GDP Draft Report

GDP Group

March 31, 2022

Contents

1 Introduction

The main objective of the project was to design a smart and autonomous golf caddy. A golf caddy is a person that carries a player's golf bag and clubs and gives the player advice and moral support. Our project consists in designing an electronic trolley that can perform to its best the caddie's job and help players improve their golf game without having to pay a second person for their service. This means being able to follow the player and suggest the optimal golf club to choose in each shot. Although this idea has long been invented, we are aiming to produce a prototype of a product that could stand out in the market. Our competitive advantage over our rivals is our club release mechanism, which releases the suggested club when needed and the fact that the caddy will not follow the user if he or she enters the green or a hazard.

The improvement of the existing idea was proposed by James Siabi to our primary supervisor Dr Mohamed Moshrefi-Torbati. James is our industrial supervisor who after many years of playing golf and experiencing electronic golf trolleys, found the gap in the market. Due to our lack of resources and time, the final product consists in a prototype that will not be ready to commercialise but to prove the possibility of a future commercialization.

2 Design Brief

2.1 Project Aims

At the beginning of the academic year the group decided on a set of aims to achieve by the end of the project:

- Autonomously follow golfer
- Automatically avoid hazard areas
- Suggest optimal club choice
- Incorporate a mechanism to release suggested club
- Incorporate a touchscreen display with a user interface

2.2 Project objectives

The objectives are more specific targets set to achieve the aims. These are:

- Enable real time GPS communication between pod and trolley
- Design an algorithm to avoid obstacles in an efficient way
- Learn golfer‘s tendencies through a machine learning algorithm
- Develop a light mechanism to release desired golf club
- Design with Python a graphical user interface

3 Mechanical Design

The final design of the AGC is shown in Fig. (1).

Figure 1: Final AGC design

3.1 Golf Bag

The golf bag is one of the crucial parts of the project as it holds the release mechanism of the golf club and the essential electronics. An old golf bag had been modified as manufacturing a new golf bag is unrealistic. The old golf bag had been dismantled to accommodate the release mechanism, compartment such as electrical base and mechanical base had been implemented to ensure the mechanism isn't disturbed by the electrical system.

Golf Bag Infrastructure The whole bag had 3 different parts, the top base of the golf bag, the mechanical compartment and the electrical compartment as shown in Fig. (2). The top base of the golf bag is used as a support and segregation compartment for the golf clubs. The mechanical compartment consists of the motor compartment and the release mechanism. The motor compartment is placed in between of two support to ensure the motor keep in place during operation. The release mechanism is on top of the motor compartment. It had a support base to ensure the tube and the release mechanism stay aligned. All the electronics and the cabling are placed in the electrical compartment to ensure the cabling do not disturb the mechanical system and also for the ease on waterproofing all the electronics.

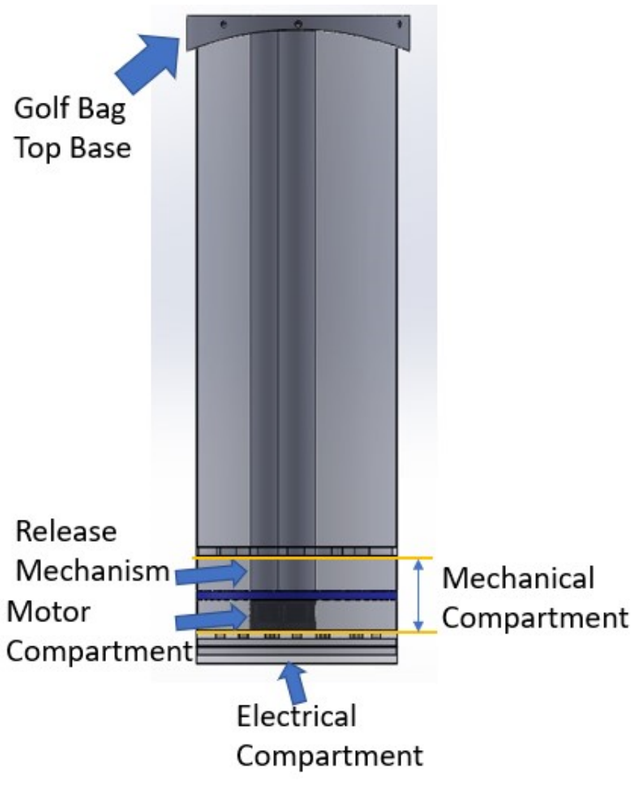

Figure 2: Infrastructure of Golf Bag

Top Part of Golf Bag The initial top part had been removed due to unfriendly design; the initial top part had uneven spacing for all the holes. By removing the initial top part, we could fully utilize the available space of the bag. A new design shown in Fig. (3) had been implemented to fit 14 equal size release mechanism for the golf club. Each slot had equal spacing and had sufficient space between the slots. The new design had been initially laser cut with Plywood. The tube fitted the design and are well supported by it. Due to the limitation of the 3D printer size (250 x 210 x 210mm), the new design had a dimension of (255 x 230 x 41 mm). It had been sliced into 4 equal pieces and been sent for 3D printing. The 3D printed parts had been assembled together by hot glue and glued on the frame of the old golf bag top frame as shown in Fig. (4)

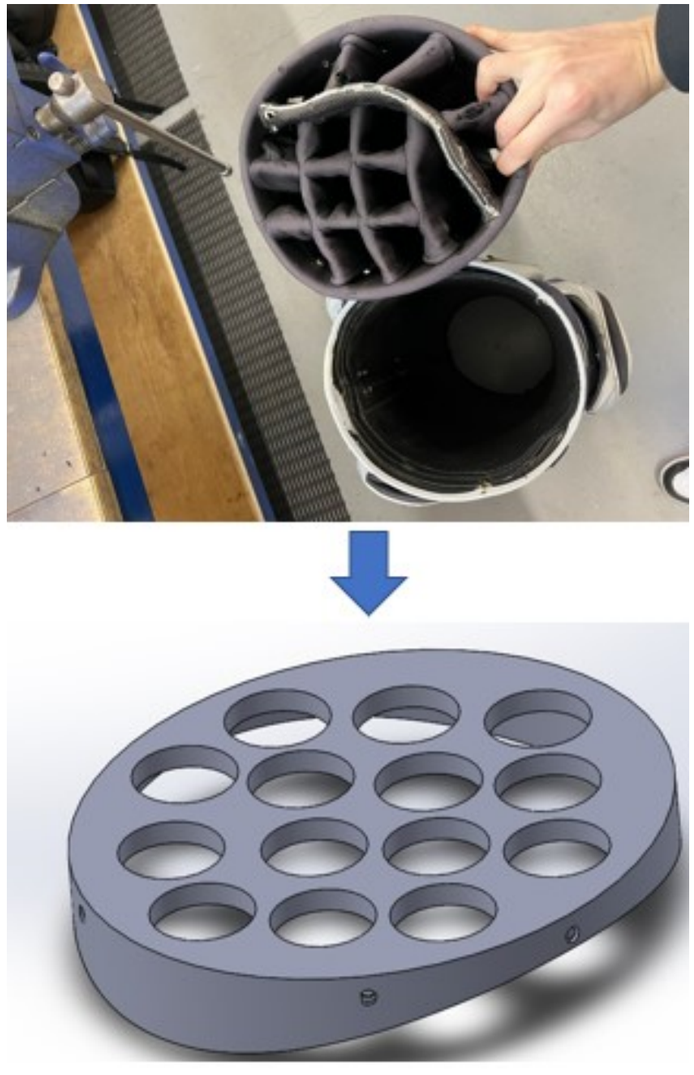

Figure 3: Top part of golf bag

Figure 4: 3D printed top part

The design in Figure 3Dparts.png had been used as the support for the release mechanism during testing. Due to the final changes in the of the release mechanism. A new top base had been designed to fit the latest release mechanism.

Extension of Golf Bag The release mechanism used up 150mm of vertical height space of the whole golf bag. To prevent the golf club be-

ing extended too high in the mid air. The golf bag need to be extended to compensate the height used up by the mechanism. Therefore, the bag had been extended using rolled metal sheet. The extended part is used to fit in the mechanical base and the electrical base. The extended part is necessary as the release mechanism require an extra 90mm space. Therefore, the bag had been extended by 150mm. 90mm are used for the release mechanism, 50mm are used to fit the motor and 10mm are for the electrical base.

Touchscreen Casing The LCD screen currently do not have any casing and all the electrical parts are exposed to the environment. In order to protect the LCD screen from environment damage such as rainwater damage and physical debris damage, a LCD screen casing had been made which are well compatible with the screen holder. The overall casing design are shown in Fig. (5). The casing contains a top casing and a bottom casing having the screen in between them.

