SCHOOL OF MECHANICAL ENGINEERING

UNIVERSITY OF LEEDS

MECH3890 – Individual Engineering Project

Design, Simulation and Evaluation of an Autonomous Window Cleaning Robot

PRESENTED BY **Alex Bury**

If the project is industrially linked, tick this box [ ] and provide details below

COMPANY NAME AND ADDRESS:

STUDENT DECLARATION (from the "LU Declaration of Academic Integrity")

I am aware that the University defines plagiarism as presenting someone else's work, in whole or in part, as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance. I promise that in the attached submission I have not presented anyone else's work, in whole or in part, as my own and I have not colluded with others in the preparation of this work. Where I have taken advantage of the work of others, I have given full acknowledgement. I have not resubmitted my own work or part thereof without specific written permission to do so from the University staff concerned when any of this work has been or is being submitted for marks or credits even if in a different module or for a different qualification or completed prior to entry to the University. I have read and understood the University's published rules on plagiarism and also any more detailed rules specified at School or module level. I know that if I commit plagiarism I can be expelled from the University and that it is my responsibility to be aware of the University's regulations on plagiarism and their importance. I re-confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes. I confirm that I have declared all mitigating circumstances that may be relevant to the assessment of this piece of work and that I wish to have taken into account. I am aware of the University's policy on mitigation and the School's procedures for the submission of statements and evidence of mitigation. I am aware of the penalties imposed for the late submission of coursework.

Signed REDACTED date **26/04/2022**

Abstract

Buildings require regular maintenance and inspection to ensure that they meet their lifetime requirements, and this is traditionally done manually. However, with recent advancements in robotics, it may be possible to autonomise the processes to reduce costs. This purpose of this project was to design and evaluate an autonomous window cleaning robot, and this report contains the procedures followed, design specification, and critical evaluations of the final designs.

The quadcopter locomotion method was chosen due to its versatility, simplicity, and availability of components, while an axial rotating drum brush with a ground-tethered water feed was designed. The use of a ground-tethered power supply was also investigated but was deemed impractical for this project. Consequently, the resulting product has a flight time of around 17 minutes with 178Wh of onboard batteries.

In addition, a simulation environment in Unity was created, and a computer vision (CV) program was developed in Python using OpenCV [1] to recognise and track windows in the scene. While the filters used to recognise the windows were found to be reasonably effective, the implemented midpoint tracking method failed to track windows reliably. The CV programme was also tested on a pre-recorded video [2] where similar results were observed, although performance was generally worse due to the additional detail. A tree-based tracking algorithm is suggested to overcome this issue alongside the issue of occlusion.

Table of Contents

1. Introduction

1.1 Introduction

In the modern world, materials such as glass, aluminium and concrete are widely used in construction, particularly as cladding in large buildings. These materials are exposed to the atmosphere and therefore are designed to be resistant to the elements, however without proper maintenance they are likely to degrade. This maintenance generally involves regular cleaning and inspection to ensure that debris from pollution and other environmental factors does not promote degradation of materials. Traditionally, this process has been very labour-intensive and therefore expensive, but with recent advancements in technology, it may be more economical to conduct the cleaning autonomously using a robot.

In addition, the applications of autonomous cleaning robots extend well beyond just cleaning buildings, for example Aerones manufacture a commercial drone which is used for inspection and maintenance of large wind turbines [3]. Other areas include cleaning solar panels, where a small amount of dust or debris can reduce power output by as much as 85% [4], and the cleaning of an aircraft's exterior where dirt can significantly increase drag and therefore fuel consumption [5].

The purpose of this project was to design and evaluate a cleaning robot, with a primary focus of cleaning individual windows on buildings. This report documents the design process for the mechanical, electrical and programming components of such a robot, the challenges faced, and potential solutions for them.

1.2 Aim

To design, simulate and evaluate an automated robot to clean windows of conventional buildings.

This includes selecting the locomotion system (quadcopter) and cleaning system (axial rotating brush), designing the robot in CAD, creating a full bill of materials with a complete cost and weight breakdown, setting up a simulation environment, developing computer vision programming and critically evaluating its effectiveness.

1.3 Objectives

Table 1.1 contains the objectives for this project, their deadlines and the work required to meet them.

Table 1.1 – Project objectives, required work and deadlines.

	Objective	Required Work	Deadline
1	Create a high-level design specification for the robot.	Evaluate existing cleaning and autonomous robots, their capabilities and limitations.	30/11/21
2	Design the mechanical aspects of the robot including the propulsive systems.	Research the design process, create a design to meet the requirements in CAD and source components for the design.	11/01/22
3	Design the electrical aspects the robot.	Evaluate the requirements, research different sensing/actuation methods and design their implementation in CAD.	26/01/22
4	Create a simulation environment for the drone.	Determine which platform and languages to use, what libraries are needed and create the environment.	08/02/22
5	Create a computer vision programme to map and track windows in a scene.	Determine suitable libraries, study computer vision operations and design processes to recognise and track windows.	18/03/22
6	Evaluate the mechanical, electrical and programming implementations.	Study the implementations and their effectiveness, comparing them to commercial solutions where appropriate.	21/04/22

1.4 Report Layout

Sections 2 and 3 detail the mechanical and electrical design respectively, and a complete bill of materials for both sections is presented at the end of section 3. In each of these sections, the overall design requirements are considered and specific requirements for each aspect of the robot are evaluated. From this, components are selected and their implementation is demonstrated in labelled diagrams and renders.

Section 4 details the design process specifically for the window mapping computer programme. Here, aims are considered and the implementation is presented as a broken-down list of steps. Python code from the key parts of the computer programme is presented in Appendix 5. In addition, the effectiveness of the programme is evaluated using a pre-recorded video [2].

Section 5 concludes the report with a brief evaluation and discussion of the overall design and highlights areas for future work.

2. Mechanical Design

For a robot to function correctly, it's mechanical elements must be efficient and effective. This section details how the mechanical side of the design process was carried out and includes details of the high-level concept selection process.

2.1 Objectives

A number of design requirements were determined based upon the overall aims of the project. These specify that the vehicle must:

1. Be able to start, manoeuvre, and stop under its own power while outside in mild weather conditions at altitudes of less than 2000m
2. Be attached to a ground unit via a tether at all times, through which water and power are delivered, or alternatively be battery powered and have a run time of 10 minutes or greater.
3. Be able to clean window panes of the most common sizes, orientations and heights.
4. Be able to reach a window height of 10 metres while cleaning
5. Consist of readily available and affordable components, with an overall part & material cost of less than £1000.

2.2 Concept Selection

2.2.1 Locomotion System

For the robot to clean windows & building cladding, which are often vertically orientated, it must be able to navigate in a 3D environment in all directions at low speed. Therefore the choice of locomotion system was limited to three options: rotary wing, climbing, or hybrid UAV systems. Other systems like fixed-wing or rocket UAV propulsion were ruled out because of their inherent high-speed and low-control characteristics. Ground-based locomotion systems such as those presented by Siegwart, Illah and Scaramuzza [6] are clearly inappropriate as they do not meet the requirement of traversing vertical walls.

Climbing robots can offer the advantage of lower energy consumption, as they are supported by the frictional force between themselves and the substrate/cable they climb on, while rotary wing aircraft rely on momentum and pressure thrust generated by the rotors which require continual energy input. However, climbing robots are far less flexible than rotary wing UAV as they are limited by the nature of the substrate material and the contours on it. Given this lack of flexibility, commercial climbing window cleaning robots are almost always tethered to the building via a safety cable to minimise the risk of the robots falling from the surface.

Tethered window cleaning robots such as the SkyScraper-I [7], Gekko Façade [8] and Welbot Cleaner [9] require at least one fixing point at the top of the building to function. This means that the owner must undertake significant installation and maintenance costs to use the product, and this may outweigh the benefit of automating the process altogether, particularly in small buildings. In addition, a robot that requires permanent infrastructure to operate will likely not be able to move between sites and therefore will not be used for a significant proportion of it's life. If instead the robot could be taken between different sites, it could operate for a much lower cost per m^2 of window cleaned as economies of scale will apply.

Rotary wing aircraft are a compelling option for the locomotion system as they can navigate in true 3D space and generally do not require specific infrastructure to operate. Of these rotary wing aircraft, quadcopters are attractive because although heavier than single rotor aircraft, they have relatively high operational stability, excellent load capacities and are relatively compact [10]. In addition, quadcopters generally rely on varying the rotor speed for movement and do not require complex mechanisms like their single rotor counterparts [10]. Moreover, a large range of low-cost quadcopter components are widely available. For these reasons, it was chosen to use a quadcopter propulsion system for this robot.

2.2.2 Cleaning System

Commercial automated window cleaning systems generally involve a variation of a rotating brush, although some rely on water high pressure water jets to remove dirt [11]. In an investigation by Sangpradit, it was found that polythene material that had been exposed to the elements and left to accumulate dirt was best cleaned by soaking and mechanically scrubbing, while the waterjet failed to lift much of the dirt [12]. For this reason, it was decided to implement a mechanical scrubbing brush with a water feed. An axial brush arrangement was chosen over radial brushes because the drone tilts to apply the normal force, and so a radial arrangement would require a complex and heavy tilting mechanism. In addition, given the high mass of water and its ability to prevent dust and take away particles from the cleaning site, it was decided to use a water feed that is tethered to the ground.

2.3 Specification Outline

2.3.1 Dimensions

The cleaning head is the one of the main considerations when determining an appropriate size for the quadcopter because of design target 3; the requirement to be able to clean the most common sizes of windowpanes. Another primary consideration is that the cost of the quadcopter is likely to increase exponentially with increasing dimensions because a greater number of high-power motors, and therefore electronic speed controllers (ESCs), would be required. Consequently, the dimensions of this prototype are likely to be smaller than an equivalent commercial solution to minimise cost.

For the above reasons, it was decided that the quadcopter should use either 5 or 6 inch diameter propellers with a 250mm square frame, as 8 inch propellors would drastically increase the overall size and weight while not offering a significant performance advantage. This would enable a cleaning head of around 270mm in length to be used, which is small enough to allow access to most windows.

2.3.2 Mass

A target mass of 3kg was selected by considering weights of similarly sized commercial quadcopters, the propulsive force generated by commercially available motor-rotor combinations, the weight of common LiPO batteries and the requirement to carry a tethered water tube up to heights of 10m.

2.3.3 Normal Scrubbing Force

From common experience, the normal force applied between a scrubber and substrate is closely linked to the rate of dirt removal; if you press harder while you scrub, you increase the shear force between the two surfaces and dirt is likely to be lifted faster. However, too much force may result in damage to the surface as the dirt is pressed into the substrate, therefore the normal force should be carefully chosen. From experience, humans vary the normal force automatically as we see areas that have not been properly cleaned, and then return to these areas with more force.

To mimic the cleaning action of humans, a robot would require some form of closed-loop control where the robot evaluates the cleanliness of the substrate before and after cleaning, and then adjusts the parameters accordingly to maximise efficiency. While this may be possible with more resources, it is an unrealistic expectation for this project, and so a pre-determined normal force was used.

To determine this force, a hand was placed onto a vertical scale and a force was applied as it would be when cleaning, and it was determined to approximately be the equivalent of 1kg (9.81N). To achieve this, the quadcopter is required to tilt so that a component of its thrust force acts towards the wall, and this is affected by its weight and propulsive capabilities. Given that the maximum force of the propulsive system needs to be around twice that of the quadcopter's weight of 3kg [13, 14] (or 30N), this force is realistically achievable.

2.3.4 Propulsive Design

To have the quadcopter powered via a tether cable is desirable because it enables continuous operation, therefore the electrical power requirement was evaluated.

Considering the quadcopter's target mass of 3kg, and that the maximum thrust generated should be at least twice the quadcopter's weight reference [13, 14], each rotor would need to generate a minimum of 1.5kg of thrust (14.7N). Table 2.3.1 contains a number of commercially available quadcopter motor and rotor combinations that have been found to generate thrust in this region through static tests. Out of all these quadcopter-rotor pairs, the Axis AF227 was found to have the lowest peak power consumption at 760W, and therefore a quadcopter consisting of four of these rotors would require a 3040W power supply.

Given that the maximum power limit of a standard mains UK outlet is 3000W (13A), the quadcopter would likely have to be powered from an industrial outlet, which are considerably less common than standard outlets. In addition, to transmit 3000W at 24V (125A) with 95% efficiency, using an online calculator [15] it was calculated that a pair of cables with a cross section of 70mm^2 would be required. Given that at least 10 metres of cable is needed to reach the desired height, the quadcopter would need to carry at least 140cm^3 of copper which would weigh 1.25kg assuming a density of 8.96g cm^3 [16], not to mention the additional mass from insulation, connectors and strain relief mechanisms.

Table 2.3.1 – A non-exhaustive list of commercially available motor and rotor pairs that generate over 1500g of thrust.

Brand	Size	Thrust /g	Motor Mass /g	Power /W	Propellor	Total Mass /g	Motor Cost
Axis AF227 [17]	2207	1591	50	760	GF 51466	216.52	£21.90
Hobbywing Xrotor race pro [18] [19]	2207	1890	32	953	Azure Power 5150 - 3	152.36	£19.50
Brotherhobby returner R4 [20] [21]	2206	1667	29	807	tj6045	132.8	£19.50
GepRC GR2207 [22] [23]	2207.5	1738	32	733	HQ 6045	146.8	£16.98
T-Motor Velox V2 [24]	2306	1587	32	823	T5146	146	£12.45
iFlight XING-E [25]	2306	1642	33	893	6045	151.2	£9.68

Therefore, a power supply would most likely need to be fitted to the quadcopter to enable the use of a lighter transmission cable in which power could be supplied at much higher voltages to reduce resistive losses. However, it was determined that no commercially available power supplies meet the weight and size requirements for this application, hence a custom design would be required. Consequently, the decision was made to abandon the idea of supplying power via a tethered cable as the design of such a supply lies outside the scope of this project, and instead it was chosen to power the quadcopter using a conventional LiPO battery for this prototype.

