AUTONOMOUS GOLF PULL CART

By

Dillon McNulty

Kyle Gibbs

Oumar Soumare

Final Report for ECE 445, Senior Design, Spring 2020

TA: Jonathan Hoff

08 May 2020

Project No. 62

Abstract

Golfers have several options when it comes to how they will carry their golf equipment. Some of these options such as a golf cart or caddy service are expensive, while other options such as using a pull cart or manually carrying bags require great physical exertion. The goal of this project is to create an autonomous golf pull cart that would provide a balance between price and physical activity to golfers. In the original solution, this was completed by having the user wear a wearable device that would communicate with the cart. In our improved solution, we suggest using GPS and Bluetooth communication with the user's phone, which adds less hardware, provides the user more information on the cart, and allows the cart to avoid pre-defined restricted zones. Additionally, we plan to create a balancing system on the cart that will allow for smoother navigation on a golf course.

Contents

1. Second Project Motivation

1.1 Updated Problem Statement

When golfing, players usually decide between a golf cart, a pull cart, a caddy, or carrying their clubs around the golf course. All of these methods have downsides attributed to the cost or physical exertion required. A golf cart requires the player to drive instead of walk and it would be expensive to buy one. Then the added fee for renting a cart from the golf club every time would also be a large expense. The standard pull cart requires players to pull their bag around which may be difficult for some players to maintain around an 18 hole golf course. Hiring a caddy is the most expensive option listed since they're mainly available only on nice, expensive courses such as country clubs, and players are expected to tip the caddy. The last option of carrying one's own clubs is the lowest of options for many people. These people either physically can't or don't want to carry their golf bag that can weigh upwards of 30 pounds around an entire golf course.

Golf is also one of the most popular leisure sports in the world with an estimated 23.8 million players in just the United States as of 2017 [1]. Outside of the US, golf is most popular throughout Europe, Canada, South Africa, and Australia [2]. Due to these numbers, there is a large market opportunity for a product such as the Autonomous Golf Pull Cart that solves the problem for people who still want the walking as exercise but can't or don't want to carry their own clubs in a more cost effective manner than hiring a caddy.

1.2 Updated Solution

Our project looks to address these problems. By utilizing GPS, Bluetooth, ultrasonic sensors, preloaded course maps, and a gyroscope/accelerometer, our Autonomous Golf Pull Cart will follow the user around a golf course within a reasonable distance. Our project will allow the player to still walk the course and exercise but without the unnecessary added weight of a bag. The user will also be able to focus on their golf game and not have the added exhaustion of carrying their bag. The cart will connect to an app on the user's phone to create an easy user experience. A manual mode will be available that allows the user to control the cart with their phone to avoid any difficult obstacles. In some difficult and unforeseen cases, the user may still need to physically pull the cart out of or around an obstacle. The cart will have automated horizontal actuators to shift weights along the axles for the cart to stay upright during elevation changes.

There are other products on the market that attempt to tackle this problem. The Alphard eWheels Club Booster Electric Push Cart Conversion Kit [3] and the CaddyTrek Mobile Autonomous Robotic Golf Cart Caddy [4] are two products similar to our Autonomous Golf Pull Cart. Even though these products may be beneficial to some people, we noticed some key differences with our proposed project. The Alphard eWheels Club Booster Electric Push Cart Conversion Kit [3] is not a full cart but a conversion kit that can be added to a cart to create a remote-controlled push cart. This product does not introduce any autonomy and allows users to remotely move their bag with them as they walk. This product keeps the user from just focusing on their game which is why we wanted an autonomous cart.
The CaddyTrek Mobile Autonomous Robotic Golf Cart Caddy [4] does have autonomy built into a complete cart setup, not just a conversion kit. This product is very expensive and has to utilize a separate

piece of equipment the user must wear. Our on-person equipment will only be a cellphone. Both of these other options are also much more expensive than what we think we can design.