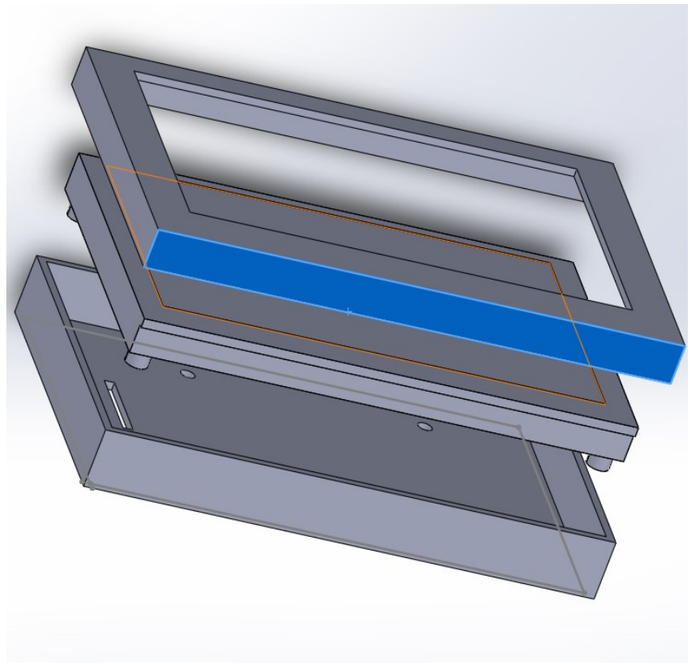

Figure 5: Bottom view of the bottom casing with all the holes

Bottom Casing The bottom casing fit the LCD screen perfectly, there is a slot behind to allow the electrical ribbon to connect to the caddy. The bottom casing fits perfectly with the screen holder. All of the holes on Fig. (6) will screw directly to the LCD screen except for hole A and B. The screw will go through the holder and the casing to the LCD screen to ensure the screen and the casing hold firmly on the screen holder.

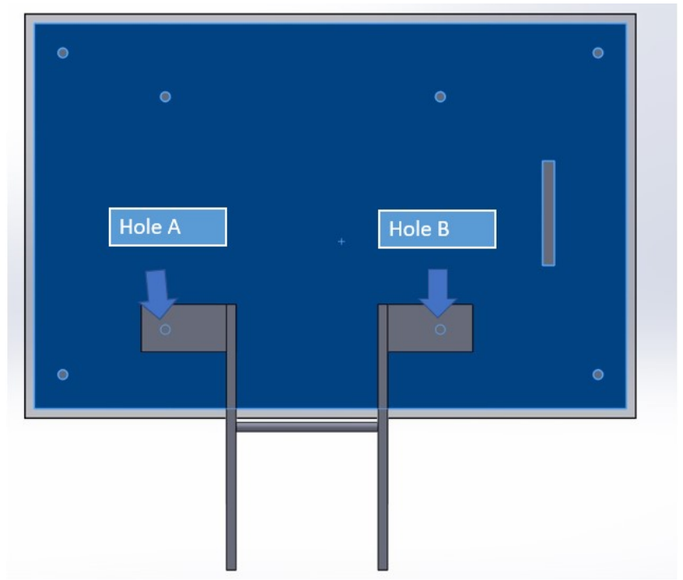

Figure 6: Bottom view of the bottom casing with all the holes

Top Screen The top screen covers the bottom casing edges and also the LCD screen as shown in Fig. (7). The orange line represents the LCD screen, the yellow line represents the bottom casing while the grey line represents the top screen casing. There is an empty slot on the top screen to install a screen protector on it. The screen protector will be glued on the casing and a layer of resin will be added to prevent water intake. The screen protector does not affect the touch screen functionality yet it provide rain resistance.

Figure 7: LCD screen (orange line), bottom casing (yellow line), top casing (grey line)

Screen Friction Hinge The screen had been installed right below the caddy handle. The location of the screen is well placed as it utilizes the abundant space of the caddy, it does not affect the folding operation of the caddy. The screen can be folded and unfolded by using a friction hinge design as shown in Fig. (8). The folding action took place by pushing the hinge 90 degree. The hinge shown in Fig. (9) relies on the friction provided from the bolts and nut. The tighter the bolts and nut, the harder to rotate the hinge.

Figure 8: LCD screen fit well without affecting the caddy folding operation

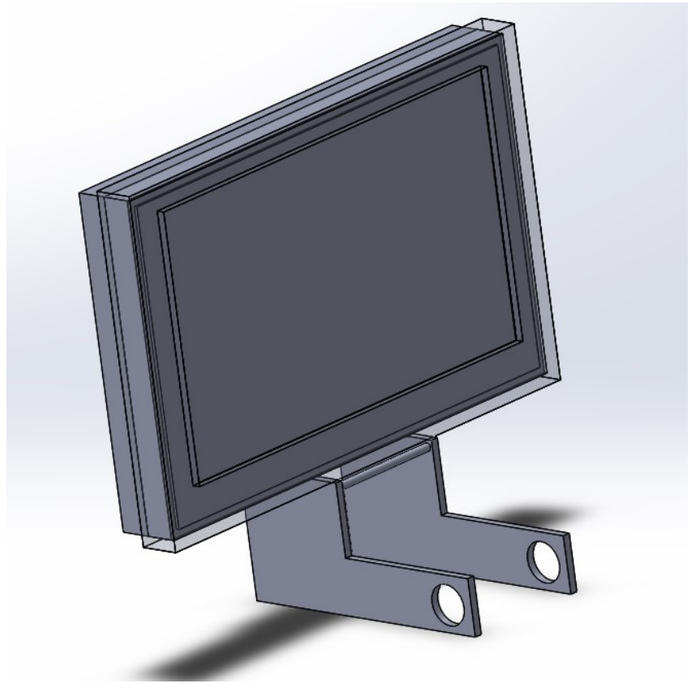

Figure 9: Friction hinge for the LCD screen

3.2 Club Raising Mechanism

One of the key objectives for this project was to design and implement a release mechanism that will give a golf club to the user based on its position on the golf course and the distance to the green. This system will be used in conjunction with the machine learning algorithm which will be the main brain towards selecting which club to give to the user.

3.2.1 Design Requirements

- Need to lift at least 10 cm above other golf club in order to make it clear which club is being given
- Needs to withstand the weight of the golf club which produces an axial loading
- Needs to be lightweight in order to not affect the entire golf bag since it will be carried by a user
- Needs to be a robust design since it will be used a lot of times

3.2.2 Design Process

Initially the team started discussing on the available workspace that can be used to hold the mechanism. Two locations were identified, one was on the golf AGC where the golf bag rests and the other being inside the golf bag itself. The most feasible workspace was determined to be inside the golf bag since the system on the golf AGC will have a higher probability of failing to work as it would need to be aligned perfectly such that each mechanism is directly below each golf club.

We then started investigating on possible configurations of systems that can offer a linear motion that will push up the golf club. Some of the most prominent ideas were to use linear actuators, slider-crank mechanism, an umbrella like compression spring mechanism, and a stepper motor with a threaded rod.

First Iteration After going through the various ideas presented, there were some clear advantages and disadvantages with each of them. Starting with the linear actuators, they are easy to use, robust in terms of mechanical functionality, can be automated and be used in a controlled environment. However, they are prone to be heavy in mass with one actuator weighing about 1.2kg. If they were to be used for 14 clubs, that would be an extra added weight of 16.8 kg which will make the new golf bag, with the added actuators, 2.12 times heavier than without them. This would be an issue for the golfer as they will have to carry a bag over 30 kg, and it will also have an influence on the motors required to drive the AGC since they will need provide more torque to propel forward.

For the slider-crank mechanism, although it appears to be the simplest design to adopt and use, the diameter of the crank would have had to be 10 cm if it is required to elevate a golf club by 10 cm. Due to the restricted workspace available in the golf bag to fit the mechanism, it would not be physically possible to fit 14 cranks of diameter 10 cm in the golf bag.

It now leads us to the first feasible idea which is an umbrella like mechanism using compression springs. Adopting a similar mechanism as in an umbrella, where a compression spring is under compression and release due to the force input from a user, the first design was built on CAD and is shown in Fig. (10). Fi. (11) shows a 3D printed pin that will act similar to one found in an umbrella.

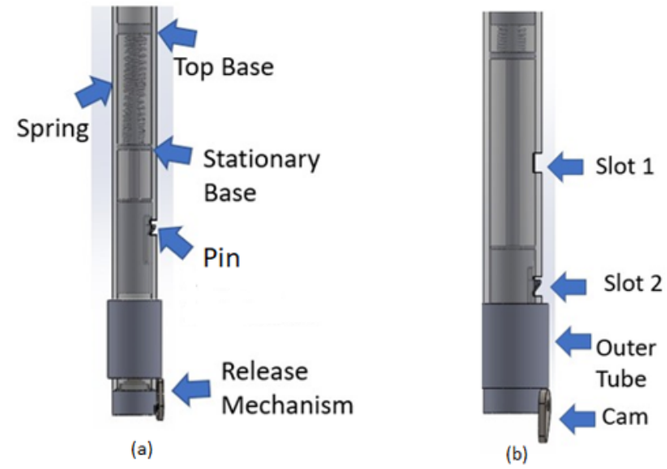

Figure 10: Mechanism in a) retracted state b) released state

Figure 11: Pin

In this design, a compression spring is attached to a moveable top base and stationary base with a rod in the middle which leads to a pin that sits in either slot 1, when the mechanism is in “released” state or slot 2, when the mechanism is in “loaded” state. A user input is required to compress the spring to make the mechanism enter the “loaded” state. In order to release the compressed spring, which in turn gives the elevating effect, a stepper motor is used with a cam attached to its axle. An outer tube rests on the cam and when the cam rotates, the tube transfers its axial force received from the stepper motor, tangentially to the pin, which then pushes the pin inside and releasing the compressed spring. The pin will return to its original position once it reaches slot 1 after it suffered an elastic deformation from the outer tube.