In this case, given the relatively small cost of propellors, the iFlight XING-E 2306 1700Kv [25] motors were chosen because of their low price and relatively low operating current which should mean they run cooler. In addition, they are relatively lightweight and compatible ESCs and batteries are readily available.

2.4 Final Design

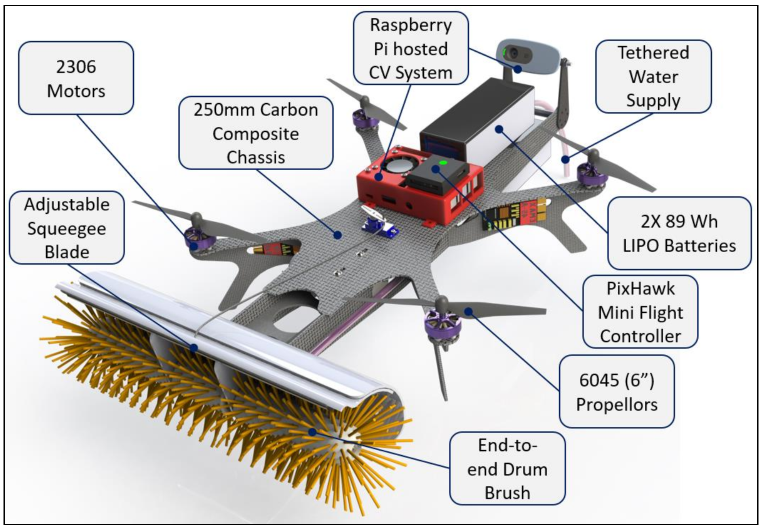

Raspberry Pi hosted CV System
2306 Motors
Tethered Water Supply
250mm Carbon Composite Chassis
Adjustable Squeegee Blade
2X 89 Wh LIPO Batteries
PixHawk Mini Flight Controller
6045 (6") Propellors
End-to-end Drum Brush

Figure 2.4.1 – Labelled Render Diagram showing some of the key design aspects of the robot. Some component CAD models provided by [26, 27, 28, 29, 30, 31, 32, 33, 34]

Figure 2.4.1 contains a labelled render diagram of the final design. The principles outlined by Pounds and Mahony [35] for large practical quadrotors were followed wherever possible to ensure that the final design is efficient and effective. However, compromises were made in some areas, primarily due to the relatively low-cost target and the availability of commercial-grade UAV components. These include that many of the components are intended for hobby aircraft – the motors for example are commonly used in point-of-view (POV) racing quadcopters and are therefore not optimised for this practical application.

While this may restrict product development in future, it was deemed appropriate to use hobby-grade components in this prototype as this project is only intended to investigate the concept, not to design a marketable product.

2.4.1 Weight Analysis

The design weight of the robot plays a pivotal role in motor and rotor selection, which is why it was one of the first parameters to be assigned. Throughout the design process, it was closely monitored to ensure that the final product meets this target. Figure 3.4 contains a complete list of all the components and their weights, and the overall mass of the final robot was calculated to be 3.089kg.

While slightly over the target design mass, the quadcopter has met its requirements since the motors are specified to each have 1.642kg [36] of thrust which means the absolute maximum design mass of the quadcopter is 3.284kg [13, 14]. However, given the non-linear relationship between mass and endurance [37], it is desirable to minimise weight as much as possible.

Further weight reduction may be achieved by optimising the chassis and cleaning head and using more energy dense batteries, as these are the heaviest components by a considerable margin.

2.4.2 Cleaning System

The direction of rotation was an important consideration in this design, as if the frictional force acts in the same direction as the quadcopter's weight, the motors would be required to increase their power output and as a result, the quadcopter would have a tendency to be 'sucked' into the wall; whereas if the frictional force acts in the same direction as the thrust forces (as shown in Figure 2.4.2), the quadcopter is likely to be more stable and will have a lower power consumption.

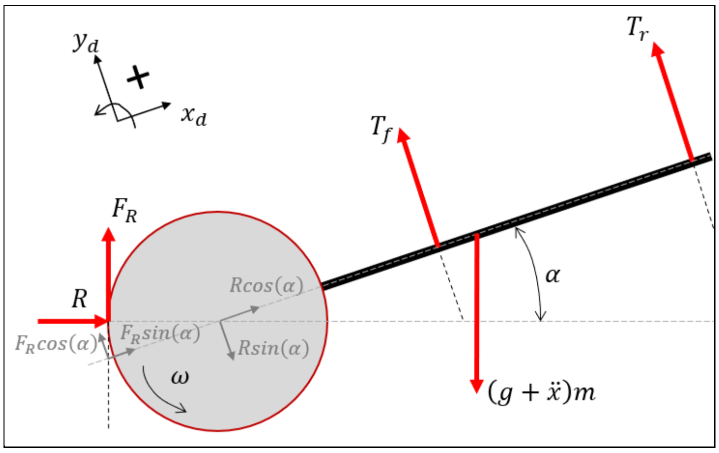


Figure 2.4.2 – Free Body Diagram of a Window Cleaning Quadcopter

Knowing this, and the fact that the quadcopter must clean windows moving downwards due to the nature of gravity and its effect on the wastewater, it was determined that the linear velocity of the brush must be at least twice that of the quadcopter's velocity while cleaning to ensure a good amount of shear force between the surface and brush for dirt removal. Consequently, the minimum required RPM of the brush was calculated using Equation 2.1, assuming that the quadcopter's cleaning velocity is 0.5ms^{-1}.

$$B_{RPM} = \frac{v_l}{2\pi r} \times 60 = \frac{1}{2\pi \times 0.085} \times 60 = \mathbf{112RPM} \qquad (2.1)$$

In addition, a suitable motor and gear ratio were selected to ensure that the motor has enough torque to rotate the brush at an acceptable velocity. To determine this, a simple model was used that assumed the brush to be a homogenous rigid cylinder, and that the coefficient of friction between the glass and the brush is 0.6. This value is based upon an approximate average of the experimental findings of Holopainen and Salonen from an investigation into the cleaning of metallic air ducts using brushes [38]. Although the substrate material is different in this investigation, glass windows are generally smooth, so the coefficient of friction is likely to be lower than that of the air duct material. Equation 2.2 was used to calculate the required motor torque, assuming a gear ratio of 1:1, and the derivation is presented in Appendix 4.1.

$$T_M = \mu R r_B G_r = 0.6 \times 9.81 \times 0.085 \times 1 = \mathbf{0.5Nm} \quad (2.2)$$

From this evaluation, the Crouzet 82862006 [39] was chosen to be the drive motor for the brush. Since this motor cannot be packaged easily into the cleaning head, it was mounted in-board, and is connected to the cleaning brush via a 4mm wide timing belt. This type of belt was chosen because it is the narrowest commonly available timing belt and therefore minimises the gap between the bristles.

The inclination angle of the quadcopter while cleaning, α, was calculated using Equation 2.3, assuming 1ms^{-2} acceleration downwards. The full derivation of this equation is presented in Appendix 4.2.

$$\alpha = \tan^{-1}\left(\frac{R}{(\ddot{x}m + gm - R\mu_{\mathrm{eq}})}\right) = \tan^{-1}\left(\frac{9.81}{(3 + 9.81 \times 3 - 9.81 \times 0.6)}\right) = \mathbf{20.3}° \quad (2.3)$$

2.4.3 Cleaning Head Implementation

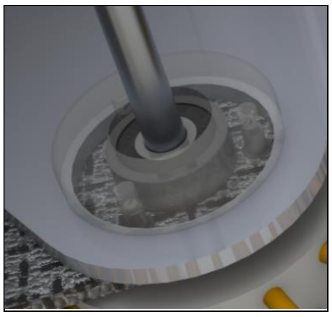


Figure 2.4.3 – Close up of the bearing housing (shown transparent for clarity).

Since the brush will be rotating at high speeds (greater than 112RPM) and will be loaded in the axial and radial directions, two ball bearings were selected. Here, both bearings are of the doubly sealed variety to ensure that the water from the cleaning operation does not wash out the grease, and equally so that the grease does not contaminate the working surface. However, the squeegee mechanism will only rotate by small fractions at very low speeds, therefore an unlubricated plain bearing was deemed suitable.

To ensure that the head is capable of cleaning along the entirety of its width, the gaps for the supporting parts of the chassis and drive belt were minimised. Consequently, a bearing housing was created that sits within a recess in the end of the rollers which also acts as the pivot point for the squeegee blade. Figure 2.4.3 demonstrates this implementation.

2.4.4 Transitions Between Operational Modes

Once the quadcopter is positioned next to the window, the brush will begin to turn and the quadcopter will gently touch the window, slowly increasing the inclination angle up to the desired value. It is important that the brush begins turning before it touches the window to prevent the larger static coefficient of friction from causing a sudden jolt which could make the quadcopter crash, and to prevent high current spikes from the brush's drive motor when being started under load.

3. Electrical System Design

The electrical systems of the robot were designed following the mechanical design as the selection of various components, including the propulsive motors, took place in the mechanical design phase. This section details the selection processes for components and provides an evaluation of the final design.

3.1 Objectives

From the overall objectives of the project, a number of specific objectives were defined for the autonomous system. These include that the quadcopter must:

1. Take-off and manoeuvre
2. Identify and track windows
3. Generate a response to approach and clean the windows
4. Land and disarm for the operator

3.2 Environmental Sensing Methods

For a robot to navigate autonomously in an environment, it must be able to conduct **s**imultaneous **l**ocalisation **a**nd **m**apping (SLAM) [40, 41, 6], and to achieve this it needs to sense the surrounding environment. This section details common environmental sensing methods and discusses their issues in this specific application, and ultimately describes which method is most appropriate and why.

3.2.1 Wave-Detection-and-Ranging methods

Radio **d**etection **a**nd **r**anging (RADAR) was developed in the mid-20th century, primarily for military use [42], and was designed to identify and locate enemy aircraft and vessels in the battlefield [42]. Still used today, it is a contactless method for taking distance measurements from the surrounding environment, and makes use of a radio wave transmitter, radio detector and a precise clock to evaluate the time of flight (time between sending the pulse and receiving the reflection), which is proportional to the distance travelled [43].

Light **i**maging, **d**etection **a**nd **r**anging (LIDAR) is similar, except it makes use of Light Amplification by Stimulated Emission of Radiation (LASERs) and a light detector, in place of the radio equivalents [44] [45]. Although lidar has applications extending way beyond environmental sensing for robots, including measuring atmospheric conditions [44] and mapping the surfaces of terrestrial bodies [45], it is often used as one of the primary navigational sensing methods in robots [6] [41].

Sound **n**avigation **a**nd **r**anging (SONAR) uses sound waves to achieve the same result, and is often used in sub-marine environments [46] [41]. Ribas et. Al. discusses how sonar can be used for "simultaneous localisation and mapping" [41] in underwater autonomous vehicles, despite their lesser reliability [41].

3.2.2 Computer Vision

A number of image processing libraries such as JavaVis [47], Khoros [48], OpenCV [49], VIGRA [50] and CVIPtools [51] are available and enable patterns, colours, contours and other visual characteristics from images to be recognised and interpreted in real time. Although they may be computationally expensive [6], the high-bandwidth nature of vision [6] means that it may be possible to rely solely on computer vision for navigation.

One challenge with computer vision techniques is determining the distance from and between objects in an image [6]. A number of different approaches to solving this problem exist, including:

1. Calibration – a pre-determined pattern with known dimensions is placed in the scene, and the computer vision algorithm recognises it and can relate from it the surrounding environment [6].

2. Stereo imagery – multiple images are used, taken from two different cameras or the same camera at different locations, and compared to determined depth [6].

3. Hybrid sensing – using another sensing method in combination with a camera and combining the depth data before processing. RGBD is an example, which is discussed in the next section.

3.2.3 RGBD

Red, **G**reen, **B**lue and **D**epth (RGBD) sensors allow the mapping of a real 3D environment by combining a coloured image with a depth plot [52, 53]. When processed using more advanced computer vision programmes, a greater precision can be achieved versus regular RGB cameras [47].

Many RGBD cameras use the structured-light method to determine depth at different points in a scene [52] [54]. This method generally uses an IR projector and camera in conjunction with a traditional RGB camera and combines the sensor data from both systems into a single RGBD map [52].

KinectFusion, a tool which enables rapid mapping of an indoor space using a moving Microsoft Kinect, was developed by Izadi et al. [55] and demonstrates that it is possible to conduct real-time localisation and mapping using only an RGBD sensor.

3.2.4 Contact

While contact sensors may be of use in some circumstances, using it as the primary sensing method for navigation would likely prove inefficient and ineffective as when a vehicle, particularly an airborne vehicle, collides with an obstacle, it is likely to crash and suffer damage. Therefore the vehicle would be required to first survey the environment very slowly while mapping it out, which would be a very time consuming process. In fully autonomous robots such as iRobot's Roomba [56], which primarily relies on contact and close-range non-contact sensing methods for navigation [57], this inefficiency is acceptable because the robot is designed to operate in one area only such that no intervention is required between operations.

3.2.5 GPS Localisation

The Global Positioning System (GPS) was developed by the United States' Department of Defence for their military operations and was later made available for civilian use [58]. It makes use of an array of satellites which communicate time information to ground based antennas, from which accurate localisation can be determined [58].

3.2.6 Other Methods

Methods such as those which rely on magnetic fields, fluid pressure or light intensity for example are unlikely to be of use for the main navigational systems because the operational environment will change as the robot completes its window cleaning task. However, these sensors may be of use in the other operational systems of the robot, such as measuring the rotation speed of the cleaning brush using an optical encoder or measuring the liquid flow rate using a flowmeter. A more exhaustive range of robotic sensing methods are discussed by Hall [59].

3.2.7 Sensor Selection

While an RGBD sensor coupled with a computer-vision programme is an obvious choice for many robots, the operational environment for a window cleaning quadcopter must be considered. This environment is outdoors and requires recognition of highly reflective and transmissive surfaces (windows).