Our project's use of GPS will be a better solution than using sensors in the other 2 products mentioned. Preloaded maps of the course will have marked-off safe zones so the cart will not blindly follow the phone on to the green or into sandtraps. This project would be a great solution for all ages as well. The older golf demographic may struggle more with carrying and transporting their golf bag while the younger golf demographic may really enjoy the technology and automated features of this project.

1.3 Updated High-Level Requirements

1. Autonomous pull cart follows the user using A* algorithm around the course while only moving at minimum distance of 3 meters from the user
2. The mobile application can be used to control the cart manually by using a remote control or picking a specific point on the course map to send the cart
3. Pull cart remains balanced and drives smoothly when placed on inclinations less than or equal to 15°

1.4 Updated Visual Aid

As shown in Figure 1.1, the cart will be designed similar to a traditional golf pull cart with two rear wheels and one front wheel. It will include motors on the rear wheels to allow the cart to move autonomously. There will also be two more motors with counter weights, one moving left and right and the other moving forward and backward, to maintain the balance of the cart.

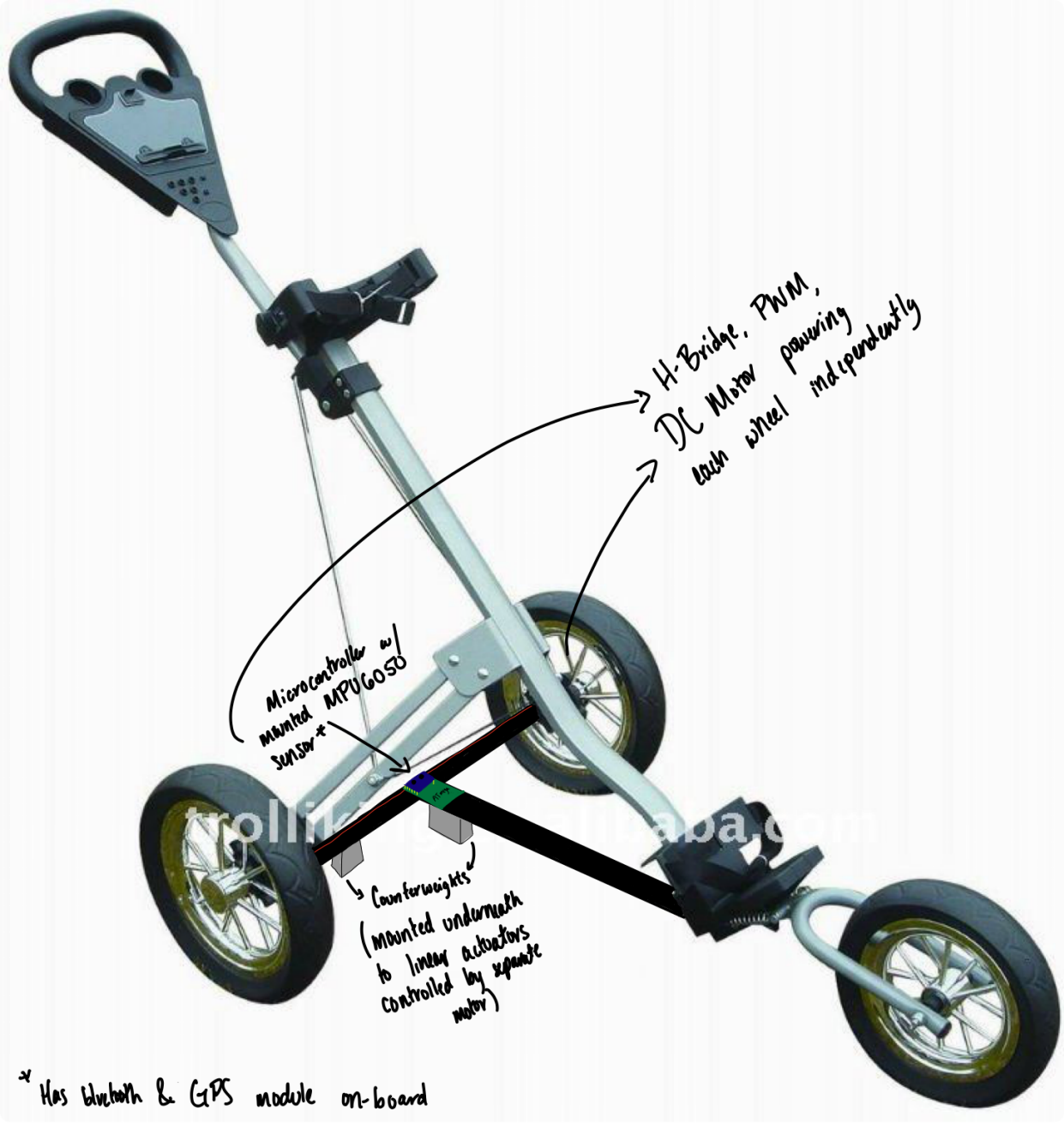


Figure 1.1 Physical Design for Problem Solution

Figure 1.2 Mobile application design concept. Allows the user to switch between modes, and view themselves and their cart's location on the map.