The mechanism was well suited for its purpose, however, the structural integrity of the pin soon failed. Since it had to be 3D printed, due to its intricate shape, the force acting on the pin acted as a cyclic loading which causes stress induced cracking at the point of bending. This in turn made

the pin to fracture prematurely during testing. It was deemed that this design was too fragile to be used for a prolonged amount of time as on a golf course.

This design was later improved due to the discovery of a mini electric lock that has an inbuilt motor inside a door latch design. The mini electric lock and the improved mechanism is shown in Fig. (12).

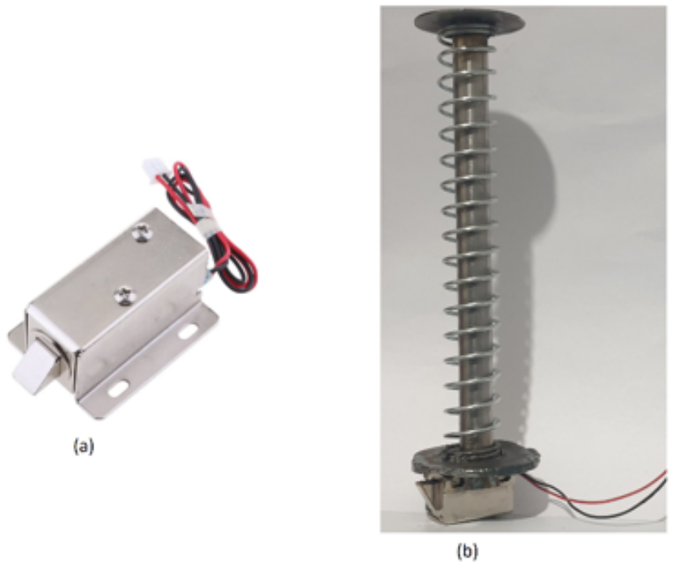

Figure 12: a) Mini Electric Lock b) Improved Release Mechanism with electric lock

The improvements on the original design made it more compact and rigid as a mechanism. However, the electric lock did not perform well when it was in the "loaded" state. The motor required a high torque and therefore a bigger size lock would be needed. The overall dimensions of each electric lock would then exceed the required amount in order to fit 14 in a golf bag for each club. Moreover, since it is not a controlled release, safety issues may arise should the mechanism release the golf club with excessive force which may hit the user. This design as well was deemed not usable as a release mechanism.

Second Design Iteration Another innovative idea was to use a mechanism that employs a nut and bolt analogy. If either a nut or bolt is fixed in free space and the other is rotated, the only direction for motion is either up or down, depending on the direction of rotation. Using this methodology, a stepper motor with a threaded rod was used and a plate holder was attached to a nut that will sit on the threaded rod of the stepper motor. This design was made to sit on the side of the tube and by using the inner walls of the tube to prevent the rotation of the nut, the only direction that the plate holder can move would be in the z-axis. Fig. (13) shows how the setup of this design look like.

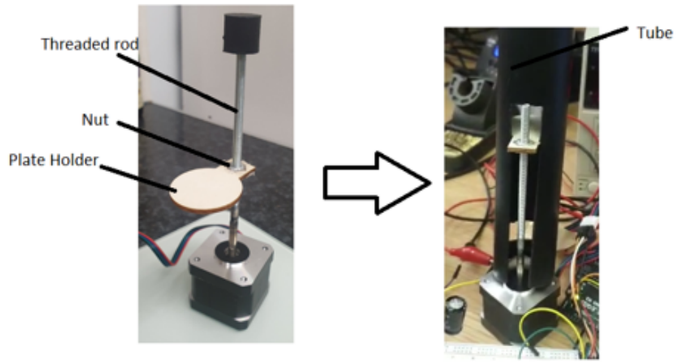

Figure 13: Stepper motor with plate holder in tube

Having weighted all the golf clubs given, the maximum mass recorded was around 700g. This mass was used in further calculation in order to obtain a required torque to lift the club. Using a pulley analogy, for a mass of 700g being pulled up at a radius, r. Eqs. (1 and 2) can be used to calculate the force, F, and torque, T, required to raise the club. Where m is the mass of the club.

$$F = W = mg_0 \tag{1}$$

$$T = Fr \tag{2}$$

The issues regarding this design were that the plate holder can get stuck within the tube and therefore failing the whole system. Another issue was that the pitch of the threaded rod was too close together which would mean that the motor would have to rotate for a long amount of time to elevate the golf club for 10 cm as required. This will decrease the overall operational lifespan of the motor.

Final Design Finally, the mechanism that the team has opted for is a stepper motor with a leadscrew alongside a 3D printed tube that will be elevated. Fig. (14) shows the final design chosen for the release mechanism.

Figure 14: a) Stepper Motor with Leadscrew and guiding rails b) 3D Printed tube that sits on nut of leadscrew

As the stepper motor shaft rotates, the tube that sits on the motor will try to rotate as well. However, instead of using the inner walls of the tube to hold the rotation, it now uses guiding rails prevent the rotation of the leadscrew's nut that is attached to the lower plate of the tube. Since there is a rotation of the shaft and there is a restriction placed on the rotating movements, the only direction of motion for the tube is either up or down.

The design chosen is more compact as it relies mostly on the operation of the stepper motor and can be contained mostly in a small space. It also allows for a more controlled release of the golf club and will have a high longevity since stepper motors can last for a long time. The overall weight of this design only amounts to 400g per mechanism and if it is to be used for 14 clubs in total, that would be 5.6 kg added to a golf bag which in turn is about 37% increase from an original weight of 15 kg. Fig. (15) shows how the mechanism is set up inside in the golf bag.

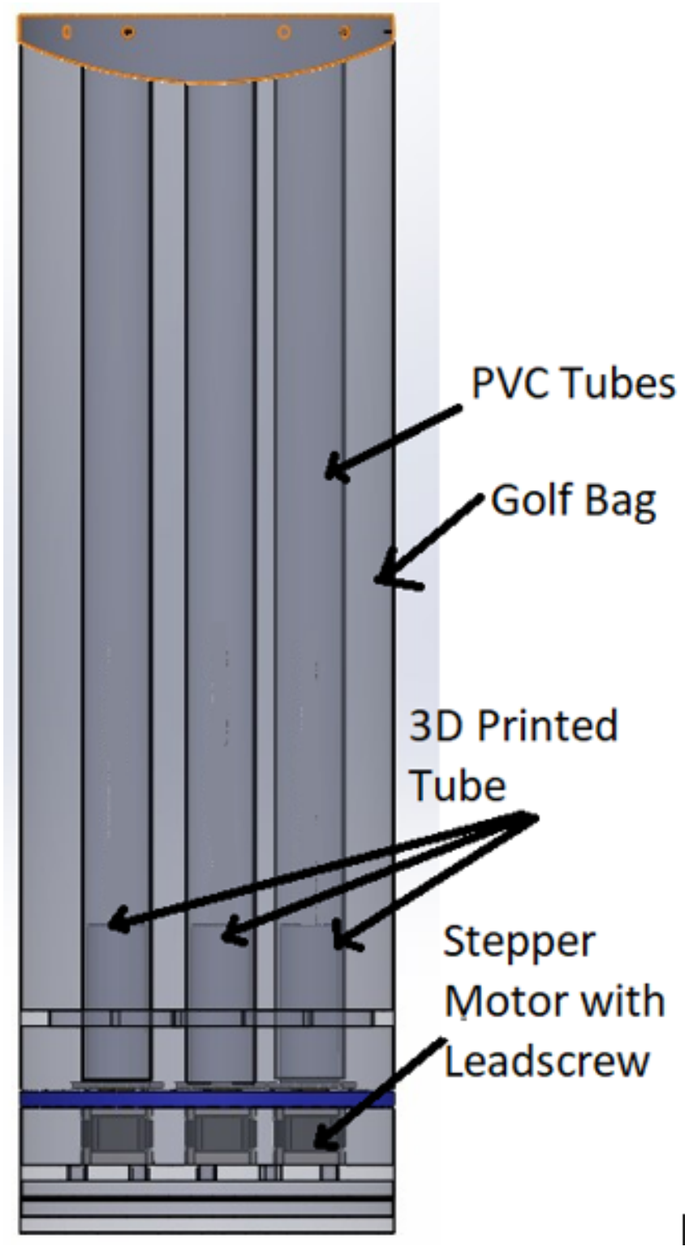

Figure 15: Release Mechanism in golf bag

Table 1 lists the specification of the stepper motor used. This torque produced is well enough within the required torque to raise the golf club even with a factor of 3.