Although lidar is used extensively in robots [6] [45], one of its main drawbacks is its inability to consistently identify windows as most of the light passes through the glass [60]. This poses a real challenge for this application and therefore it was deemed inappropriate.

Given this window cleaning robot will be required to operate in urban areas close to very large buildings, GPS localisation is likely to be severely inadequate as the urban environment is likely to have an adverse effect on the GPS signal and may result in miss-identification [61]. For this reason, GPS cannot be relied upon.

For sensors that use the structured light technique to gather depth information [40], such as the Intel Realsense D435 [54] and Microsoft's Kinect [62], performance is likely to be poor outside as the structured light projectors within the sensors are unlikely to work reliably in brightly lit environments with reflective surfaces [63]. For this reason, a sensor which uses stereo-camera system to gather depth information, such as the Intel D455 [64], would be more appropriate. However, the monetary cost and weight is likely to increase as additional processing power would be required.

An alternative technique which can achieve a similar result is the "structure from motion" [6] approach which makes use of the robot's ability to move. In this method, multiple images are taken using one camera which are then compared to generate an accurate depth map [6]. Although this method is computationally expensive [6], it is likely to be more reliable than the structured light approach and cheaper than the stereovision approach, and for these reasons it was chosen to be the primary navigational method for the robot's autonomous system.

One of the challenges of this method is the requirement to know the position of the camera in the scene, however it may be possible to determine this if there is a known reference dimension within the field of view, or if the precise kinematics of the drone are measured, perhaps using an accelerometer.

An alternative approach, investigated by Majdik et Al., is to use Google street view for localisation in urban environments [61]. While an interesting idea, it does not provide the level of flexibility required to clean windows, as buildings almost always have windows which are facing away from the street and therefore are not shown by the google street view photos. However, this technique could be used in combination with a structure-from-motion approach to improve localisation precision and reliability.

Although the most suitable localisation method has been determined, this investigation does not include a full practical implementation of this method, as a decision was made to focus on the more unique window-cleaning specific aspects of the project.

3.3 Data Processing

While CV-based sensing techniques are generally more processor intensive than others [6], in recent years low cost and compact computers such as the Raspberry Pi have become significantly more powerful. For this reason, and the fact that Pi's have proven effective CV platforms in other projects [65, 66, 67], it was decided to use a Raspberry Pi 4 [68] as the main autonomous computer for the robot. In addition, it was decided to use a commercially available flight controller for the robot as if the Pi was used instead, the drone would be vulnerable to crashing when the processor is loaded by the CV programme. The PixHawk Mini running PX4 was selected as it is a relatively low cost open-source product with a number of great features and excellent community support [69].

3.4 Final Design

Figure 3.4 contains a diagram of all the electrical systems and the connections between different components.

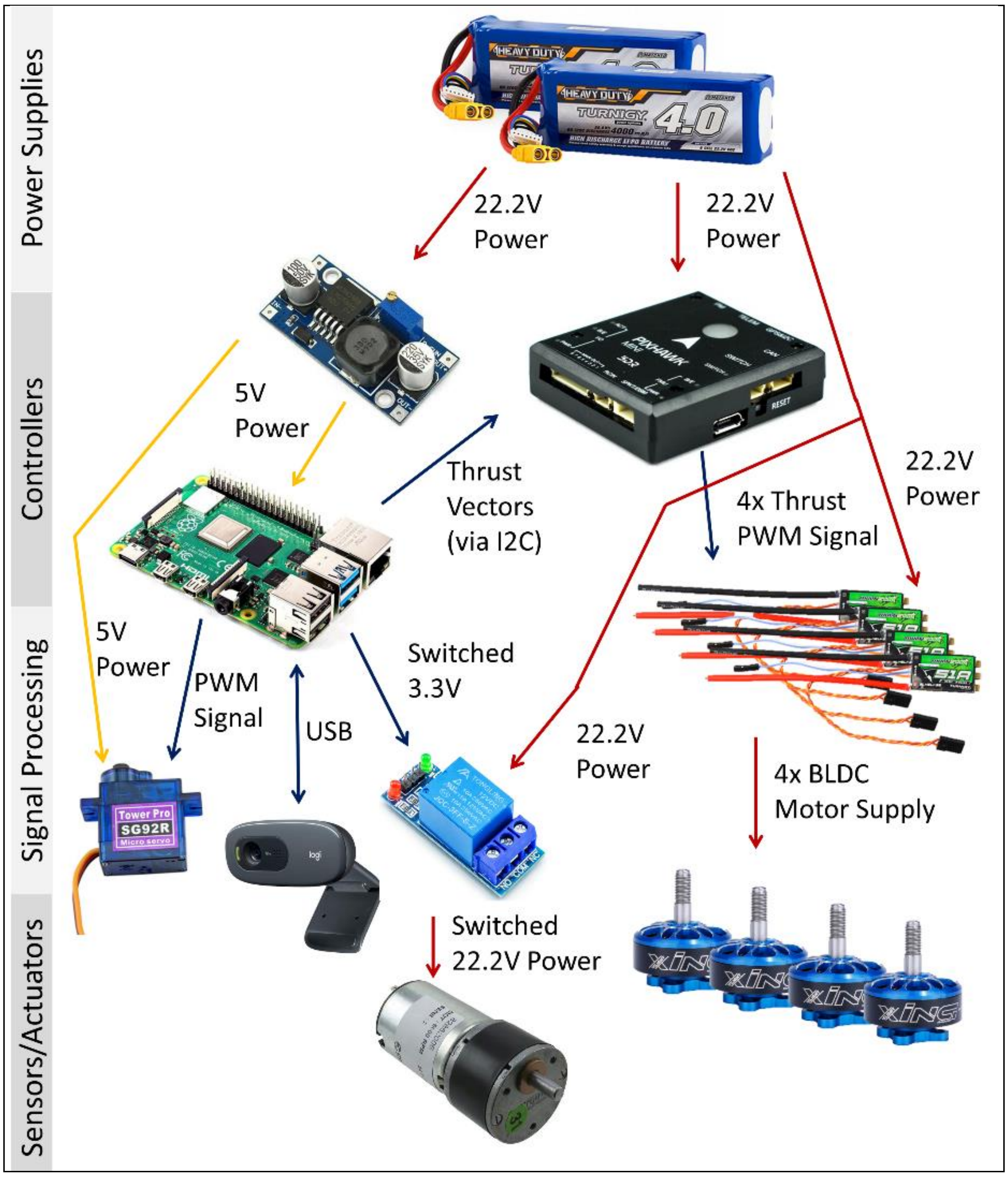


Figure 3.4 – Diagram of the final electrical system, showing connections between devices and the type of connection. Red lines show unregulated power supplied directly from the batteries (~22.2V), yellow lines show regulated power supplied at 5V, and blue lines show signal connections. Images taken from listing websites which are referenced in Table 3.1.

3.4.1 Endurance Analysis

To evaluate the endurance of the quadcopter, the power consumption of devices and the efficiency of the power supplies were considered while travelling at a constant velocity and in cleaning mode, and these are presented in Table 3.1. The power consumption of the propulsive motors was derived by linearly interpolating between thrust and power values specified by the manufacturer [25] [36], the ESCs were assumed to have an electrical efficiency of 0.8 based upon an approximate average of the findings of Gong and Verstraete [70], the 5V voltage regulator was assumed to have an electrical efficiency of 0.9 based upon its maximum efficiency of 0.92 [71], and the wiring and connectors were assumed to have an electrical efficiency of 0.98.

Given that the two 6S LIPO batteries have a capacity of 6000mAh at 22.2V, the quadcopter has a total of 178Wh of battery capacity. From this, the flight time in minutes was derived using Equation 3.1:

$$Flight\ Time = \frac{Total\ Wh\ capacity}{Average\ Total\ Power} \times 60 = \frac{178}{617} \times 60 = \mathbf{17.3\ \mathit{Minutes}} \qquad (3.1)$$

Therefore, the quadcopter exceeds the flight time requirement of 10 minutes. Given that the DJI Inspire 2, a heavier quadcopter with a smaller battery, has an average power of 436W and a flight time of 27 minutes [72], the power consumption of this quadcopter is greater than expected.

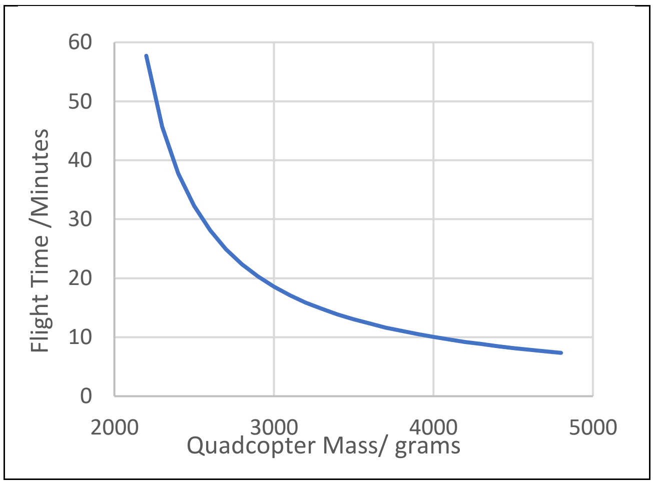


Figure 3.5 – Plot of flight time against quadcopter mass for this powertrain.

However, the data for these motors was provided by the manufacturer and is assumed to be reliable, therefore it may be suggested that the efficiency of the selected motors is significantly less than those used on the Inspire 2. This would benefit from a practical investigation where to determine the true power consumption of the motors and efficiency of the ESCs.

Figure 3.5 shows how flight time varies for as mass increases for this specific powertrain. Power data was extrapolated from the motors' specification sheet [36]. If the mass is decreased by 16% (500g), the endurance of the robot would be extended by 74%, thus showing the importance of minimising weight.

*Table 3.1 – A complete bill of materials, considering the masses and power draws of all the components. * Power draw based upon an estimate of electrical efficiency.*

	Type	Name	QTY	Total Avg. Power Draw/ W	Total Mass / g	Cost (each)	Cost (total)
Electrical Components	Battery	Turnigy 4000mAh 6S 60C [73]	2	-	1364	£52.07	£104.14
	Motors	iFlight XING-E 2306 1700kV [36]	4	856.0	132	£11.38	£45.52
	Flight Controller	Pixhawk Mini [74] [75]	1	2.5	12	£51.13	£51.13
	Navigation Computer	Raspberry Pi [68]	1	6.0	46	£34.00	£34.00
	ESCs	Turnigy MultiStar BLheli_32 ARM 51A [76]	4	171.2*	69	£16.07	£64.28
	PI PSU	Voltage Regulator [71]	1	1.2*	20	£1.99	£1.99
	Cleaning Brush Motor Relay [77]		1	0.5 *(est)*	18	£7.09	£7.09
	Cleaning Head Motor	Crouzet 82862006 [78]	1	3.9	160	£111.29	£111.29
	Squeegee servo	TowerPro SG92R [79]	1	3.3	9	£4.00	£4.00
	Camera	Logitech C270 [80]	1	2.5	75	£25.99	£25.99
	Wiring & Connectors	Estimate	1	20.9*	100 *(est)*	£10.00	£10.00
Mechanical Components	Propellors	HQ Prop 6045 [81]	4	-	20	£0.26	£1.03
	Cleaning Head	Custom 3D printed with brush bristles	1	-	400 *(est)*	£5.00	£5.00
	Drive Pully	Optibelt 32T Timing Belt Pully [82]	1	-	20	£8.11	£8.11
	Drive Belt	Optibelt 4 T2,5 / 420 [83]	1	-	10 *(est)*	£9.96	£9.96
	Brush Bearings	604_2RS [84]	2	-	4	£2.39	£4.78
	Fixings & Fasteners	Overall Assumed	1	-	50 *(est)*	£5.00	£5.00
	Elbow Pipe Fitting	4mm Tefen Elbow [85]	2	-	4 *(est)*	£1.25	£2.50
	Tee Pipe Fitting	4mm Tefen Tee [86]	3	-	6 *(est)*	£1.25	£3.75
	Pipe	4mm ID x 15M Pipe [87]	1	-	100 *(est)*	£4.25	£4.25
	Water	in 12M of pipe	1	-	150	£0.00	£0.00
	Chassis - Material	2mm Carbon Fibre Sheet 950X500 [88]	1	-	300	£153.30	£153.30
	Chassis - Adhesive	VM100 [89]	1	-	20 *(est)*	£5.70	£5.70
Totals				**1068 W**	**3090 g**		**£662.81**

4. Computer Vision Implementation

While localisation is not unique to this application and is a problem faced by many autonomous robots, the window mapping process is novel. Therefore, the decision was made to focus on the mapping processes, and the development and final implementation of these are discussed in this section.

4.1 Objectives

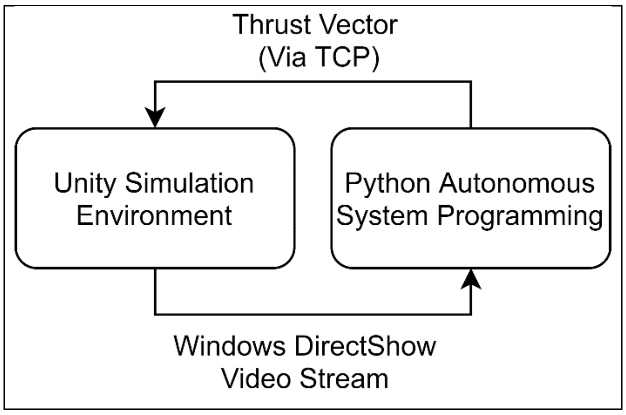
The second and third objectives listed in section 3.1 specify that the quadcopter's autonomous system must be able to 'Identify & Track Windows' and 'Generate a response to approach and clean the windows'. These objectives rely on the programming of the computer-vision system, and therefore this system is designed to these requirements.

