted to the quadrotor instead of being fixed in the surroundings. This is illustrated in Figure 52, with similarity to Figure 13.

It might be necessary to investigate compensation to stabilize the on-board camera while rolling and pitching, as seen in Figure 53, where the field of view becomes inaccurately distorted with incorrect measurements of the position of the target. Compensation could be implemented using mechanical compensation, where the on-board camera is mounted with a two degree-of-freedom gimbal which

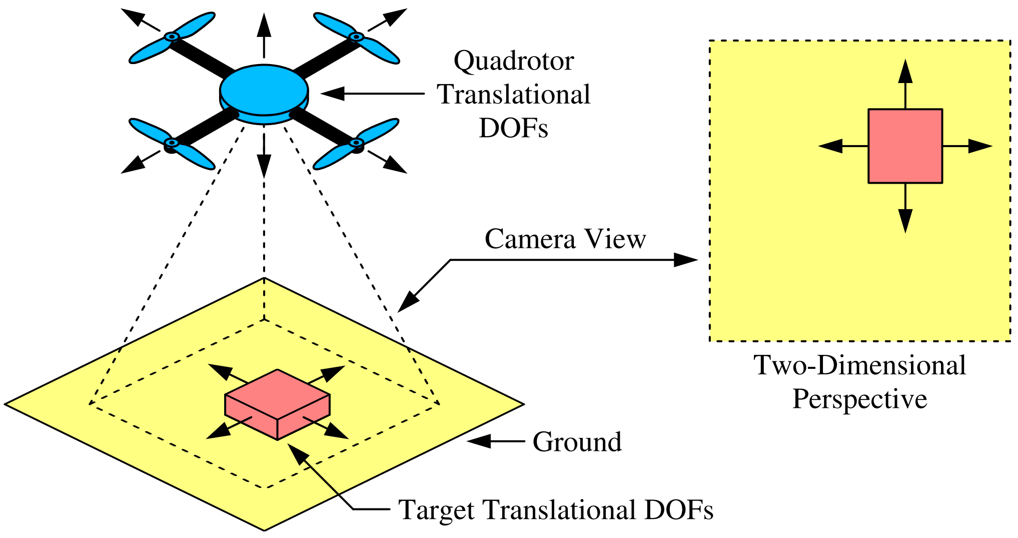


Figure 52: Arrangement for end-point closed-loop control through detection and tracking of a target.

isolates the on-board camera from the effects of rolling and pitching; or digital compensation, where the image is slightly cropped to allow for virtual stabilisation [7]. Mechanical compensation will retain the field of view while being faster without processing delays as compared to digital compensation, so it is recommended for mechanical compensation to be used if compensation is required [7]. However, no compensation should first be adopted to judge if it is satisfactory.

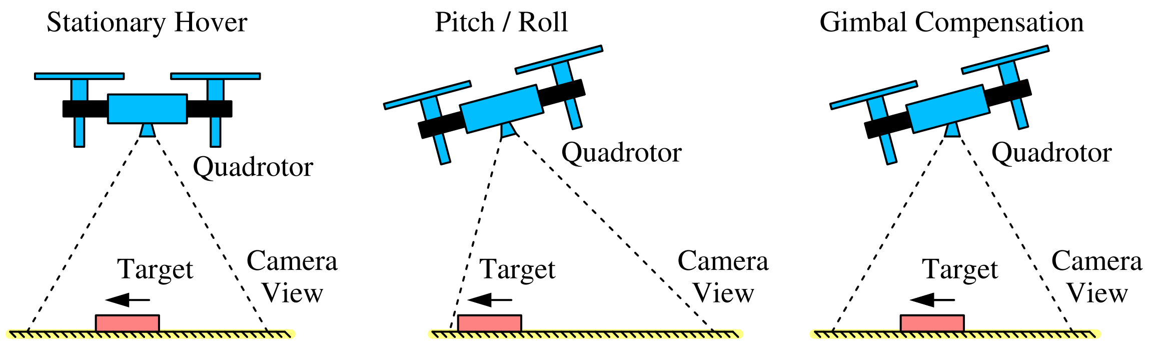


Figure 53: Comparison between the changes in the field of view during extreme/exaggerated rolling or pitching without (middle) and with (right) compensation as the target begins a motion.

E.2 BIGQUAD APPARATUS

For the quadrotor, the Crazyflie control board was mounted to a TransTEC Freedom frame, which has diagonals of 215mm and is constructed from 3K carbon fibre with aluminium 7075 supports for the shell and plastic bumpers at the rotor mounts to produce a mass of 116g. The electronic components for the flight of the quadrotor include four EMAX Bullet 30A electronic speed controllers (ESCs), with a mass of 4.9g, BB2 processor, current rating up to 30A, and support for DShot, Multishot, and Oneshot protocols; four EMAX RSII 2206-1700KV rotor motors, with a mass of 26.7g and maximum thrust up to 2040N/kg; four plastic propellers, with a mass of 4g and length of 127mm; and Tattu R-line 75C LiPo battery with a mass of 160g, four cells in series, and capacity of 1300mA.hr at 14.8V.