Parameter	**Value**
Leadscrew Diameter	8 mm
Leadscrew Length	150 mm (±1 mm)
Motor Dimensions	42 mm × 42 mm × 40 mm (LxWxH)
Step Angle	1.8°
Nut Material	Brass
Holding Torque	400 mN m
Thrust (Full Step)	12.5 kg
Rated Current/Phase	1.5 A
Rated Voltage	3.3 V
Rated Resistance/Phase	2.2 Ω ± 10%
Mass	400 g

Table 1: Stepper motor specifications

3.3 AGC Electrical Base

3.3.1 Design Requirements

In order to control all the newly placed systems around the AGC, a base, that will house all the electrical components and the battery, is required. The design requirements for it are listed below.

- Needs to hold all the components required for the AGC to operate the wheels, GPS module, Raspberry Pi, LCD Screen, and motors.
- Needs to be lightweight otherwise it increases the weight of the AGC which affect the motor size and power.
- Needs to be easily accessible since the battery will be inside the casing and needs to be remove when charging is required.
- Needs to be waterproof and have a rigid structure such that the components do not move during movements on the golf course.

The most feasible place to have a base that will incorporate all the requirements, is where the previous battery was located. After removing the previous motors and battery plate holder, the new allocated space was measured and built on CAD in order to design a new compartment. Fig. (16) shows the before and after replacing the previous motors with the new electrical base.

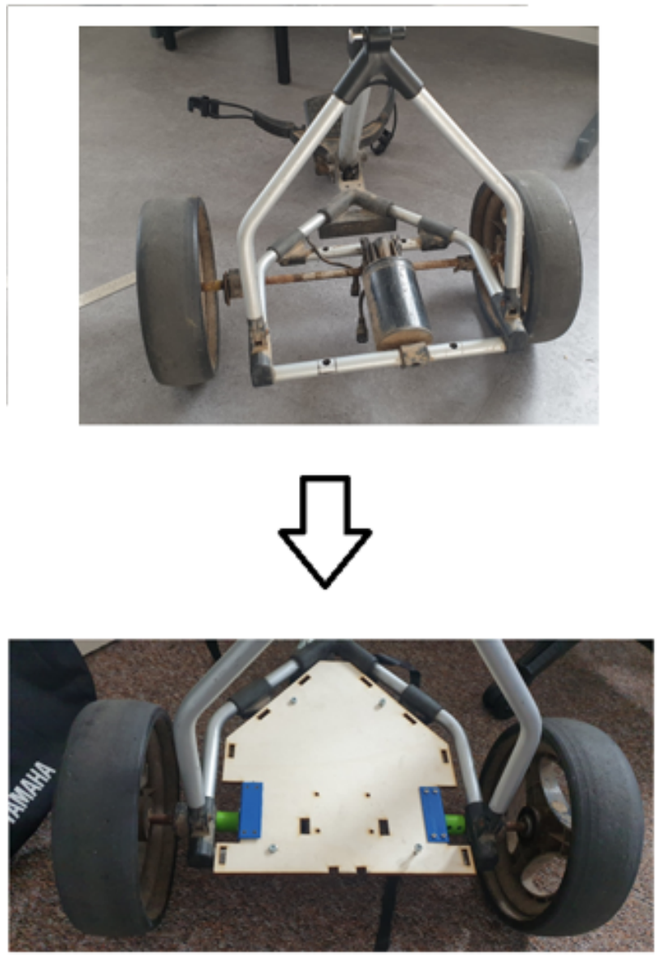

Figure 16: Replacing the previous AGC's motor with a base

Final Design Fig. (17) shows how the final base that will hold the electronics and the battery looks like.

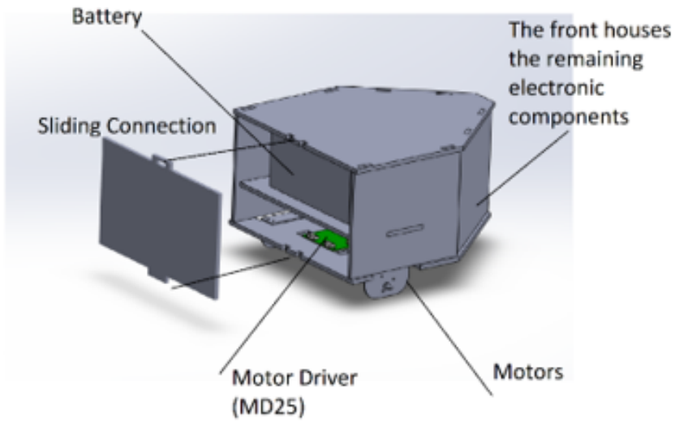

Figure 17: Full new AGC base

3.4 Wheels

3.4.1 Design Requirements

The wheels of the AGC are one of the most important components that will ensure a smooth and stable drive. They need to be able to give enough friction and be able to adapt to the surface on a golf course. Since the AGC will have to be autonomous and will need to be able to turn in certain events, a list of new requirements for the wheels are listed below:

- Maintain enough friction on wheels during acceleration such that it can be used on golf course
- Incorporate a system that allows the new motors chosen to operate one wheel individually
- Front wheels need to be able to turn 360 in order to accommodate when the rear wheels are made to make an arc using speed differentials between each wheel.

Design Process Initially the rear wheels on the original golf AGC were driven by a single shaft as shown in Fig. (18) and the front wheel was held in such a way that allows movement only forward and backwards.

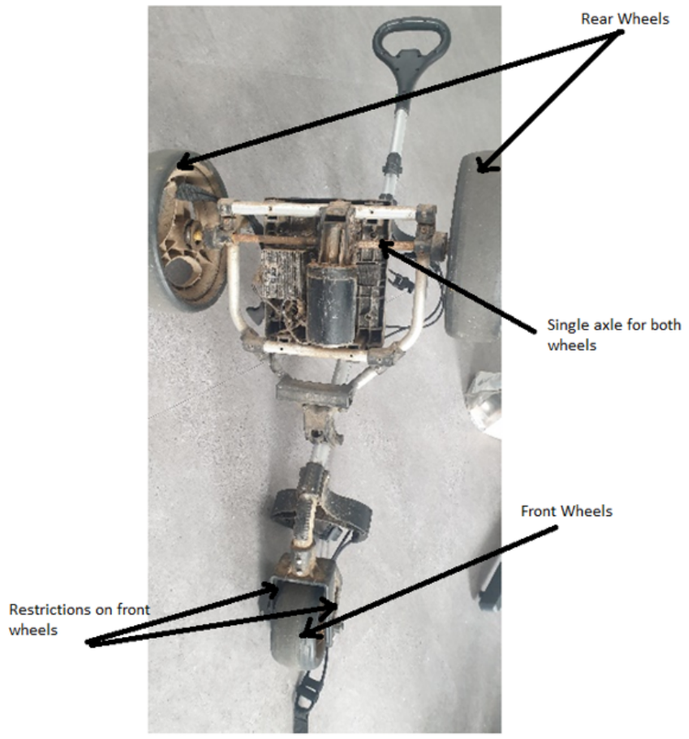

Figure 18: Previous wheels connection on original golf AGC

There were three options available to meet these design requirements. It was to either design and manufacture our own wheels, to buy another readily available wheel of the market, or to be more cost effective and re-use the current wheels. The current rear wheels were on a single axle and therefore to make it compatible to function with two separate motors, the shaft was cut down at two separate locations. These locations were chosen since the wheel themselves required a locking mechanism to the axle as shown in Fig. (19). In order to not having to redesign the locking mechanism, it was best to cut further down the axle such that it will be easier to join to the shaft of the motors.

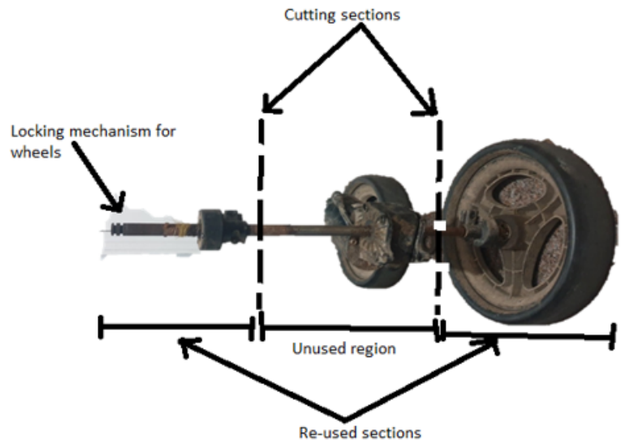

Figure 19: Single axle from previous golf AGC

The wheels with the shaft attached and the shaft of the new motors were then attached together with a shaft coupler. This shaft couple was designed, and 3D printed with M2 holes that will go through each shaft and allow for a rigid connection through the coupler. The shaft coupler 3D printed part is shown in Fig. (20) and the connection between the wheels and motor is shown in Fig. (21).