4.2 Approach

As discussed in section 3.2, a number of computer vision libraries exist. OpenCV was chosen because it is a mature, open-source library that has backing from a number of large organisations like Intel, Google and Microsoft [90], and the fact that there are many resources regarding its implementation and use [91] [92].

Python was chosen as the development platform for the computer vision programming because it is highly portable, supported on many different platforms and is highly object-orientated meaning it has excellent scalability [93]. In addition, the OpenCV library is commonly used via its python implementation [94], meaning there are many community resources available.

4.3 Simulation Environment


Thrust Vector
(Via TCP)
Unity Simulation Environment
Python Autonomous System Programming
Windows DirectShow
Video Stream
Figure 4.1 – Data flow between the simulation environment and the autonomous control programming.

The 3D graphics engine Unity was used to simulate the physical environment for the quadcopter, and the camera output from this environment was streamed to the python programming via Windows DirectShow. This is the same way that a webcam is streamed to different programs on a Windows computer and therefore can be substituted for one. This video stream is the only data passed to the python programming from the simulation environment, which is analogous to a real implementation. Assets from a New York themed pack [95] were used to create the scene.

To imitate the transmission of thrust vectors from the Raspberry Pi to the PixHawk flight controller, Transmission Control Protocol (TCP) was used. Here, a TCP server was configured in the C# programming, and a TCP client was configured in Python. TCP was chosen as it is a connection-orientated and robust method of communication [96] that has readily available libraries which make setting up the server and client relatively straightforward [97] [98].

Figure 4.1 demonstrates the flow of data between the simulated environment and the autonomous system programming

4.4 Final Implementation

The final implementation uses Python 3.3.7 [99], running in real time via PyCharm Community 2020.1.4's built in interpreter [98]. Table 4.1 lists the libraries used in both the Python and C# programming and what they have been used for.

Table 4.1 – Libraries used in both the C# and Python Programming.

	Library Name	Version	Description
C#	*SimpleTCP* [97]	1.0.24	Allows quick and easy initialisation and management of a TCP server, used for thrust commands from the Python controller.
	System.Collections	Inbuilt in Unity 2020.3.27f1 [100]	Allows use of coroutines and other functions.
	UnityEngine		Required for scripts to work with Unity.
Python	*opencv-contrib-python* [94] [1]	3.4.17.61	The OpenCV library for python with additional modules.
	numpy [101]	1.21.5	Allows manipulation of data created using the opencv-python library.
	keyboard [102]	0.13.5	Enables keyboard input to be used.
	socket	Inbuilt in PyCharm 2020.1.4 [103]	Used for to setup a TCP client and send thrust data to the server running in Unity.
	csv		Used for reading and writing the settings on each launch and close of the programme.
	exists (from os.path)		Used to check that settings file exists before opening.
	time		Used to manage timeouts on window objects (during object tracking).

A video demonstration of the final implementation is available online at: https://www.youtube.com/watch?v=9L-0Ks-nzPc

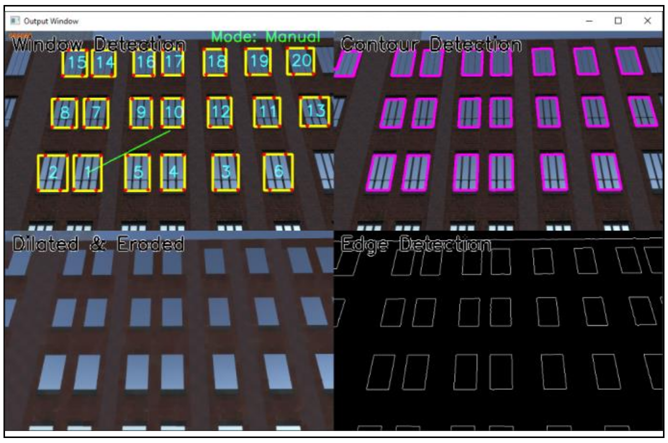


Figure 4.2 – The output window showing the real time images at various stages in the computer vision process. The top right output shows the initial image overlayed with the final window tracking and force vectoring line.

The functional aspects of the computer vision programme are split into six distinct stages:

1. ***Frame Acquisition***: The stream is created using the UnityCapture library [104], and is opened using the OpenCV library in Python [94]
2. ***Frame processing***: Each frame is taken individually and the following steps are carried out on them to prepare and simplify the image ready for window detection:
    a. Cropping (numpy function)– Unwanted areas are removed from the image. This is done first to save on processing power.
    b. Dilating (OpenCV Function) – A morphological filter is used with a square kernel, the size of which is defined by the user, to remove any small border regions in the image (e.g. window frames). The number of iterations is also defined by the user.
    c. Eroding (OpenCV Function)– A second morphological filter with the same size of kernel and number of iterations is used to counter-act some of the enlargement effects of the dilating operation.
    d. Greyscale Conversion (OpenCV Function) – The image is converted into greyscale. This step is carried out after dilation and erosion to ensure boundaries between different colours of similar brightness are not lost.
3. ***Window Recognition***: The following steps are carried out to find the windows in each frame:
    a. Canny Edge Detection (OpenCV Function) – Finds all the edges of an image based on Canny's algorithm [105, 106]. Upper and lower threshold values are defined by the user.
    b. Contour Detection (OpenCV Function) – Finds all the closed contours in the binary edge-detected image. This effectively filters out all the open contours.
    c. Vertex Detection (OpenCV Function) – Approximately locates all the vertices of the closed contours using the Douglas-Peucker algorithm [107].
    d. Vertex Filter – Uses the fact that most buildings have rectangularly-

shaped windows to filter any shapes that do not have four vertices.

   e. Dimension Filters – Minimum and maximum window sizes can also be defined by the user to prevent the programme from assuming any very large or very small objects are windows.

4. ***Window Tracking***: Tracked window objects are stored in a custom *window* class which are identified by unique ID numbers, and these numbers are overlayed on the output to show how the programme recognises that windows are unique and where it fails to track them. The following processes are carried out for each frame, noting that a list of tracked window objects is stored between frames:
   a. Firstly, the last update time of each of the tracked windows is checked, and any windows that have exceeded the timeout are discarded. This saves memory and processing power, and prevents old windows from re-appearing.
   b. Then the midpoints of all the recognised windows in the current frame are calculated and are each compared the last known location of all the tracked windows. If the difference in midpoints is less than a threshold value, and the tracked window hasn't already been assigned, the window in the current frame is assumed to be the same as the tracked window. If the tracked window has already been assigned, the next closest window is evaluated.
   c. If there are no tracked windows within the threshold distance, it is assumed that the window has appeared in this frame, and a new tracked window object is created.
5. ***Thrust Vectoring***: When in auto mode, the programme calculates the difference between the centre of the image and the position of the window with the lowest ID number, and applies a translational thrust vector with the aim of centring the window on the frame. Once the centre of the window lies within 10 pixels of the centre of the frame, the programme will apply thrust such that the quadcopter approaches the window whilst maintaining its central position. When the quadcopter is so close that the window is too large to be recognised with the current settings, the thrust vectoring is stopped. At this stage, a further cleaning algorithm would need to be implemented. A line is drawn on the output window to show the direction of the vector and this turns red when in auto mode.
6. ***Data Output & User Interface***: A control panel and video output window are presented to the user, on which the detected windows are highlighted and numbered in real time. This allows the user to fine-tune the settings for the specific site which are saved in a CSV file between executions of the programme.

Figure 4.2 contains a capture of the output window.

4.5 Simulation Testing

4.5.1 Tracking & Occlusion

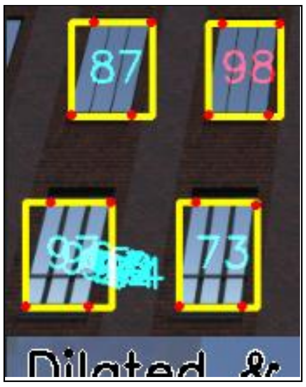


Figure 4.3– Image showing the failure of the program to track the bottom left window.

While at low speeds, tracking generally works very well, at higher speeds the programme encounters issues tracking the objects between frames. Figure 4.3 demonstrates this failure, showing how the window is assigned a new ID each frame. The old IDs have not yet timed out so they remain on the screen for a few hundred milliseconds, hence why they can be seen here. This is due to the midpoint-tracking nature of the program where a window is only tracked between frames if it lies within a certain threshold. While this threshold can be increased, it results in unreliable assignment as the IDs can swap with the surrounding windows.

In addition, the problem of Occlusion presents a real issue as once a window object leaves the screen and times out, it cannot be retrieved. This would prevent the drone from following a cleaning path as all other windows would move out of view when approaching a window.

One possible solution to resolving both the occlusion and tracking issues would be to use a tree data structure for all the windows so that their relative positions are recorded. For example, if the programme knew that window 98 was above window 73 in Figure 4.3, it would be able to re-assign it the same ID after occlusion. From this, it would likely be possible to reconstruct the entire map of windows using only one window.

2.5.2 Thrust Vectoring

The current thrust vectoring technique applies thrust proportionally to the distance from the centre of the screen. While this means that the drone does eventually reach its target providing the tracking is not lost, it does result in overshoot. To rectify this, a closed-loop controller could be implemented to critically damp the system. In addition, the use of an accelerometer may allow for the mapping of movement within the scene, which could assist the window tracking process. For example, if the drone knew how far it was away from the window, it would be able to fine-tune the settings to recognise windows at that distance.

2.5.3 Performance

When running on an i5-7300HQ with 12GB of ram and a Nvidia GTX1050M, the computer vision programme performs well at low speeds, but begins to run slowly when moving quickly. This may be due to the simulation environment running in Unity's interpreter simultaneously, as when using a video as the input (shown in section 4.5), the programme performs consistently well. This indicates that the programme may be able to run on a relatively low-power Raspberry Pi 4, as specified in the design.

4.6 Real-World Testing

Figure 4.5 contains an image of the output window when the input to the CV programming was changed to a video taken from a drone [2]. This scene was chosen as the weather is overcast, the building has a consistent size and shape of window, and the movement is smooth and continuous.

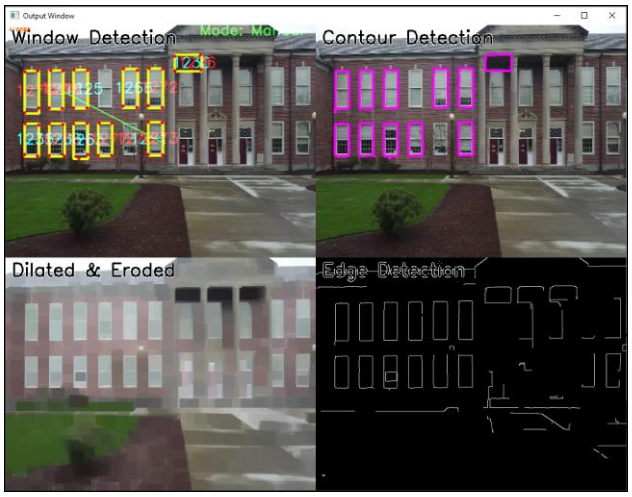


Figure 4.5 – The output window when the CV programme is fed a real video [2] from a drone.

Although the programming does recognise and track windows, it does so very poorly and unreliably even after tweaking the settings. This is to be expected because the real world is much more complicated and is more detailed than the simulated environment, therefore the window recognition process is much less straightforward.

Interestingly, the blinds behind the windows had little effect on the window recognition performance, however there were a number of areas that were incorrectly identified as windows and some windows that were not recognised whatsoever.

As a result of the poor window recognition performance, the window tracking suffered. This is because if the window is not detected within the timeout period, it is discarded and therefore the tracking is lost.

To improve window recognition performance, a mask based on the geometry of the environment could be used. This would mean that the surrounding pavement, landscape and other buildings could be ignored, and then the settings could be adjusted accordingly. While this would likely require the environment to be scanned, it could be done so only once before the first cleaning operation, then stored in memory and retrieved every time the drone is cleaning, as buildings do not change regularly.

5. Conclusion

5.1 Achievements

Through the evaluation of existing designs and literature [11, 3, 14, 60, 40, 35, 9, 6, 8, 7], a high level design specification has been developed for an autonomous window cleaning quadcopter. From this, detail designs of the mechanical elements and electrical systems of the robot have been created, a full bill of materials with cost and weight breakdowns has been provided, and a full 3D CAD model has been created in SolidWorks. Further evaluations of weight and endurance of the final robot have been conducted by comparing the robot to a similarly sized commercial product [72].

In addition, requirements for the autonomous system have been specified, and a 3D real-time simulation environment for the drone has been created using Unity3D and C#. Using this simulation, a computer vision programme to map and track windows in a scene has been developed using the OpenCV library in Python. This implementation is presented, and its performance is evaluated when used in both simulated and real environments. Areas for further development are highlighted, and potential solutions to problems are suggested.

5.2 Discussion

It was found that by using a quadcopter locomotion system, the robot can meet or exceed all of its requirements while being versatile, accessible and affordable. In addition, an axially orientated rotating drum brush was found to be the simplest and most effective implementation of a cleaning system, noting the use of a tethered water supply to loosen and wash away debris. While the use of a tethered power supply would be desirable, it was found to be impractical in this scenario due to the power requirements of the propulsive system and lack of commercially available products.

The dimensions of the quadcopter were limited by the size of conventional building windows and cost targets. Consequently, a 250mm chassis with 6” propellors and 2306 motors were designed and selected based upon a weight analysis of the drone, and a 270mm wide cleaning head was designed. In addition, the inclination required to apply 1kg equivalent of force was found to be around 20^o from a simple rigid body model based upon a 1kg normal force and assuming a coefficient of friction of 0.6 [38].

Moreover, the control systems were designed to make use of a Raspberry Pi 4 and a PixHawk mini flight controller. The Pi 4 was selected because of its relatively high power, small size, low mass, and excellent flexibility, while the use of a dedicated flight controller ensures that the safety of the device is not compromised by the CPU-intensive computer vision programming.