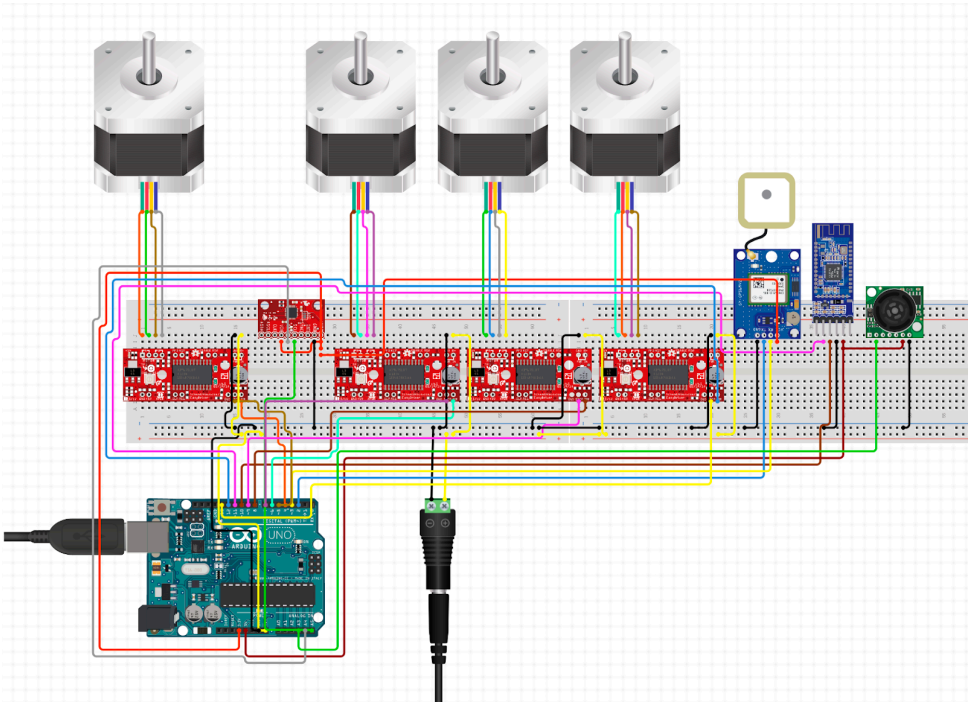


Figure 1.3 Breadboard prototyping schematic to test and assess different modules

1.4 Updated Block Diagram

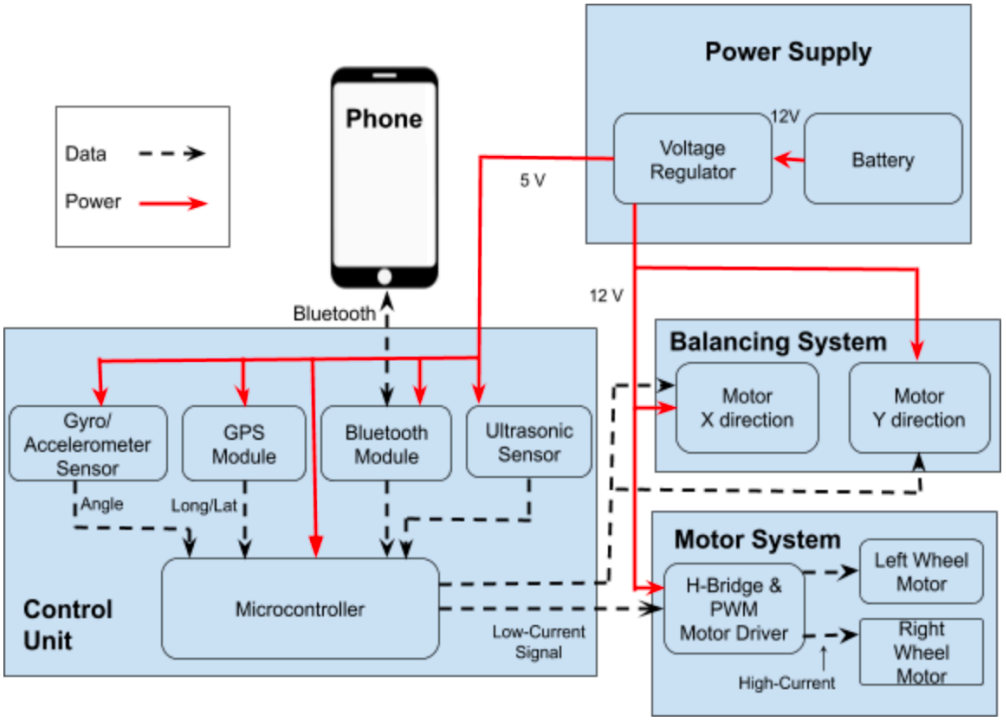


Figure 1.4 Block diagram illustrating signal and power connections for all subsystems of our project

2. Second Project Implementation

2.1 A* Algorithm

Given a graph, the user's location, and the cart's location, an A* search algorithm can be used to find the shortest path from the cart's location to the user's location while avoiding restricted areas. In this implementation, we were able to create the algorithm and simulate the navigation process on a subset of scenarios that may occur on a golf course. For any A* algorithm, a heuristic is needed to estimate the cost to move from point A to point B. The heuristic used in our implementation of the A* algorithm is based on the Euclidean distance formula presented in equation (2.1), where a is the user's x coordinate, b is the cart's x coordinate, c is the user's y coordinate, and d is the cart's y coordinate.

$$Heuristic = \sqrt{(a-b)^2 + (c-d)^2} \tag{2.1}$$

The A* Algorithm was conducted on four different scenarios. In Figure 2.1, we present the results of the scenario where there are no obstacles. Since our algorithm is based on Euclidean distance, when there are no restrictions, the cart should always traverse at a diagonal resulting in the shortest path. In Figure 2.2, we have now added a sand trap, which is commonly found on golf courses. Ideally, the quickest path to the user would be to follow the same path in Figure 2.1. However, the algorithm recognizes that there is a sand trap in the middle and plans an optimal path around the sand trap. In Figure 2.3 and 2.4, we present two more scenarios: a water hazard for Figure 2.3 and other terrain restrictions, such as trees, or bushes for Figure 2.4. In both of these scenarios, we see that the algorithm successfully plans around the obstacles to reach the goal. It is also worth noting that in each of the scenarios the cart follows a path along the edge of the obstacle, which could be dangerous in the event of GPS inaccuracy. Therefore, to avoid this issue, we plan to add a buffer around the obstacles when constructing the graph to account for possible inaccuracy and allow the cart to avoid the obstacles at a safe distance.