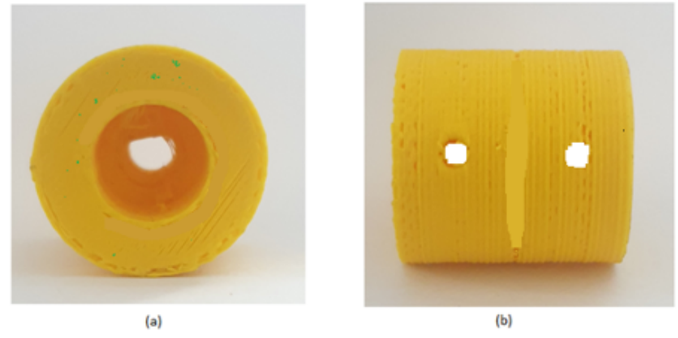

Figure 20: 3D Printed Coupler to connect wheels and motor

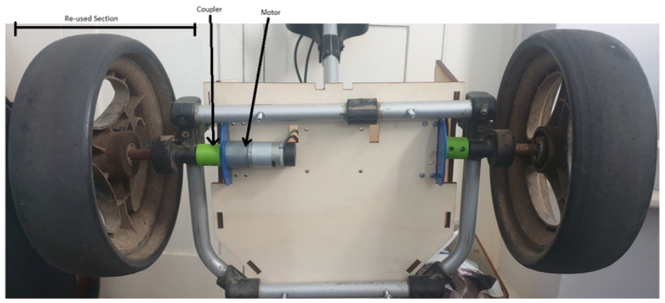

Figure 21: New arrangement of motors and wheels with coupler

The next adjustment that needed to be done is the front wheel. The available options we had were:

- Design and manufacture a bracket which will hold the front wheel alongside a ball caster that allows for a full rotation of the wheel
- To buy a swivel castor wheel of the market

The team decided to go with the second option since the cost and reliability of a ready-made swivel castor was better than a self-made one. The selected swivel castor is shown in Fig. (22).

Figure 22: Swivel Castor Wheel

Since the original location of the attachment of the wheels is not horizontal, the new swivel castor wheel could not rotate properly since it was at angle. Therefore, a sheet metal was attached at the uttermost low point of the AGC's main frame for the front wheel and then the wheel was attached to it. Fig. (23) shows how the previous wheel and the new swivel caster wheel are attached to the golf AGC.

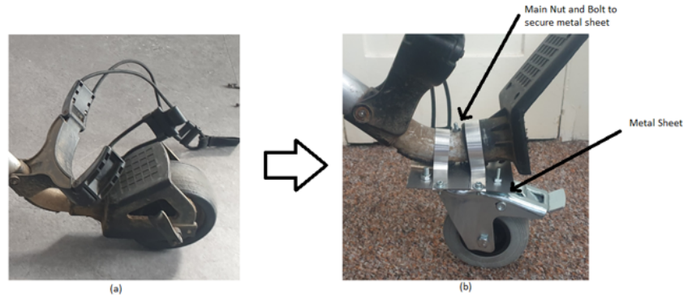

Figure 23: a) Initial front wheel connection b) Swivel castor wheel attached to metal sheet

Final Design Fig. (24) shows how the AGC looks like after assembling all the wheels.

Figure 24: Complete assembly of golf AGC with new wheels connection

4 Software

In this section all software required for the operating cycle of the AGC is discussed. The AGC main computer is a Raspberry Pi 4 running a 64-bit Debian port, the same is true for the Raspberry Pi used in the tracking pod. The GPS navigation system requires high precision decimal storage to operate properly. Data type `float32` can store 23 significant bits, compared to data type `float64` which can store 52 significant bits[1]. The calulcations shown below in Fig. (25) show the physical significance of which datatype is chosen.

Only 180° need to be accounted for because sign bit is separate, so to represent 180° the bits required is:

$$\log_2 180 \approx 7.49185 bits$$

This leaves the $n - 7.49185$ significant bits left for sub degree representation, the physical precision in meters we can then derive from each data type of a longitude value at the equator can be calculated as shown in the equations below.

For `float32` $n = 23$:

$$\frac{\frac{R_e \cdot \pi}{180}}{2^{(52-\log_2 180)}} \approx 2.386m$$

For `float64` $n = 52$:

$$\frac{\frac{R_e \cdot \pi}{180}}{2^{(52-\log_2 180)}} \approx 4.44nm$$

The radius of Earth. R_e, value used was $6371000m$.

Figure 25: Calculations showing physics precision of single vs double precision floats

The 180° is normalised to a power of two so we are able to use the non integer value of bits in the calculation for physical precision, were we to not perform this normalisation then $7.4918 \rightarrow 8$ full bits are required to represent the degrees and the final precision is slightly less for each datatype. To ensure this calculated precision is representative of our system all GPS coordinates are normalised to a power of two. We can see from the calculations that a significantly higher precision is achieved using double precision floating point data type to store GPS values. The precision yielded by the double precision floating point is unnecessary, however the precision of the single precision floating point is too low as our GPS module is capable of providing more accurate GPS measurements as discussed in section 7. Therefore double precision floating point datatypes will be used to store GPS data, which requires us to use a 64-bit compatible operating system which is why the 64-bit Debian port was chosen. The higher precision achieved comes at the cost of slightly higher compte time for calculations, however as most of the algorithms onboard operate only on small amounts of data, this effect is unnoticable when operating the AGC.

4.1 Control Software

This section discusses the software used to make the AGC follow the golfer. All of the control software apart from the elctronics specific code, such as motor controller communication, was written first using the OpenGL simulator that was built throughout the year. Building and using a simulator allowed us to write and test all the control software / algorithms that would be required to make the AGC follow the golfer as intended while avoiding hazard regions.

All of the control software tested in the simulator was written in C as this was easiest to use with OpenGL, however we decided to change to

PyQt5 for designing the onboard GUI. As the ACG software was now to be written in Python the control algorithms were re-written in Python, however upon testing it was found that their performance was significantly slower than the C implementations. This is most likely a large number of array passing and manipulations are required, which is significantly slower in Python where function arguments are passed as object reference instead of pointers in C. Therefore the original C control algorithms were compiled as a dynamic link library (DLL) and the Python "C Types" library was used to load the DLL and create Python wrappers for the control algorithms. This significantly improved the performance as all of the resource intensive calculations were now performed with a compiled C program.

The main control loop that determines the AGC's behaviour is shown below in Fig. (26).

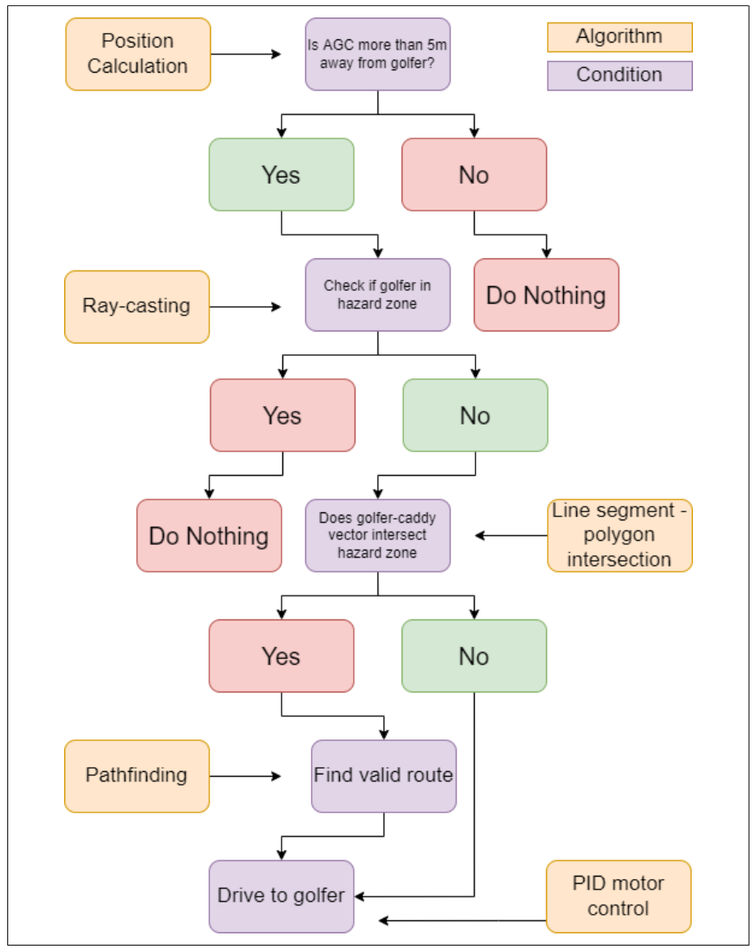

Figure 26: Main AGC control loop

4.1.1 Map Processing

As discussed in the introduction to section 4, all GPS coordinates are normalised to a power of two. We also "pad" the hazard zone coordinates, before normalisation, by a safety factor of two meters, preventing inacurate GPS readings or hazard zone coordinates resulting in the AGC entering a hazard zone. We "pad" the hazard zones by calculating the coordinates of centroid of the polygon, then extending the vector from the centroid to each perimeter point by two meters. The centroid of a polygon can be calculated from the coordinates of it's perimeter points as shown in Eqs. (3 and 4).