While this design has been thoroughly considered, no part of the robot has been manufactured. Consequently, its performance is unknown and no conclusion can be drawn about the true effectiveness of the implementation.

However, the computer vision programming, used for the mapping and tracking of windows, has had some testing in a simulated environment and with a pre-recorded video [2]. Here, it was found that after fine-tuning the settings to the specific scene, the programme can recognise windows relatively reliably but is less effective at tracking them between frames. In addition, the programme does not track windows after they have been occluded and occasionally mis-identifies them. This is because the implementation, which finds the midpoint of each window in the current frame and compares them to the previous frame, does not keep track of where the windows are relative to one another. Implementing a tree data structure to determine the positions of windows relative to each other is suggested as a method that would likely fix these misidentification and occlusion problems.

When used on the pre-recorded video [2], the CV programme performs reasonably well but does face the same issues as in the simulation but to a greater extent. This is expected as the greater amount of detail, shadows and depth in reality makes the scene trickier to digitally analyse. A technique using a 3D scan of the building to mask the surrounding environment is suggested to minimise the number of false window recognitions. Further work on the SLAM technique is required as the current CV implementation is limiting and would not be able to autonomously clean windows in its current state.

Overall, the design process for the CV implementation was somewhat effective, as a functioning solution is presented, however the approach taken may need to be re-considered or expanded using the findings of this report.

5.3 Conclusions

While the mechanical design aspects of the drone remain untested, the implementation is assumed to be somewhat appropriate. The difficulties associated with a tethered power supply forced the use of batteries, and this limits the flight time to around 17 minutes which is less than ideal. The CV programme developed has proved effective at recognising windows in both the simulated and real environments, while it’s window tracking algorithm is less robust. Further work is required to develop these algorithms, and a change in approach may be necessary. Altogether the CV programming has proved the concept of using OpenCV and Python for window mapping.

5.4 Future Work

The mechanical systems of the drone would benefit from strength analysis and optimisation using finite element methods in an effort to reduce weight, while the electrical and control systems could be simulated. Afterwards, prototyping of the drone and measuring its weight, power consumption and the friction coefficient between the brush and windows would prove useful in determining the accuracy of calculated data.

In addition, the CV programme used to recognise and track windows requires significant further development to improve its effectiveness and reliability, and additional algorithms need implementing for the cleaning mode of the drone. This may be done using the relational approach suggested in the report, or by another method. Moreover, the localisation approach would need to be determined and implemented, perhaps using a stereo camera system as suggested.

Once all the above tasks have been conducted, the whole drone can be tested in a real environment, and a critical evaluation of its design may be conducted.

6. References

[1] OpenCV, *opencv_contrib,* Unkown: GitHub, 2022.

[2] S. Pierce, *Footage Of Robert A Long High School Facade,* Unkown: Pexels, 2021.

[3] Aerones, *Innovation in Wind Turbine Blade Maintenance,* Riga: Aerones, Unknown.

[4] S. A. Sulaiman, A. K. Singh, M. M. M. Mokhtar and M. A. Bou-Rabee, *Influence of Dirt Accumulation on Performance of PV Panels,* Unknown: Elsevier Ltd, 2014, p. 54.

[5] Airbus, *A320 Family performance retention and fuel savings,* Blagnac: Airbus, 2008, p. 46.

[6] R. Siegwart, I. R. 1.-. Nourbakhsh and D. Scaramuzza, *Introduction to Autonomous Mobile Robots,* Cambridge, Mass.: MIT Press, 2011.

[7] N. Imaoka, S.-g. Roh, N. Yusuke and S. Hirose, *SkyScraper-I: Tethered Whole Windows Cleaning Robot,* Taipei: IEEE, 2010.

[8] Serbot, *GEKKO Facade Hightec Robot,* Switzerland: Serbot, Unknown.

[9] Welbot Technology, *SMART External Wall Cleaning Service,* Hong Kong: Welbot Technology, Unkown.

[10] S. Bouabdallah, P. Murrieri and R. Siegwart, *Design and Control of an Indoor Micro Quadrotor,* New Orleans: IEEE, 2004.

[11] Aerones, *Aerones Drone for High-Altitude Building Cleaning,* Riga, 2018.

[12] K. Sangpradit, *Study of the solar transmissivity of plastic cladding materials and influence of dust and dirt on greenhouse cultivations,* vol. 56, Pathum Thani: Elsevier, 2014.

[13] O. Liang, *How to Choose FPV Drone Motors,* Unkown: OscarLiang.com, 2019.

[14] J. Brown, *Quadcopter Motors: Understanding The Driving Force of The Drones,* Unkown: Jack Brown, 2020.

[15] Solar WInd, *DC Cable Sizing Tool,* Unkown: Solar Wind, Unkown.

[16] Royal Society of Chemistry, *Copper,* Unkown: Royal Society of Chemistry, Unkown.

[17] Unmanned Tech, *Axis AF227 2207 Brushless FPV Motor,* Unkown: Unmanned Tech, Unkown.

[18] Hobby Wing, *XRotor Race Pro 2207,* Unkown: Hobby Wing, Unkown.

[19] RC Life, *Hobbywing XRotor 2207 race pro 2450KV v1,* Unkown: RC Life, Unkown.

[20] EngineerX, *BrotherHobby Returner R4 2206-2300KV Thrust Tests,* Unkown: YouTube, 2016.

[21] Build Your Own Drone, *Brotherhobby Returner R4 2206 2300KV Motor,* Unkown: Build Your Own Drone, Unkown.

[22] EngineerX, *GepRC GR2207.5-2400KV Static Thrust Tests & Overview,* Unkown: YouTube, 2019.

[23] Cheap Drone, *GEPRC GEP-GR2207.5 2207.5 1700/1920KV 6S 2400/2750KV 4S Brushless Motor CW Thread for RC Drone FPV Racing,* Unkown: Cheap Drone, Unkown.

[24] Unmanned Tech, *T-Motor Velox V2 2306 Motor (1950kV, 2400kV) 4-6S,* Unkown: Unmanned Tech, Unkown.

[25] Unmanned Tech, *iflight xing e-2306-2 6s brushless motor 1700kv-2450kv,* Unkown: Unmanned Tech, Unkown.

[26] B. P. N, *Logitech c270 Webcam,* Unkown: GrabCAD, 2019.

[27] P. Maroli, *Pixhawk Mini,* Unkown: GrabCAD, 2017.

[28] S. S. GUNDA, *SERVO MOTOR,* Unkown: GrabCAD, 2022.

[29] Craftedkwads, *2306 motor model,* Unkown: GrabCAD, 2021.

[30] m. elfakharany, *Drone blade,* Unkown: GrabCAD, 2021.

[31] Optibelt, *CAD Service Tool,* Unkown: Optibelt, 2022.

[32] t. hdp, *raspberry pi 3-B case,* Unkown: GrabCAD, 2021.

[33] e. vaidya, *emax esc,* Unkown: GrabCAD, 2019.

[34] J. A. G. Canales, *LIPO BATTERY,* Unkown: GrabCAD, 2018.

[35] P. Pounds and R. Mahony, *Design Principles of Large Quadrotors for Practical Applications,* Canberra: IEEE, 2009.

[36] Drone Authority, *iFlight XING-E 2306 2-6S Brushless Motor,* Unkown: Drone Authority, 2022.

[37] M.-h. Hwang, H.-R. Cha and S. Y. Jung, *Practical Endurance Estimation for Minimizing Energy Consumption of Multirotor Unmanned Aerial Vehicles,* Gwangju: MDPI, 2018.

[38] R. Holopainen and E.-M. Salonen, *Modelling Bristle Behaviour in Rotating Brush Duct Cleaning,* Helsinki: Otamedia, 2003.

[39] DigiKey Electronics, *Crouzet 82862006,* Unkown: DigiKey Electronics, 2022.

[40] A. D. L. ESCALERA, L. MORENO, M. A. SALICHS and J. M. ARMINGOL, *Continuous mobile robot localization by using structured light and a geometric map,* vol. 27, Unkown: Taylor & Francis, 1996, pp. 771-782.

[41] D. Ribas, J. Neira and P. Ridao, *Underwater SLAM for Structured Environments Using an Imaging Sonar,* Berlin: Springer, 2010.

[42] Unkown, *Radio Detection and Ranging,* vol. 152, London: Nature, 1943, pp. 391-392.

[43] M. I. Skolnik, *Radar Handbook,* New York & London: McGraw-Hill, 1970.

[44] R. T. H. Collis, *Lidar,* 8 ed., vol. 9, California, 1970.

[45] T. S. Taylor, *Introduction to Laser Science and Engineering,* Boca Raton: CRC Press, 2019.

[46] A. D. Waite, *Sonar for Practising Engineers,* 3 ed., John Wiley & Sons, 2002.

[47] M. Cazorla and D. Veijo, *JavaVis: An Integrated Computer Vision Library for Teaching Computer Vision,* Alicante: Wiley, 2013.

[48] R. Fisher, S. Perkins, A. Walker and E. Wolfart, *Khoros,* Albuquerque: Unkown, 2003.

[49] OpenCV Team, *About,* San Jose: OpenCV, 2022.

[50] U. Köthe, *The VIGRA Computer Vision Library Version 1.11.1,* Heidelberg: University of Heidelberg, 2017.

[51] CVIPtools Dev Team, *Welcome to CVIPtools at SIUE,* Illinois: Southern Illinois University Edwardsville, 2022.

[52] L. Cruz, D. Lucio and L. Velho, *Kinect and RGBD Images: Challenges and Applications,* 25 ed., Rio de Janeiro: IEEE, 2012.

[53] S. Naudet-Collette, K. Melbouci, V. Gey-Bellile, O. Ait-Aider and M. Dhome, *Constrained RGBD-SLAM,* vol. 39, Gif-Sur-Yvette: Cambridge University Press, 2020, pp. 277-290.

[54] Intel, *Intel® RealSense™ Depth Camera D435,* Unkown: Intel, 2022.

[55] S. Izadi, R. A. Newcombe, D. Kim, O. Hilliges, D. Molyneaux, S. Hodges, P. Kohli, J. Shotton, A. J. Davison and A. Fitzgibbon, *KinectFusion: Real-Time Dynamic 3D Surface Reconstruction and Interaction,* Vancouver: SIGGRAPH, 2011.

[56] iRobot, *Roomba,* Unkown: iRobot, 2022.

[57] B. Tribelhorn and Z. Dodds, *Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education,* Rome: IEEE, 2007.

[58] A. El-Rabbany, *Introduction to GPS: The Global Positioning System,* Boston/London: Artech House, 2002.

[59] D. J. Hall, *Robotic Sensing Devices,* Pittsburgh: Carnegie-Mellon University, 1984.

[60] J. Choi, M. Kim, J. Kim and W. Lee, *Designing an Interactive Indoor Delivery Robot and Its Implication,* Unkown: Springer, 2022.

[61] A. L. Majdik, D. Verda, Y. Albers-Schoenberg and D. Scaramuzza, *Air-ground Matching: Appearance-based GPS-denied Urban Localization of Micro Aerial Vehicles,* 7 ed., vol. 32, Zurich: IEEE, 2015.

[62] T. Leroux, S.-H. Ieng and R. Benosman, *Event-Based Structured Light for Depth Reconstruction using Frequency Tagged Light Patterns,* Pittsburgh: ResearchGate, 2018.

[63] M. Gupta, A. Agrawal, A. Veeraraghavan and S. G. Narasimhan, *Structured Light 3D Scanning in the Presence of Global Illumination,* Colorado Springs: IEEE, 2011.

[64] Intel, *Intel® RealSense™ Depth Camera D455,* Unkown: Intel, 2022.

[65] M. Anandhalli and V. P. Baligar, *A novel approach in real-time vehicle detection and tracking using Raspberry Pi,* Hubbali: Elsevier, 2016.

[66] I. Iszaidy, R. Ngadiran, R. B. Ahmad, M. I. Jais and D. Shuhaizar, *Implementation of raspberry Pi for vehicle tracking and travel time information system: A survey,* Unkown: IEEE, 2016.

[67] N. Wanluk, S. Visitsattapongse, A. Juhong and C. Pintavirooj, *Smart wheelchair based on eye tracking,* Bangkok: IEEE, 2016.

[68] The Pi Hut, *Raspberry Pi 4 Model B,* Unkown: The Pi Hut, 2022.

[69] PX4 Autopilot, *PX4 User Guide,* Unkwon: PX4, 2022.

[70] A. Gong and D. Verstraete, *Development of a dynamic propulsion model for electric UAVs,* Cairns: Research Gate, 2015.

[71] Hobby Components, *LM2596 DC-DC 3-35V adjustable step-down power Supply module,* Unkown: Hobby Components, 2022.

[72] DJI, *Inspire 2 Specs,* Unkown: DJI, Unkown.

[73] HobbyKing, *Turnigy Heavy Duty 4000mAh 6S 60C Lipo Battery Pack w/XT90,* Unkown: HobbyKing.com, 2022.

[74] S. Dade, *Pixhawk (and APM) Power Consumption,* Unkown: DIY Drones, 2015.

[75] U. Tech, *Pixhawk Mini Board,* Unkown: Unmanned Tech, 2022.

[76] Hobby King, *Turnigy MultiStar BLheli_32 ARM 51A Race Spec ESC 2~6S (OPTO),* Unkown: Hobby King, 2022.

[77] Amazon UK, *1 Channel Relay Module, 5V 1 Channel Relay Module Relay Board with Optocoupler Low Level Trigger Expansion Board for Arduino 5V/12V/24V(5V),* Unkown: Amazon UK, 2022.

[78] DigiKey Electronics, *Crouzet 82862006,* Unkown: DigiKey Electronics, 2022.

[79] The Pi Hut, *TowerPro Servo Motor - SG92R Micro,* Unkown: The Pi Hut, 2022.

[80] Logitech, *C270,* Unkown: Logitech, 2022.