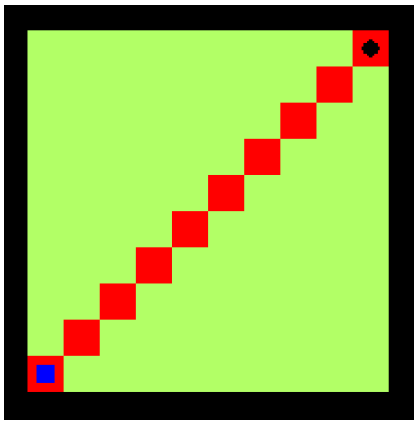


Figure 2.1 Basic test path for A*

At the end of each simulation, a list of coordinates representing the path is generated as shown in Figure 2.5. The results shown in Figure 2.5 represent the path taken to avoid the sand trap in Figure 2.2. For this project, this path will be converted to a list of compass directions that would be sent to the microcontroller for motor control. For instance, the path generated in Figure 2.5 will be translated to the

directions (NE, NE, N, N, N, N, NE, NE, NE, E, E, E, E), which will allow the cart to move one meter for each of the given directions.

The A* algorithm is a key component of our project as it is the basis for having the cart navigate to the user autonomously. By successfully completing the A* Algorithm, we are able to fulfill our first high level requirement.

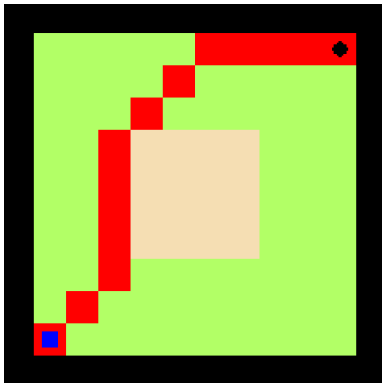


Figure 2.2 Basic path with a sand trap solution using A*

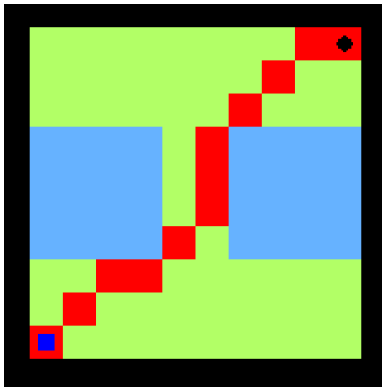


Figure 2.3. Demonstrating that the A* search will also avoid water

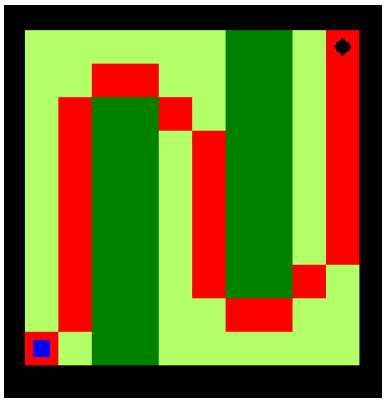


Figure 2.4 Demonstrating the avoidance of other hazardous objects

```
Path : deque([(10, 1), (9, 2), (8, 3), (7, 3), (6, 3), (5, 3), (4, 3), (3, 4),
 (2, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10)])
Path Length: 14
```