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{3}$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{4}$$

4.1.2 Ray Casting

The ray casting algorithm is used to determine whether a point falls within a polygon. The algorithm can only accept polygons with straight edges; our polygons are represented by a series of GPS points along their perimeter, and so our polygons are defined by a series of small straight edges. The concept of this algorithm is that given a point you wish to test, if you cast a ray from the point in one direction to infinity, the number of intersections the ray has with the polygon in question indicates whether the point falls in the polygon or not. If an even number of intersections are found, then the point lies outside the polygon, and if an odd number of intersections are found the point lies within the polygon. A visual of this concept can be seen below in Fig. (27).

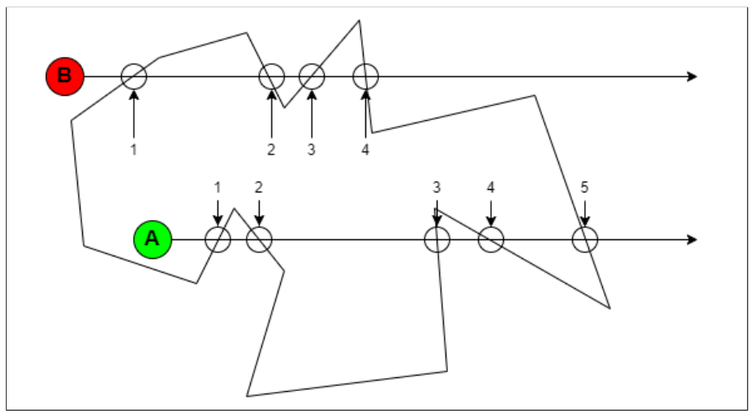

Figure 27: Raycasting Example

The figure shows two points which are to be tested to determine if they lie within the polygon. Casting the ray to the right, towards $x \to \text{inf}$ in our implementation, we can count the intersections each ray has with the polygon. Point A has five intersections and so lies outside the polygon, point B has four intersections and lies within the polygon.

There are additional considerations in our implementation to accout for the ray intersecting a vertex or for the ray being collinear with a polygon edge. If a vertex is intersected then two intersections are counted, one for each edge forming the vertex, and the result is not affected. If the ray is collinear with a polygon edge then only a single intersection is counted, but the ray will undoubtedly also intersect at least one of the vertices on the collinear edge which will be appropriately accounted for. The algorithm has been tested and does not fail in either of these cases.

4.1.3 Segment Polygon Intersection

Before the AGC starts moving towards any target destination, it first checks that it's intended translation vector joining it's location to the target location does not pass through any hazard zones. It does this by checking that it's intended translation vector doesn't intersect any edges of polygons. The diagram shown below in Fig. (28) depicts two finite length line segments intersecting.

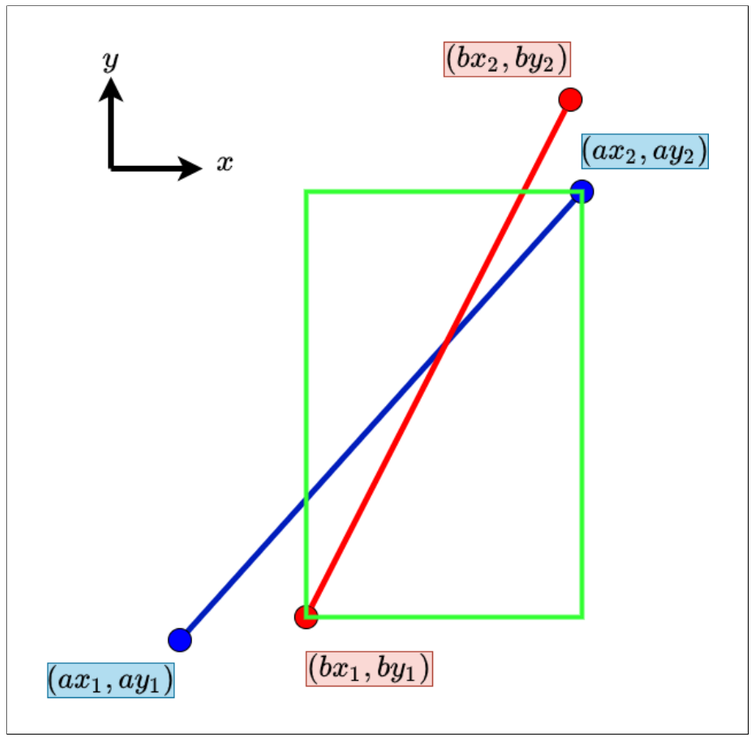

Figure 28: Line Segment Intersection Diagram

The edges of polygons are defined only by the positions of two vertices, as is the AGC intended translation vector. To calculate where the intersection point of these two line segments are a mathematical equation for a line must be constructed for each segment as shown in Eqs. (5 and 6) respectively, then these equations can be solved simultaneously to determine the location of the intersection as shown in Fig. (29).

$$ay_1 - y = \frac{ay_2 - ay_1}{ax_2 - ax_1} \times (x - ax_1) \tag{5}$$

$$by_1 - y = \frac{by_2 - by_1}{bx_2 - bx_1} \times (x - bx_1) \tag{6}$$

In our implementation, before this system is solved, we check if the lines have equal gradient, and if they are collinear or not. If they have equal graident, defined by:

$$\frac{ay_2 - ay_1}{ax_2 - ax_1} = \frac{by_2 - by_1}{bx_2 - bx_1}$$

Then we check if they are collinear, defined by:

$$(x - ax_1) = (x - bx_1)$$

If they are collinear then they intersect, if the have equal gradient but are not collinear then they do not intersect. If one or both of the lines are vertical, $(y_1 = y_2)$ then the system is not solved as below but trivially using the x position of the line.

For simplification:

$$m_a = \frac{ay_2 - ay_1}{ax_2 - ax_1} \qquad m_b = \frac{by_2 - by_1}{bx_2 - bx_1}$$

Now we solve the system defined by Eqs. (5 and 6) to determine the point of intersection resulting in the x coordinate given by Eq. (7) and y coordinate given by Eq. (8).

$$x = \frac{ax_1 m_a - ay_1 - bx_1 m_b + by_1}{m_a - m_b} \tag{7}$$

$$y = \frac{ax_1 m_a m_b - ay_1 m_b - bx_1 m_a m_b + by_1 m_a}{m_a - m_b} \tag{8}$$

Figure 29: Calculation of Line Segment Intersection Point

After the intersection point is calculated we ensure that the intersection point falls within the smallest rectangle bounded by one point from the first and second coordinates of either segment. We do this because the equation for a line is infinite, and any lines with non equal gradient will intersect, but we are only interested if the intersection occurs between points the AGC intends to travel. The bounding rectangle is demonstrated by the green rectangle in Fig. (28).

4.1.4 Pathfinding

The pathfinding algorithm is only used if the control loop shown in Fig. (26) determines that the AGC should move to the golfer but the a direct route is not possible due to the presence of a hazard zone between them. Initially the A* pathfinding algorithm was researched as it is a well known algorithm with which we could choose some heuristic function to minimise, in our case that heuristic function would likely be the shortest distance to reach the golfer. A* is generally used for pathfinding on a discrete graph, we do not have a discrete graph available and so would have to construct one for each golf course or a local graph surrounding the hazard we wish to avoid. This adds unnecessary complexity in terms of implementation and computation to our system which we wish to avoid. Additionally, A* has time complexity of $O(b^d)$ if d is the shortest path length and b is the branching factor, and the algorithm has high memory requirements as all generated nodes are kept in memory [2]. It was decided we would not use A* algorithm. The next consideration was to simply "pad" the perimeter of the hazard polygon and follow it all the way around until the AGC reaches the golfer. This concept is shown in the diagram below in Fig. (30) where the red line denotes the path that would be taken by the AGC to avoid a hazard zone. Clearly this is an inefficient path the AGC will unnecessarily follow the contour of the polygon. Additionally, this method increases the risk that the AGC enters the hazard zone if our GPS system is not performing well, or if our GPS mapping of hazard zones is erronous.