[81] Hobby King, *Diatone 6045 Plastic Self Tightening Propellers 6 x 4.5 (CW/CCW) (Orange) (2 Pairs),* Unkown: Hobby King, 2022.

[82] RS Components, *OPTIBELT Timing Belt Pulley, Aluminium 4 mm, 6 mm Belt Width x 2.5mm Pitch, 32 Tooth,* Unkown: RS Components, 2022.

[83] RS Components, *OPTIBELT 4 T2,5 / 420, Timing Belt, 168 Teeth, 420mm 4mm,* Unkown: RS Components, 2022.

[84] Simply Bearings, *Major Branded 6042RS Rubber Sealed Deep Groove Ball Bearing 4x12x4mm,* Unkown: Simply Bearings, 2022.

[85] Hortafix, *Tefen Elbow 4mm,* Unkown: Hortafix, 2022.

[86] Hortafix, *Tefen Tee 4mm,* Unkown: Hortafix, 2022.

[87] Hortafix, *4mm x 15M Black Pipe,* Unkown: Hortafix, 2022.

[88] Easy Composites, *High Strength Carbon Fibre Sheet,* Unkown: Easy Composites, 2022.

[89] Easy Composites, *VM100 Black 10min Methyl Methacrylate Adhesive,* Unkown: Easy Composites, 2022.

[90] OpenCV.org, *OpenCV,* Unkown: OpenCV.org, 2022.

[91] G. Bradski and A. Kaehler, *Learing OpenCV: Computer Vision with the OpenCV Library,* Sebastopol: O'Reilly, 2008.

[92] K. Dawson-Howe, *A Practical Introduction to Computer Vision with OpenCV,* Chichester: John Wiley and Sons Ltd, 2014.

[93] A. Speight, *Bite-Size Python: An Introduction to Python Programming,* Indianapolis: John Wiley & Sons, 2020.

[94] opencv, *opencv-python,* Unkown: GitHub, 2022.

[95] Geopipe Inc, *Real New York City Vol. 1,* Unkown: Unity Asset Store, 2022.

[96] J. B. Postel and L. L. Garlick, *Transmission Control Protocol Specification,* Menlo Park: Stanford Research Institute, 1976.

[97] B. Potter, *SimpleTCP,* Unkown: GitHub, 2017.

[98] Jet Brains, *PyCharm,* Unkown: Jet Brains, 2022.

[99] Python, *About,* Unkown: Python.org, 2022.

[100] Unity, *Manual,* Unkown: Unity, 2020.

[101] numpy, *numpy,* Unkown: GitHub, 2022.

[102] boppreh, *keyboard,* Unkown: GitHub, 2022.

[103] J. Brains, *PyCharm Features,* Unkown: Jet Brains, 2022.

[104] schelingb, *UnityCapture,* Unkown: GitHub, 2019.

[105] J. F. Canny, *Finding Edges and Lines in Images,* Massachusetts: MIT Articial Intelligence Laboratory, 1983.

[106] OpenCV, *Canny Edge Detection,* Unkown: OpenCV, 2022.

[107] D. H. Douglas and T. K. Peucker, *ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE,* 2 ed., vol. 10, Unkown: University of Toronto Press, 1973.

Appendices

A.1 Supervisor Meeting Log

Table A.1.1 contains the record of meetings with Chengxu Zhou, the supervisor of this project throughout the 2021/22 academic year. Figures A.1.2 – A.1.8 contain images of the work shared during these meetings.

Figure A.1.1 – Meeting Log Table

Time Date No.	Meeting Agenda	Progress Since Last Meeting	Key Notes/Actions from meeting
15:00 14/10/2021 1	- Discuss meeting plans for the year. - Evaluate some examples of previous robots. - Brainstorm ideas.	N/A - First Meeting	- Decide which idea to proceed with. - Evaluate different variations of the design and detail a full design concept with sensors and actuators specified.
10:00 19/10/2021 2	- Present and discuss the chosen robot concept.	- Robot idea decided upon and a full concept developed (window cleaning robot) - See Figure A.1.2.	- Determined that a quadcopter-based cleaning robot would be preferable. - Agreed that the robot needs to be able to clean separate windows rather than just a smooth glass façade. - Therefore a quadcopter-based concept must be developed.
10:00 26/10/2021 3	- Evaluate the latest quadcopter-based concept. - Discuss the key elements of the project, what is going to be focused on, and how the design process is going to be structured.	- A further quadcopter-based concept had been made, with thought given to the navigational and cleaning mechanisms - See Figure A.1.3.	- Determined that Lidar and QR codes are not necessary for the navigation system. - Discussed objectives of the project. - SolidWorks Mock-Up needed for next week, and bill of materials and approximate costs needed in 2 weeks.
10:00 02/11/2021 4	- Discuss some of the finer details, challenges and solutions.	- Initial SolidWorks mock-up developed. - Thought given to ethical considerations. - Literature reviewed.	- Determined that the chassis must be made from lighter materials to minimise weight. - Determined that an ethical evaluation from the university is not required for this project. - Determine the positioning of the navigation camera. - For next meeting, continue developing the SolidWorks model, compare cleaning methods and begin the literature review.

5	09/11/2021	10:00	- Show first prototype with bill of materials, costs and purchasing list.	- Additional work on SolidWorks model – See Figure A.1.4. - Further literature reviewed. - Scoping and Planning document plan.	- Discussed design elements and improvements. - Discussed design choices based upon reviewed literature and existing designs. - For next meeting, finish scoping and planning document for feedback.
6	23/11/2021	10:00	- Discuss scoping document and areas for improvement.	- Scoping and planning document close to final. - Consideration given to design choices.	- Feedback given on aim and scope of project to ensure designs can be justified. - Discussion of project title. - Layout of document considered. - For next meeting, continue with the mechanical design and the implementation of the sensors and actuators.
7	07/12/2021	9:30	- Discuss the practical implementation of the designs.	- Mechanical design close to finished. - Scoping and planning document submitted.	- Various design elements Discussed. - Decided to focus the project on design, simulation and analysis rather than making a prototype due to time and monetary constraints. - Agreed to finish the mechanical and electrical design by the next meeting.
8	28/01/2022	10:30	- Discuss and evaluate the final mechanical and electrical design. - Consider the approach moving forward.	- Design targets established. - Mechanical and electrical design finished – See Figure A.1.5. - Weight analysis conducted.	- Agreed to focus on the simulation of the quadcopter's autonomous system for the remainder of the project. - Decided upon using Unity as the simulation environment and OpenCV as the computer vision library. - For next meeting, have the simulation environment set up and start the computer vision programming to show a demonstration.
9	14/02/2022	2:10	- Demonstrate simulation environment. - Discuss window recognition approaches for the CV programming.	- Simulation environment set up. - Basic proof of concept computer vision programme made – See Figure A.1.6	- Agree to continue with the simulation environment. - Develop and implement a window detection algorithm to be demonstrated at the next meeting.
10	28/02/2022	2:10	- Demonstrate working simulation environment with window recognition. - Discuss window tracking implementation.	- Window detection full implementation – See Figure A.1.7. - Documentation of simulation environment started.	- Discussion of how the simulation can detect windows in each frame but how it cannot track them between frames. - For the next meeting, design a window tracking algorithm and implement it.

11 14/03/2022 2:10	- Look at simulation and plan moving forward. - Discuss final report plan and layout.	- Window tracking algorithm partially designed and implemented – See Figure A.1.8. - Final report plan made.	- Discussion of the difficulties of window tracking. - Agreed that the programme is at a reasonably acceptable level for a demonstration. - Further developments to the programme can be made but the report should be given priority.
Supervisor Signature: (all meetings)	(see signed meeting log – submitted separately)		

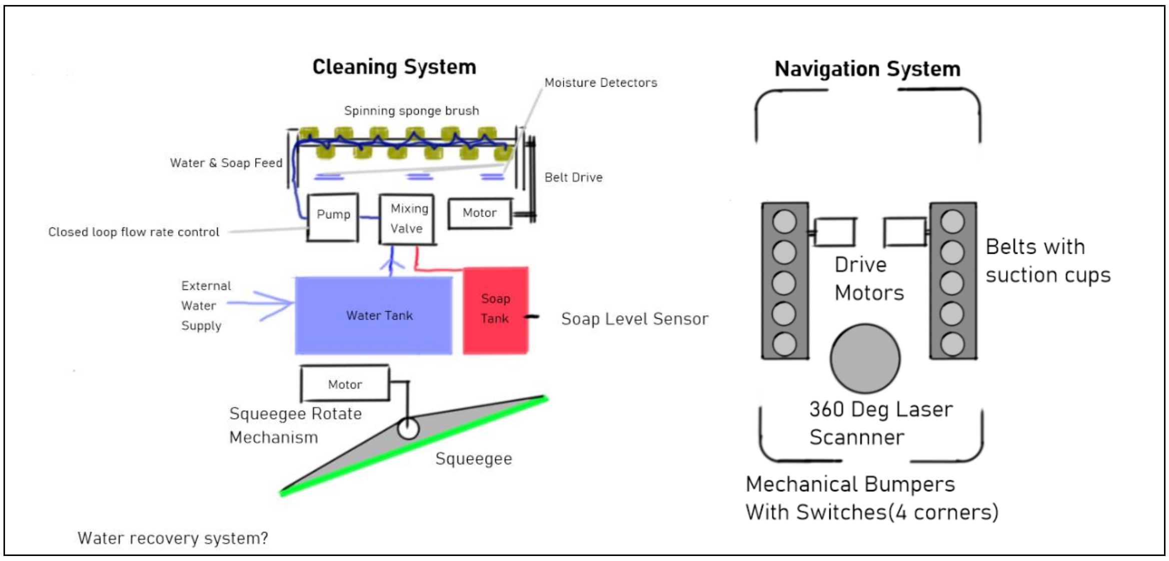


Figure A.1.2 – Initial Window Cleaning robot concept, shared in meeting 2

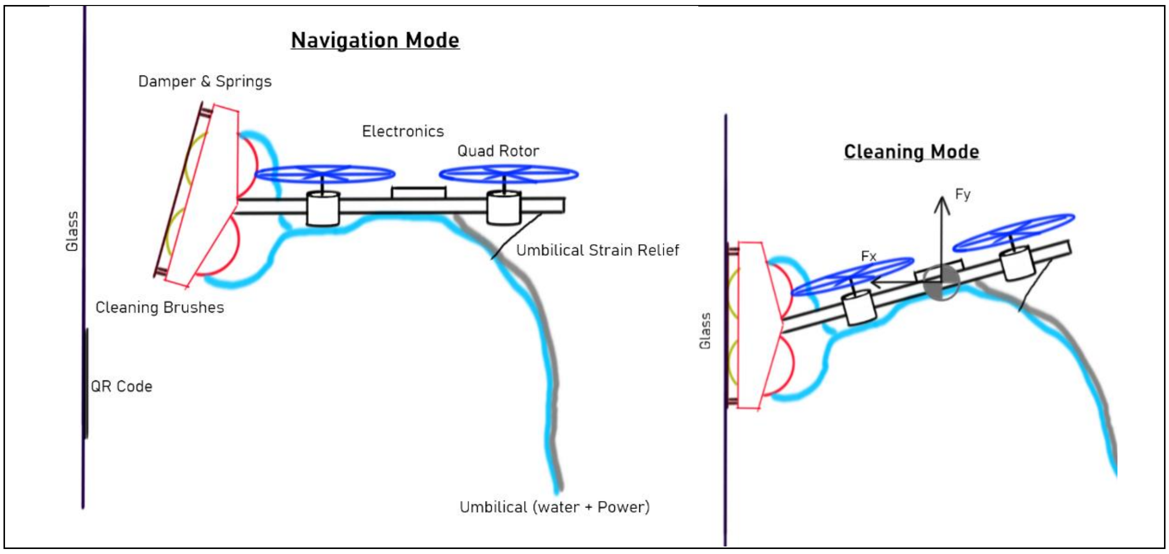


Figure A.1.3 – The chosen concept, shared in meeting 3.

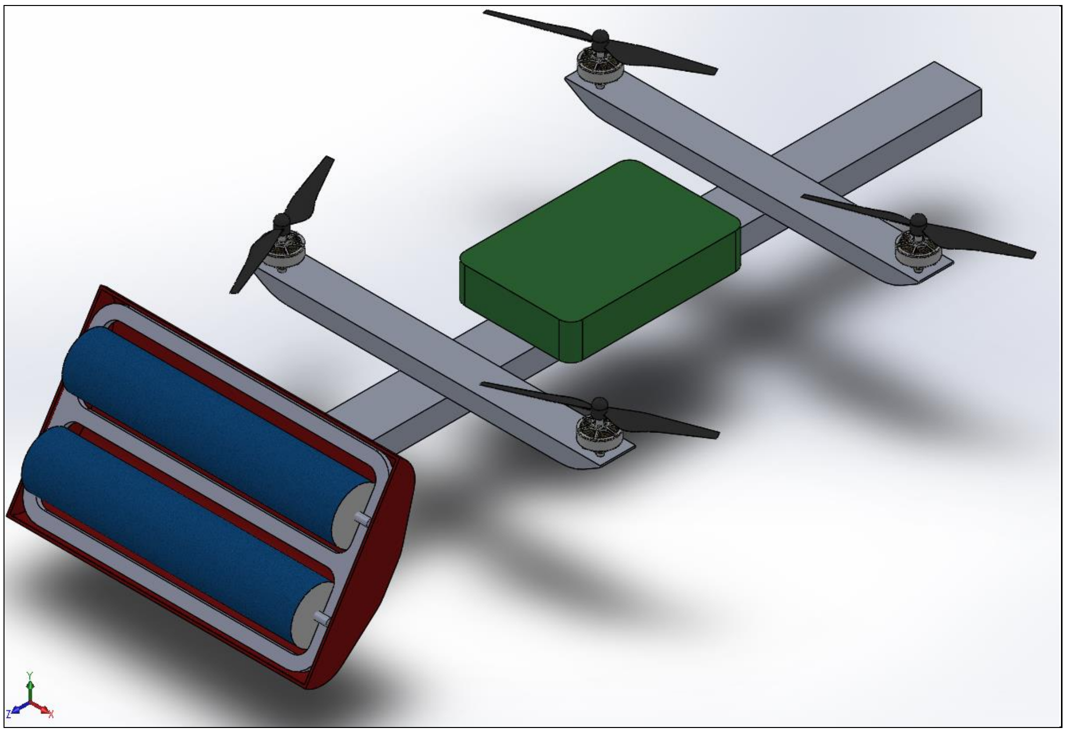


Figure A.1.4 – The initial SolidWorks mock-up of the robot design, shared in meeting 5.