To facilitate the connection between the Crazyflie and ESCs, the Bitcraze BigQuad expansion deck (subsequently referred to as BigQuad deck) will be used. This deck features breakout header connectors for the four ESCs, where a voltage and ground will be supplied along with a PWM signal at a default frequency of 400Hz to control the motors [30]. There are also additional breakout header connectors for other accessories, such as a GPS receiver, battery voltage and current monitor, buzzer, chaotic pulse position modulation (CPPM) receiver, or I2C communication protocol [30]. The connections of the BigQuad deck are seen in Figure 54 with the final assembly seen in Figure 55.

Unfortunately, the BigQuad deck is not compatible with the Flow deck without modifying the firmware and components of the hardware. As a result, the control and stabilisation is slightly more difficult and requires a different method based on measurements relative to the target detection. Moreover, the original internal cascading PID controllers for attitude flight control of position and velocity are tuned for the original frame, rotor motors, propellers, and battery, so it might be necessary to re-tune the controller gain values if the flight is found to be unstable or unresponsive.

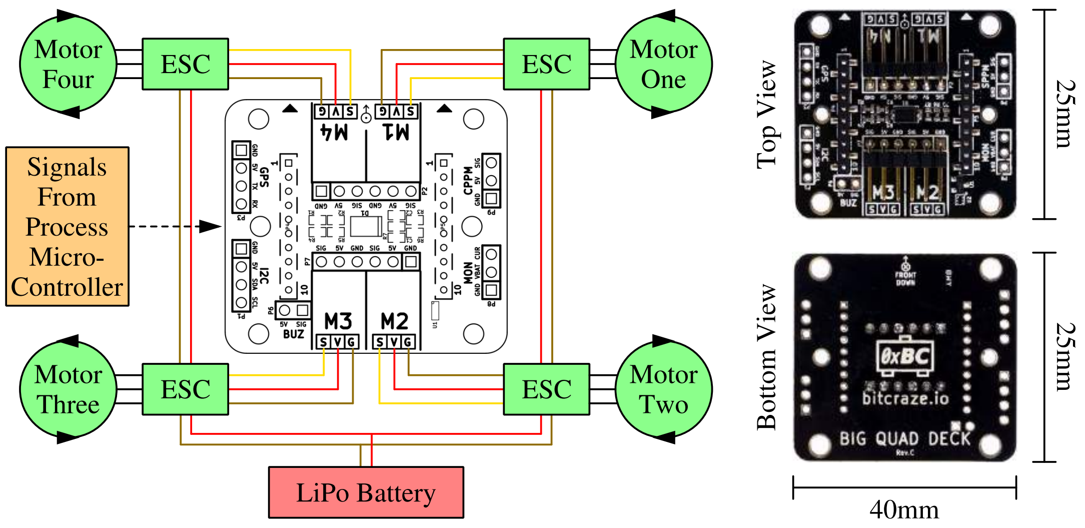


Figure 54: Schematic (left) and photographs (right) showing the Bitcraze BigQuad expansion deck and connections to the EMAX Bullet 30A ESCs and EMAX RSII 2206-1700KV rotor motors [31].

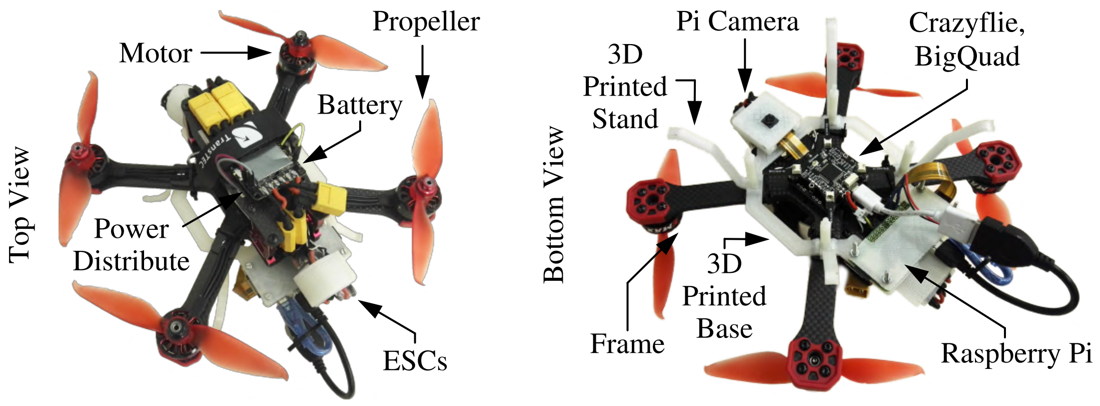


Figure 55: Photographs showing the top isometric (left) and bottom isometric (right) views of the larger quadrotor which was manually assembled, including the parts for visual servoing.