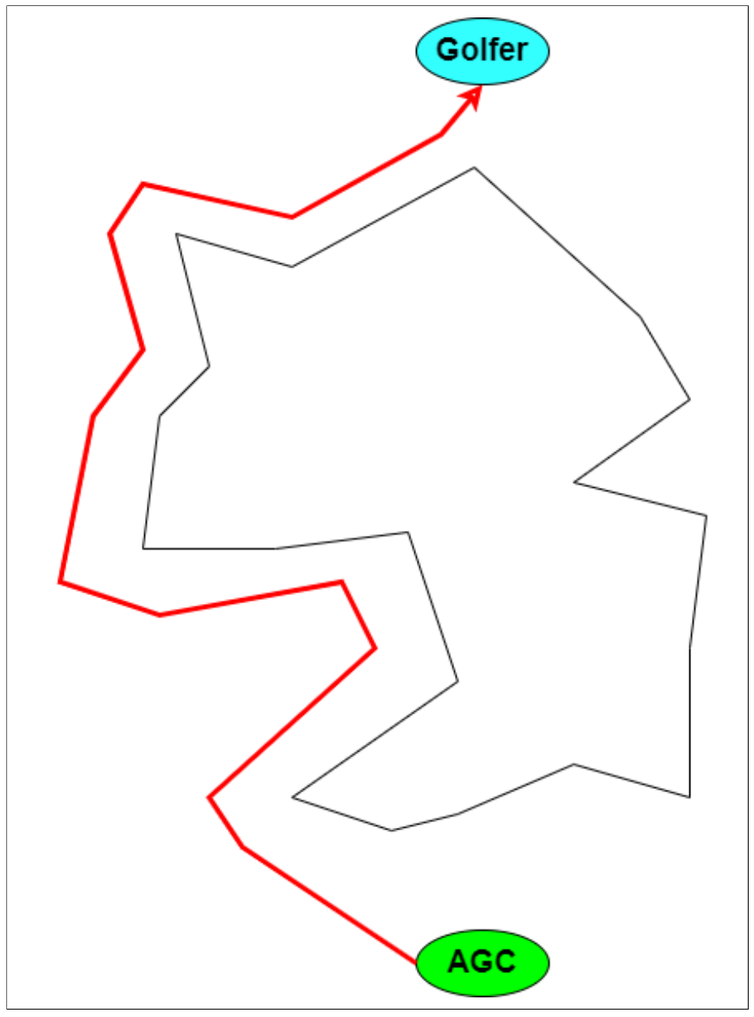

Figure 30: Path derived by "padding" polygon

A custom pathfinding algorithm was designed and implemented. The path it finds is not the optimal solution, but it computes quickly and the path derived is more efficient than using the "padding" method. Fig. (31) shows the main steps performed by the algorithm to determine where the AGC should go.

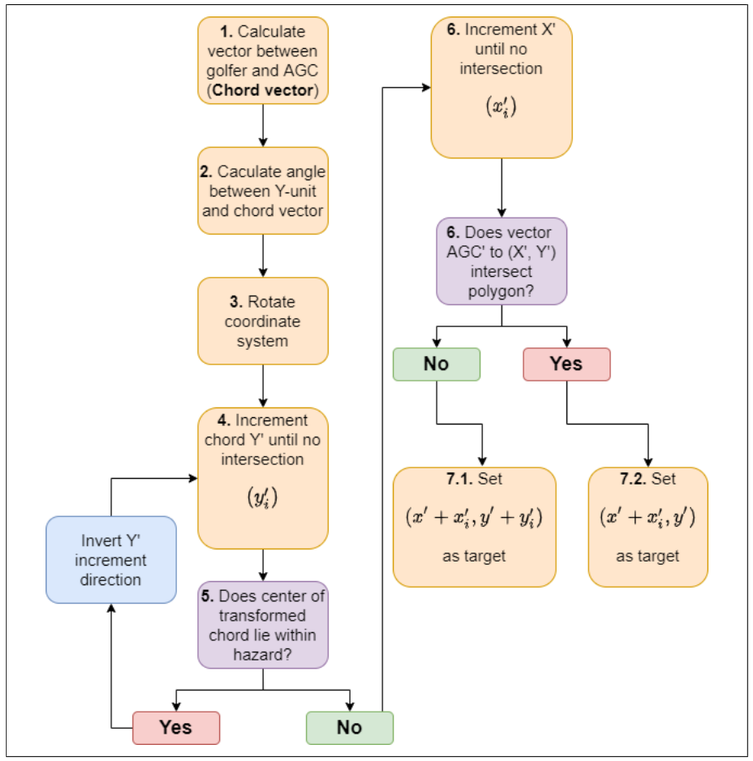

Figure 31: Pathfinding algorithm main steps

First the chord vector, $\overrightarrow{\mathbf{C}}$, is calculated, along with the angle, θ, between $\overrightarrow{\mathbf{C}}$ and the Y-unit vector, $\hat{\mathbf{y}}$, using Eq. (). The relevant polygon, golfer position, and AGC position are rotated about the z-axis with the center of rotation as the origin by θ using Eq. () where $\overrightarrow{p}$ is a three dimensional point and $\overrightarrow{p'}$ is the rotated point. The polygon is rotated by applying Eq. () to each vertex. The purpose of the rotation was to aid the design process of the algorithm and make the code more readable, removing the necesscity for vector manipulations throuhout the algorithm instead using only simple arithmetic operations. This results in $\overrightarrow{\mathbf{C}'}$ having constant y. The value of y' is increment in one meter steps until no intersection between the translated chord vector, $\overrightarrow{\mathbf{C}'_t}$, and the rotated polygon, $\mathbf{P}'$. If any point of $\overrightarrow{\mathbf{C}'_t}$ lies within the polygon the increment direction is reversed and incremented until there is no intersection and no points of $\overrightarrow{\mathbf{C}'_t}$ lie within the polygon. This increment process is repeated in the x' direction for the vector joining the rotated AGC position, (x'_{AGC}, y'_{AGC}), to $(x'_{AGC}, y'_{AGC} + y'_i)$. Once the incrementation process if finished such that no intersections are found, the vector joining (x'_{AGC}, y'_{AGC}) to $(x'_{AGC} + x'_i, y'_{AGC} + y'_i)$ is checked for intersections. If none are found then $(x'_{AGC} + x'_i, y'_{AGC} + y'_i)$ is rotated back to the original coordinate system and set as the position target for the AGC. If an intersection is found then $(x'_{AGC} + x'_i, y'_{AGC})$ is rotated back to the original coordinate system and set as the AGC position target. This process repeats until the golfer is reached. Incrementing in the y' direction is equivelant to incrementing perpendicular to $\overrightarrow{\mathbf{C}}$ in the original coordinate system, and incrementing it by x' is equivelant to incrementing in the direction of $\overrightarrow{\mathbf{C}}$.

4.1.5 Pathfinding Diagram

Fig. (32) shows a visual of how the pathfinding algorithm will determine intermediary positions for the AGC to move it towards the golfer. For the generation of this figure the pathfinding algorithm was only called again after the target position is reached, resulting in the very wide path that can be seen in the final path diagram. On the AGC the pathfinding algorithm is called every time new GPS data is available, approximately every second, resulting in a smoother and closer path being found.