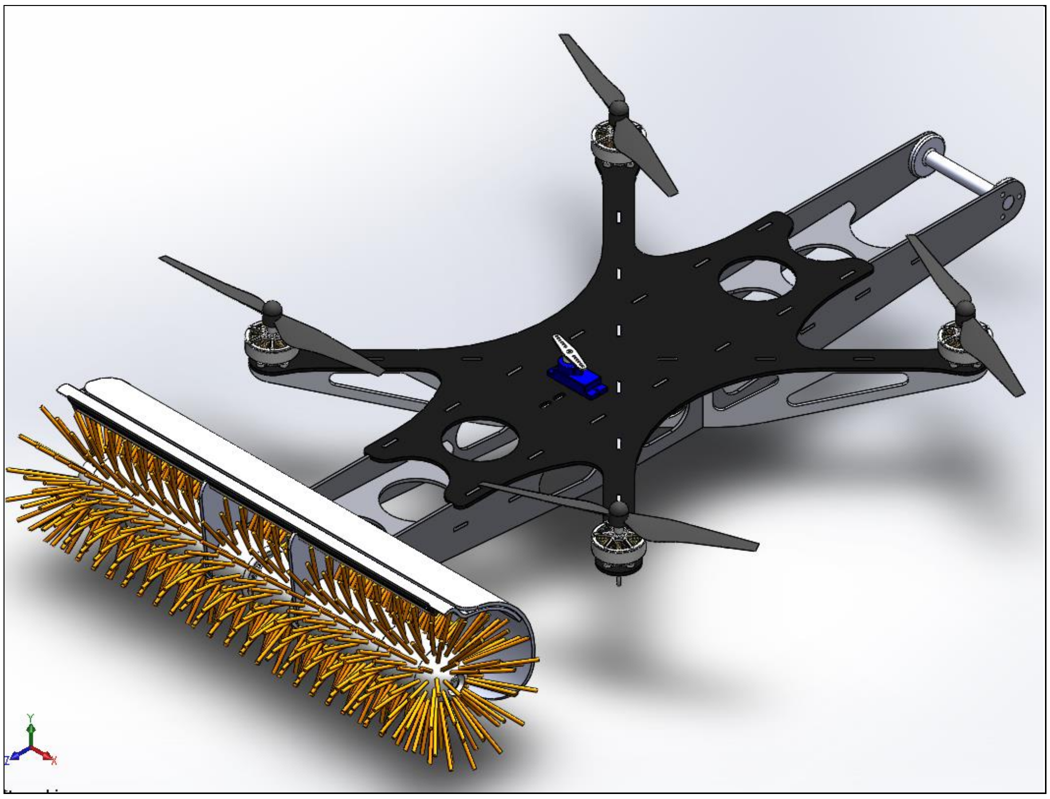


Figure A.1.5 – The final SolidWorks model of the window cleaning quadcopter, shared in meeting 8.

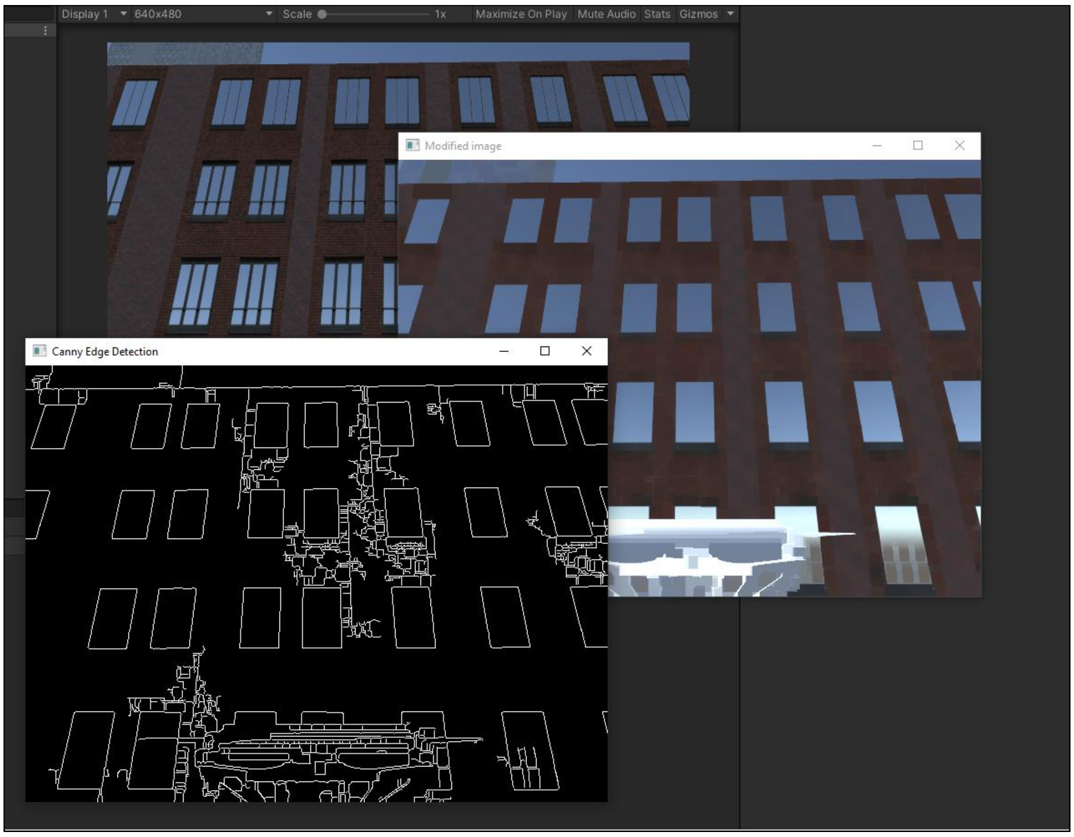


Figure A.1.6 – A demonstration of the 'proof of concept' simulation environment running in real-time, shared in meeting 9.

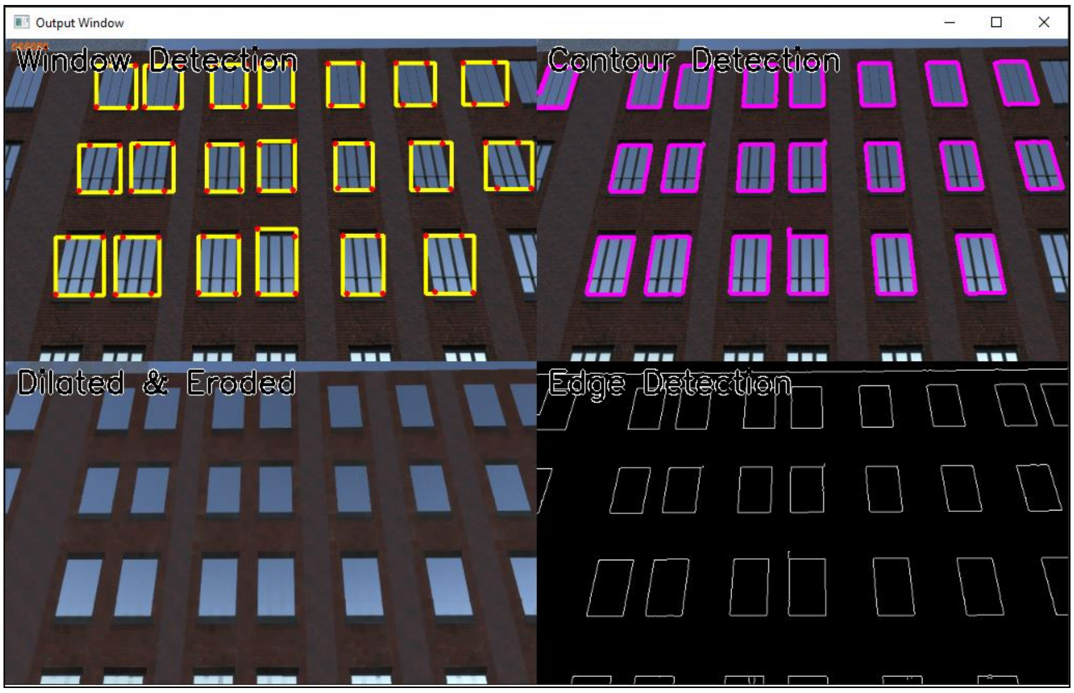


Figure A.1.7 – Image demonstrating the window recognition capabilities of the custom computer vision programming, shared in meeting 10.

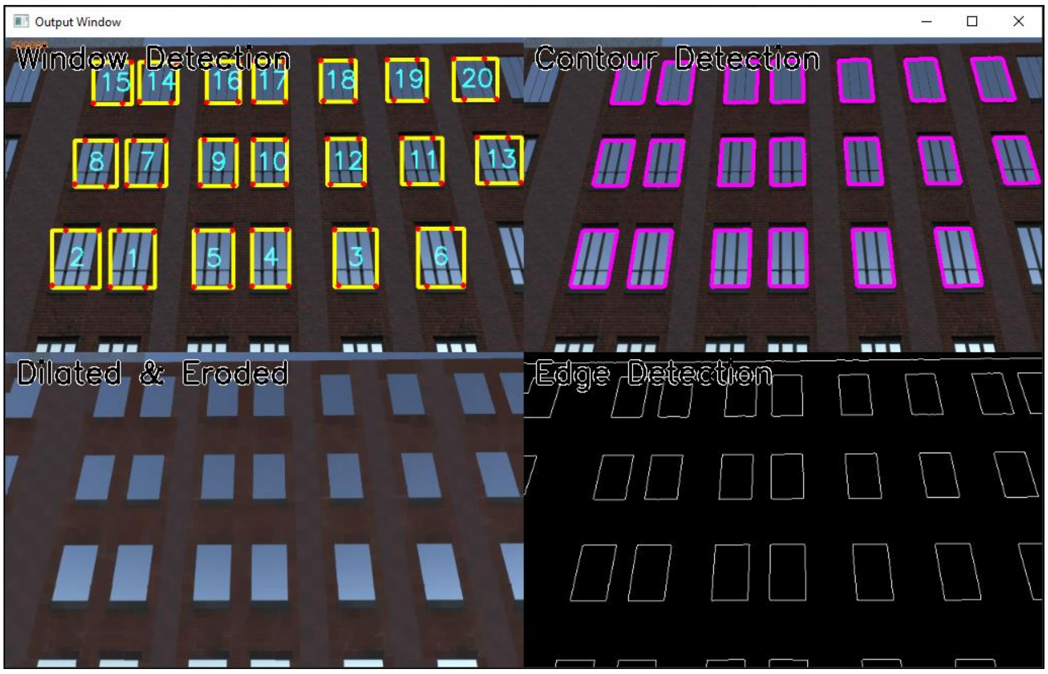
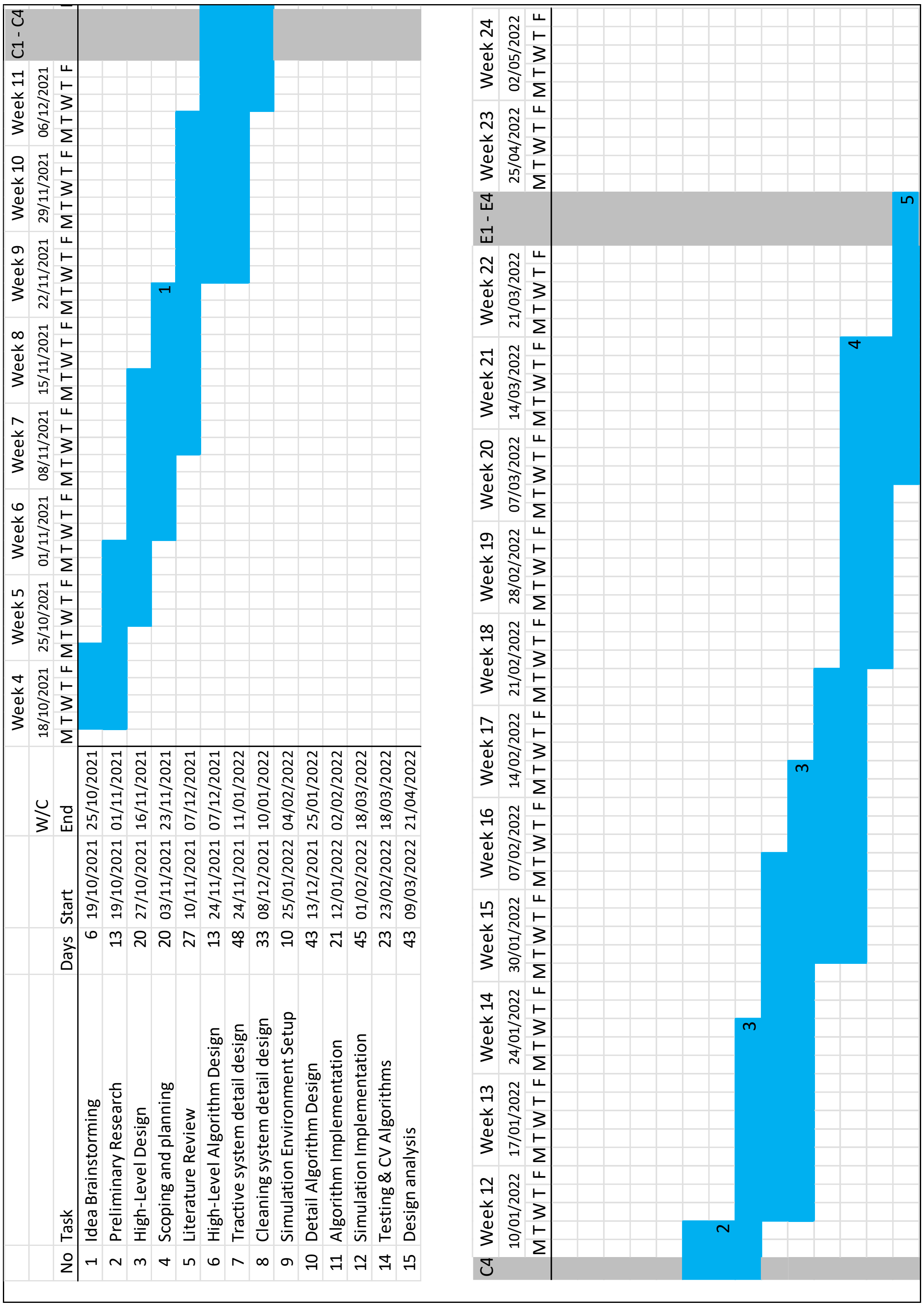


Figure A.1.8 – Image demonstrating the incomplete (at the time) implementation of the window tracking algorithm, shared in meeting 11.