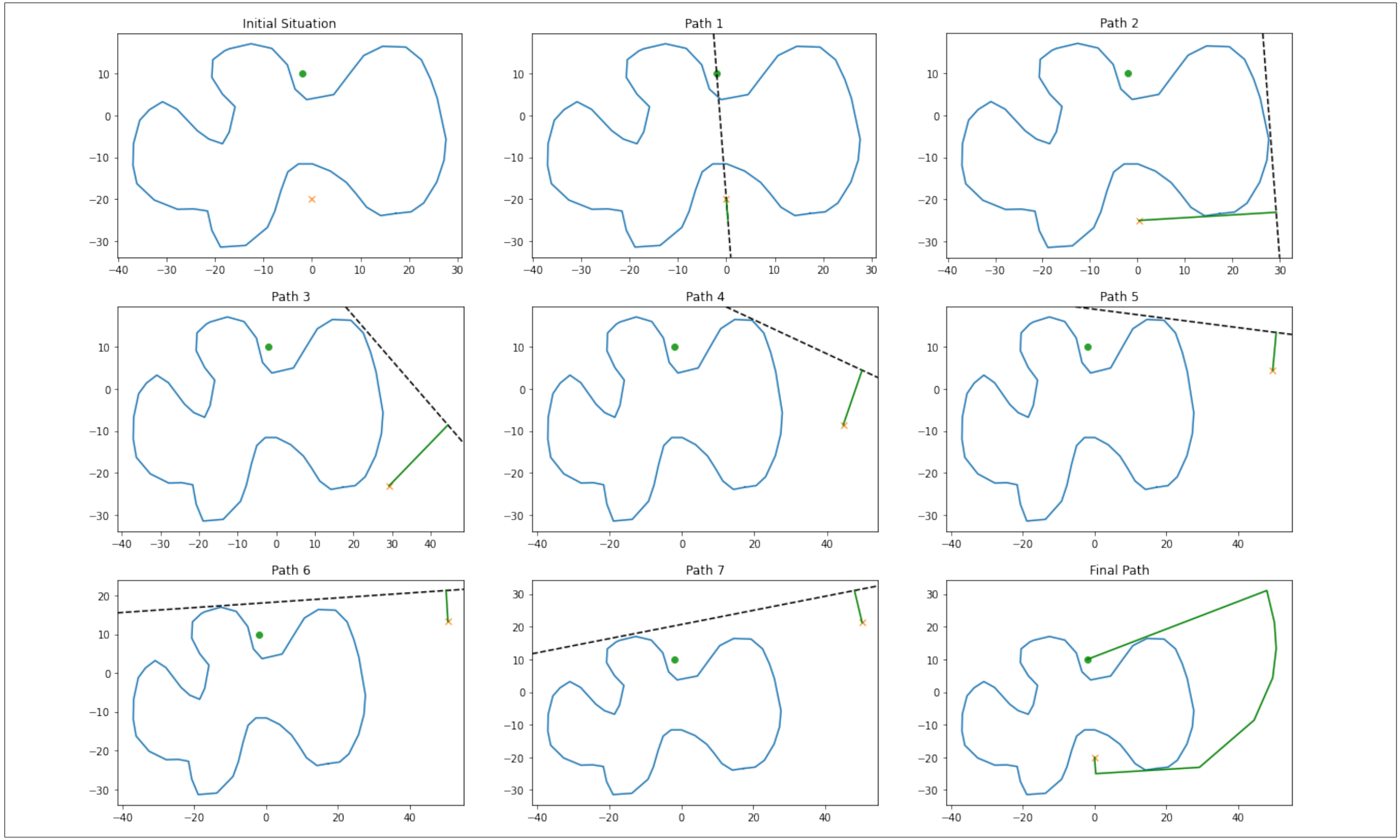

Figure 32: Visualisation of path determined by pathfinding algorithm

4.2 Communications

The EMG30 motors we are using are controlled by the MD25 motor controller board via and I^2C interface. The communication interface for this was written in C and is capable of handling multiple I^2C devices. Currently the only device we have on the communications network is the MD25. The GPS in the golfer's pod communicates to the main Raspberry Pi via Bluetooth, which is written in Python using the PyBluez library. The GPSs communicate with the Raspberry Pis via serial commuication implemented using the Pyserial library.

4.3 GCP Elevation API

The elevation is needed because the height difference in golf shots is relevant to decide the optimal club choice. Hence, the first idea was to use a Google Cloud Platform (GCP) elevation application programming interface (API). The platform allows a free trial period and after this the price per 1000 request is of 4-5 USD.

Having the location of the pod, which will be carried by the user, and the location of the hole in the golf course, the height difference can be known by transforming the location into latitude and longitude coordinates.

4.4 GUI

A GUI is developed to ease communication between the golfer and the caddy. We installed a screen which is compatible with the Raspberry Pi which acts as the brain of the AGC. The screen is intended to show displays of the map and enables the golfer to interact with the AGC for the golf selections, i.e. the golfer can choose to accept the AGC's recommended golf club or decline it. The GUI is developed using PyQt which is a Python binding of the cross-platform GUI toolkit Qt, used as a Python module.

PyQt is mainly used for creating GUI, developing desktop applications with Python. As an initial design implementation, we decided to use PyQt as the GUI of the AGC as the screen is able to showcase the GUI as an application through the Raspberry Pi. The main reason PyQt is used for this project is due to it being open source and provides many classes, methods and widgets. This allows us to make a rich app at zero cost.

4.4.1 Course Map

The course map was implemented on the GUI using the map system that was used on the simulator. We attempted to embed a map web app called folium which looks better than the map developed for the simulator, however, it doesn't allow us to retreive coordinates for a point clicked on the map in real latitude / longitude coordinates or in screen coordinates. This shortfall makes it impossible to give the golfer the ability to click a point on the map and see how far away it is from the AGC, using the simulator map allows us to add that feature.

5 Course Mapping

The coordinates of the golf map is required in order for the AGC to understand its movement boundaries on the golf course. The idea is for the AGC to know the coordinates of points where it should avoid such as bunkers, lakes and greens (hazards) while following the golfer throughout the golf course. The coordinates of the hazards are generated through a software called Mission Planner by drawing out polygons containing the coordinates of the hazards. Subsequently, these coordinates are extracted and converted into longitude and latitudes which are then used for the golf mapping.

Mission planner is created by Michael Oborne and has some features that are beneficial to this project:

- Point-and-click waypoint, using Google Maps/Bing/Open street maps
- Extraction of coordinates polygons as text file enables flexibility on how the data is used

It is used mostly as a ground control station for Plane, Copter and Rover [2]. In this project, its usefulness is its point-and-click waypoint feature which includes its ability to extract these coordinates. A waypoint is created by clicking on the map and a polygon is subsequently created. This polygon would have a unique coordinate (longitude and latitude) based on actual satellite mapping. The longitude and latitude can be extracted and saved as a text file which is the intention of this project enabling the coordinates to be implemented in other algorithms, mainly, the golf mapping and the following ability of the AGC.

Figure 33: Southampton Municipal Golf Course on Mission Planner

For testing, the Southampton Municipal Golf Course shown in Fig. (33) is chosen due to practical reasons:

- Closest and most accessible golf course for the team to go for testing which enables frequent testing at a low cost
- Appropriate size of golf course for initial testing which includes a perfect amount of hazards preventing overwhelming obstacles during testing and design

All the holes and hazards are identified and cross-checked with a golf course app, VPAR before drawing out the polygons. The waypoints are then created by drawing polygons around a parameter as in Fig. (34). For increased accuracy, the waypoints are drawn as closed as possible to each other as shown in Fig. (35).

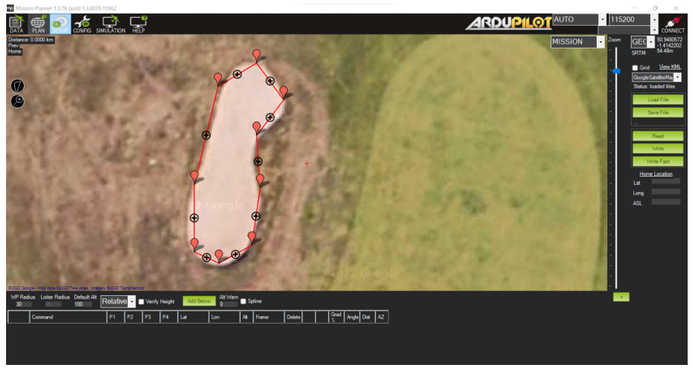

Figure 34: Drawing the polygons to define the waypoints for the hazards

Figure 35: Increase accuracy of coordinate extraction by having the polygons drawn more closely to each other

6 Machine Learning

The AGC will have the option to enable optimal club choice predictions if the user desires it. This option is placed on the AGC because current golf regulations outside the United States do not permit the use of software to help the golfer.

To determine the appropriate machine learning algorithm we first look at the type of data we have. The AGC will collect both input and output data. Input data will be mainly personal information from the user and type of club chosen while output data will be the result of those chosen

parameters. Hence, the model uses supervised learning. From supervised learning we can do a regression or classification model. The latter one is most suitable because we are predicting labels and not quantities.

7 Electronics

7.1 Motors

The motors we decided to use for our prototype version are the EMG30 motors because we already have them and this will save cost. However, they are underpowered as they will take too long to accelerate the AGC to full speed, and won't work at on anything more than a slight incline; for a final version the EMG49 motors should be used. We chose these motors by considering the top speed we wish the AGC to able to go, and how fast we want it to accelerate to that speed on a 10° incline. The calculations shown below in Fig. (36) show how we calculated the necessary toruqe and rpm we require from our motors.

The top speed we chose for the AGC was 5.0 kilometers per hour, equivalent to average walking speed. The wheel radius, W_r of the AGC wheels is 0.115 meters. We can calculate RPM using Eq. (9):

$$rpm = \frac{7.5 * \frac{1000}{3600}}{2\pi W_r} \cdot 60 \approx 115rpm \tag{9}$$

The AGC should be able to accelerate to this speed, V, within $t = 5$ seconds on a 10° incline. For predicted final mass, $M = 15kg$, the accelerating force, F, required can be calculated with Eq. (10):

$$F = \frac{V}{t} + Mg_0 \sin(10°) \approx 29N \tag{10}$$

Finally the torque required from each motor can bel calulated using Eq. (11):

$$T = W_r * F \approx 1.7Nm \tag{11}$$

Figure 36: Motor requirement calculations

According to their datasheet the EMG49 motors can provided a loaded rpm of 122 and a torque of 1.56, close to our requirements set out above. As the torque is slightly lower than calulated, the AGC will accelerate slightly to top speed on a 10° incline in slightly over five seconds.

7.2 Power Supply

The power for the trolley will be supplied by a 12v XXXXmAh LiPo battery. The battery will be splitting 12v to the stepper and DC motors and 5v to the microcontroller. Hence, the need of a 5v voltage regulator, provided by the university, between the raspberry pi and the battery. The battery was supplied by our project sponsor for free, so it is not included in the budget. The battery output connector was cut and the copper wires were introduced into a 2-pin screw terminal with an insulated and heat-shrinked plastic tube.

7.3 Touchscreen Display

The touchscreen used is a 5 inch DSI capacitive touch display. It was chosen because of its simple connection to the raspberry pi, which only consist of a ribbon, and its affordable offer. It also brings mounting holes and screws to fit it in a case as we have done in the AGC shown in fig X.

References

[1] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.

[2] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*. Pearson, 3 ed., 2009.