A.2 Gannt Chart



			W/C	
No	Task	Days	Start	End
1	Idea Brainstorming	6	19/10/2021	25/10/2021
2	Preliminary Research	13	19/10/2021	01/11/2021
3	High-Level Design	20	27/10/2021	16/11/2021
4	Scoping and planning	20	03/11/2021	23/11/2021
5	Literature Review	27	10/11/2021	07/12/2021
6	High-Level Algorithm Design	13	24/11/2021	07/12/2021
7	Tractive system detail design	48	24/11/2021	11/01/2022
8	Cleaning system detail design	33	08/12/2021	10/01/2022
9	Simulation Environment Setup	10	25/01/2022	04/02/2022
10	Detail Algorithm Design	43	13/12/2021	25/01/2022
11	Algorithm Implementation	21	12/01/2022	02/02/2022
12	Simulation Implementation	45	01/02/2022	18/03/2022
14	Testing & CV Algorithms	23	23/02/2022	18/03/2022
15	Design analysis	43	09/03/2022	21/04/2022

Figure A.2.1 – Gantt Chart, split into two sections for clarity.

A.3 Computer Vision Programming Highlights

A.3.1 Window Class

```
class Window:
    def __init__(self, ID, VertPos):
        self.ID = ID  # A unique integer number
        self.VertPos = VertPos  # A list containing the last
known position of the four verticies on the screen
        self.LastUpdate = time.time()  # The absolute time at
which these parameters were set (in MS)
        self.mp = FindMidpoint(VertPos)
        self.available = False
        self.newInFrame = True
        self.newFrameCount = 0

    def UpdateVerts(self,VertPos):
        self.VertPos = VertPos  # A list containing the last
known position of the four verticies on the screen
        self.LastUpdate = time.time()  # The absolute time at
which these parameters were set (in MS)
        self.mp = FindMidpoint(VertPos)
        self.available = False

    def Refresh(self):
        self.available = True
        self.newFrameCount += 1
        global colourDelay
        if self.newFrameCount >= colourDelay:
            self.newInFrame = False
```

A.3.2 Window Recognition Function

```
def
findContours(imgCanny,img,contHips,contMinLength,contMaxLength):
#Function to separate out windows from an edge-detected image
    imgCanny = cv2.blur(imgCanny,(2,2))
    outConts = []
    image, conts, heigherarchy =
cv2.findContours(imgCanny,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMP
LE)
    imgConts = img.copy() # Copys the raw image to plot contours
on
    for cont in conts: # Iterates through all the contours in the
current image
        area = cv2.contourArea(cont)
        if area > contHips: #A "highpass" filter to remove
contours with very small areas due to noise.
            aLength = cv2.arcLength(cont,True)
            if aLength > contMinLength and aLength <
contMaxLength:
                #print(aLength)
                verticies = cv2.approxPolyDP(cont, 0.05*aLength,
True) #Finds the approximate verticies of all the contours
                cv2.drawContours(imgConts, [cont], -1, (255, 0,
255), 4)

                if len(verticies) == 4 : #If a contour has four
corners, draw a bounding box around it
                    x, y, w, h = cv2.boundingRect(verticies)
```

```
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),4)
                    cornerPoints = []
                    for vert in verticies:  #Draws circles around
each of the window's corners

cv2.circle(img,(vert[0][0],vert[0][1]),2,(0,0,255),2)
                        cornerPoints.append(vert[0])
                    region = [[x,y,w,h],cornerPoints]
                    outConts.append(region)
        pass

    return imgConts, outConts
```

A.3.3 Window Tracking Function

```
def trackWindows(Conts,windowList, MPtimeout,
MPclosenessThreshold, imgX, imgY):

    midpoint = []
    global autoMode
    for region in Conts:    # Identifies the centre point of each
contour and adds them to midpoints queue
        VertPos = region[1]
        mp = FindMidpoint(VertPos)
        midpoint.append(mp)    # Add all midpoints to a queue
        pass
    currentWindowList = []
    print("Number of Windows: ", len(windowList))
    for wind in windowList: # Adds all non-outdated windows to a
new array which is then used (to delete outdated windows)
        dt =  np.abs(time.time()-wind.LastUpdate)
        if (dt*1000) < MPtimeout:
            wind.Refresh()  # Resets the availability flag
            currentWindowList.append(wind)
    windowList = currentWindowList
    for point in midpoint:  # Iterates through all the midpoints
in the current image
        pntindex = midpoint.index(point)
        fl = FindMinFeatureLength(Conts[pntindex])
        MPcloseness = (MPclosenessThreshold*fl/100)
        mindiff = MPcloseness + 1 # Sets an initial value for
mindiff which is used when when the program starts
        diff = []
        i = 0
        for wind in windowList:    # Iterates through all the
previously tracked groups of midpoints
            thisdiff = np.abs((point[0]-wind.mp[0])+(point[1]-
wind.mp[1]))
            diff.append([i,thisdiff]) #Finds the overall
difference in points
            i += 1

        assigned = False
        if len(diff) >= 1:
            diff.sort(key=lambda y: y[1]) # Sorts the tuple by
the second element (diff value)
            i = 0
```

```
                for dif in diff:
                    if dif[1] < MPcloseness:
                        grpindex = dif[0]   # Finds the index of the
closest group
                        if windowList[grpindex].available: # if the
group is availabe to be assigned

windowList[grpindex].UpdateVerts(Conts[pntindex][1])   # Updates
the verticies in the window to the new location
                            assigned = True
                        #print("Assigned point
",midpoint.index(point),"to existing location")
                            break
                    else:
                        break
                    if i > (len(diff)-1):
                        break
                    else:
                        pass

            if not assigned:  # If there are no points within the
closeness threshold, a new group is created
                global windowID
                windowID += 1
                wind = Window(windowID,Conts[pntindex][1])
                windowList.append(wind)
                #print("Assigned window ", midpoint.index(point), "to
new location")
            pass


        overlay = np.zeros((imgX,imgY,3), np.uint8)
        overlay = cv2.cvtColor(overlay, cv2.COLOR_RGB2RGBA)
        i = 0
        for wind in windowList:
            textSize =
cv2.getTextSize(str(wind.ID),cv2.FONT_HERSHEY_SIMPLEX, 0.8,2)
#Gets the text size to allow the text to be centralised
            if wind.newInFrame:
                colour =  (0,0,255)
            else:
                colour = (255,255,0)
            cv2.putText(overlay,str(wind.ID),(wind.mp[0]-
round(textSize[0][0]/2),wind.mp[1]+round(textSize[0][1]/2)),cv2.F
ONT_HERSHEY_SIMPLEX, 1, colour,2)
            i += 1

        if autoMode:
            modeColour = (0,0,255)
            modeText = "Mode: Auto"
        else:
            modeColour = (0,255,0)
            modeText = "Mode: Manual"


cv2.putText(overlay,modeText,(400,20),cv2.FONT_HERSHEY_SIMPLEX,
1, modeColour,2)
        if len(windowList) >=1:
            mv, p1, p2 =
movePointToCentre(overlay,windowList[0].mp,FindMinFeatureLength(w
indowList[0].VertPos))
```

```
            cv2.line(overlay, p1, p2, modeColour,2)
        else:
            autoMode = False
            mv = (0,0,0)
        return overlay, windowList, mv
```

A.3.4 Main Function

```
def main():
    global autoMode
    cap = openUnityCapture()
    loadSettings()
    host = socket.gethostname()
    port = 34343  # The same port as used by the server
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    windowList = []
    manualControl = True
    while True:
        itns = cv2.getTrackbarPos("No. Iterations", "Controls")
        krnlX = cv2.getTrackbarPos("Kernal x", "Controls")
        krnlY = cv2.getTrackbarPos("Kernal y", "Controls")
        cannyL = cv2.getTrackbarPos("Canny low", "Controls")
        cannyH = cv2.getTrackbarPos("Canny high", "Controls")
        contHips = cv2.getTrackbarPos("Min Cont Area",
"Controls")
        contMinLength = cv2.getTrackbarPos("Min Cont Length",
"Controls")
        contMaxLength = cv2.getTrackbarPos("Max Cont Length",
"Controls")
        MPclosenessThreshold = cv2.getTrackbarPos("MP Closeness
Thresh.", "Controls")  # Maximum number of previous frames kept
for tracking
        MPtimeout = cv2.getTrackbarPos("MP Timeout", "Controls")
# Midpoint Tracking timeout

        krnl = np.ones((krnlX,krnlY), np.uint8)
        success, img = cap.read()

        h, w, _ = img.shape
        roi_h = round(0.81*h)
        roi = img[0:roi_h,0:w]

        imgDilation = cv2.dilate(roi, krnl, iterations=itns)
        imgErd = cv2.erode(imgDilation, krnl, iterations=itns)
        imgGrey = cv2.cvtColor(imgErd, cv2.COLOR_BGR2GRAY)
        imgCanny = cv2.Canny(imgGrey, cannyL, cannyH)
        imgConts, outConts =
findContours(imgCanny,roi,contHips,contMinLength,contMaxLength)
        trackingOverlay, windowList, mv = trackWindows(outConts,
windowList, MPtimeout, MPclosenessThreshold, roi.shape[0],
roi.shape[1])
        roi = cv2.cvtColor(roi, cv2.COLOR_RGB2RGBA)
        roi = cv2.addWeighted(roi,1,trackingOverlay,1,1)
        roi = cv2.cvtColor(roi, cv2.COLOR_RGBA2RGB)
        #moveTargetToCentre()
        outImg((roi,imgErd,imgConts,imgCanny),("Window
Detection","Dilated & Eroded","Contour Detection","Edge
Detection"))
        cv2.waitKey(1)
        LSF = 10
        RSF = 10

        if keyboard.is_pressed('esc'):
            break
        if keyboard.is_pressed('z'):
            autoMode = not autoMode
            manualControl = not manualControl

        moveVector = [0, 0, 0, 0]
        if manualControl or not autoMode :
            autoMode = False
            if keyboard.is_pressed('w'):
                moveVector[2] = 1 *LSF
            elif keyboard.is_pressed('s'):
                moveVector[2] = -1 *LSF
            else:
                moveVector[2] = 0

            if keyboard.is_pressed('a'):
                moveVector[0] = -1 *LSF
            elif keyboard.is_pressed('d'):
                moveVector[0] = 1 *LSF
            else:
                moveVector[0] = 0

            if keyboard.is_pressed('e'):
                moveVector[3] = 1 *RSF
            elif keyboard.is_pressed('q'):
                moveVector[3] = -1 *RSF
            else:
                moveVector[3] = 0

            if keyboard.is_pressed('shift'):
                moveVector[1] = 1 *LSF
            elif keyboard.is_pressed('ctrl'):
                moveVector[1] = -1 *LSF
            else:
                moveVector[1] = 0
        else:
            autoMode = True
            moveVector[0] = round(mv[0]/10)
            moveVector[1] = round(-mv[1]/10)
            moveVector[2] = round(mv[2]/10)

        outData = str(moveVector[0]) + "," + str(moveVector[1]) +
"," + str(moveVector[2]) + "," + str(moveVector[3])
        s.send(outData.encode())

    saveSettings()

    s.close()
```

A.4 Equation Derivations

A.4.1 Cleaning Motor Torque Derivation

$$(1) - \ F_R = \mu R$$

$$(2) - \ T_B = F_R r_B$$

$$(3) - \ T_M = T_B \times G_r$$

Substituting Equations 1,2 and 3, and rearranging for motor torque

$$\Rightarrow T_M = \mu R r_B G_r$$

Assuming:

$\mu = 0.6, R = 9.81N, r_B = 85mm,\ G_r = 2,$

$\underline{T_M}$ =1Nm

A.4.2 Inclination Angle Derivation

$$\sum F_{x_d} = 0$$

$$\Longrightarrow Rcos(\alpha) + F_R sin(\alpha) - (g + \ddot{x})msin(\alpha) = 0$$

$$\Rightarrow Rcos(\alpha) + (F_R - \ddot{x}m - gm)sin(\alpha) = 0$$

$$\Rightarrow Rcos(\alpha) = -(F_R - \ddot{x}m - gm)sin(\alpha)$$

$$\Rightarrow \frac{R}{(F_R - \ddot{x}m - gm)} = -tan(\alpha)$$

$$\Rightarrow \alpha = \tan^{-1}\left(\frac{R}{(\ddot{x}m + gm - R\mu_{\text{equivalent}})}\right)$$

Assuming:

$R = 9.81N,\ \ddot{x} = 1ms^{-2},\ m = 3kg,\ g = 9.81Nkg^{-1},\ \mu_{\text{equivalent}} = 0.6,$

$$\underline{\boldsymbol{\alpha = 20.28^o}}$$