Figure 2.5 Raw output of solution path to be converted to cardinal coordinates

2.2 Balancing System Pseudocode

As will be further discussed in the unknowns and uncertainties method, I was unable to write a full Arduino project because of the complexity of the design involving several modules as well as the lack of ability to simulate anything without real data to get from the modules. Therefore, I wrote the algorithm in pseudocode to give a good idea of how it would work, as it was more important to identify the decision-making process of the microcontroller rather than the explicit details of how a gyroscope converts it's raw signal to an angle value.

```
0    //Global variables
1    pitch = 0
2    roll = 0
3
4    x_weight_pos = 0
5    y_weight_pos = 0 //range of each is 0 to 10
6    //Each of these arrays will contain the number of rotations the motor must turn 7    from
the center (either positive or negative)
8    //to be in that position for a given incline.
9     Horizontal_dict = [pos0, pos1, pos2, pos3, pos4, pos5, pos6, pos7, pos8, pos9, pos10]
10    Vertical_dict = [pos0, pos1, pos2, pos3, pos4, pos5, pos6, pos7, pos8, pos9, pos10]
12
13
14    main():
15    Initialize();
16    update_angles();
17
18    Initialize():
19    //code to set up each module as an input/output, etc
20
21    Angle_read():
22        return (roll_angle, pitch_angle)
23
24    update_angles():
25        incline_tuple = Angle_read();
26        x_incline = angle_read[0]
27        y_incline = angle_read[1]
28        if abs(roll - x_incline) > 3:
29            roll = x_incline
30            adjust_x(roll)
31        if abs(pitch - y_incline) > 3:
32            pitch = y_incline
33            adjust_y(pitch)
34        update_angles()
35
36    adjust_x(angle):
37        if angle > 15:
38            angle = 15
39        else if angle < -15:
40            angle = -15
41        angle += 15 // now in range 0-30
42        weight_position = angle / 3 // weight_position is in [0, 10]
43        move_x(weight_position)
44
45    adjust_y(angle):
46        if angle > 15:
47            angle = 15
48        else if angle < -15:
49            angle = -15
50        angle += 15 // now in range 0-30
```

```
51        weight_position = angle / 3
52        move_y(weight_position)
53
54    move_x(position):
55      diff = 0
56      curr_rotations = Horizontal_dict[x_weight_pos]
57      updated_rotations = Horizontal_dict[position]
58      if curr_rotations != updated_rotations:
59        diff = math.abs(curr_rotations - updated_rotations)
60      if diff < 0:
61        drive_motor_left(diff)
62      else if diff > 0:
63        drive_motor_right(diff)
64
65    move_y(position):
66      diff = 0
67      curr_rotations = vertical_dict[y_weight_pos]
68      updated_rotations = vertical_dict[position]
69      if curr_rotations != updated_rotations:
70        diff = math.abs(curr_rotations - updated_rotations)
71      if diff < 0:
72        drive_motor_backwards(diff)
73      else if diff > 0:
74        drive_motor_forwards(diff)
75
76    drive_motor_left(diff):
77      //code built using motor library to turn certain amount of rotations
78    drive_motor_right(diff):
79      //code built using motor library to turn certain amount of rotations
80    drive_motor_forwards(diff):
81      //code built using motor library to turn certain amount of rotations
82    drive_motor_backwards(diff):
83      //code built using motor library to turn certain amount of rotations
```

Figure 2.6. Pseudocode designed to be high-level version of balancing algorithm

Overall, the control logic for the balancing system is quite trivial. It starts by declaring all of the global variables that we will need to keep track of and reference during operation: pitch, roll, the position of each counterweight, and in lines 9 and 11 declares two arrays to hold the information on how many rotations it took the motor to arrive at that position. In order to maintain a 'perfect center,' we decided to use 11 positions on each extrusion with a given degree range, calculated by:

$$\frac{angle + 15}{3} \tag{2.2}$$

in order to map the values [-15, 15] to [0,10]. Once this position is calculated, the move_x/y functions are called to ensure that we actually need to move the weights to a different location. If we do, then the drive_motor functions (which would be a set of functions from the motor's library) are called to turn the motor a certain amount of times and drive the linear actuator to the correct position.

2.3 PCB KiCAD Schematics

Figures 2.7 and 2.8 below show the KiCad schematics for the PCB. We decided that two Arduino Unos would be necessary to support all the different sensors and motors. The overall schematic was divided into two boards that only share power inputs and outputs. The balancing system in Figure 2.8 only requires an Arduino[5], MPU-6050 gyroscope/accelerometer[6], and output mount holes that will be soldered to wires from the balancing motors. Then the entire navigation, regulation, and mobility systems are placed on another board that includes the NEO 6M GPS module[7], BlueSMiRF Silver Bluetooth module[8], Maxbotix LV-MaxSonar ultrasonic sensor[9], 12V DC motors[10], L298N motor driver[11], LM2575 voltage regulator[12], Arduino UNO[5], and battery pack[13] input. The 1x18 and 1x16 Male connectors on each schematic will be used to physically set the Arduino UNO on top of the PCB in order to connect it to the board.

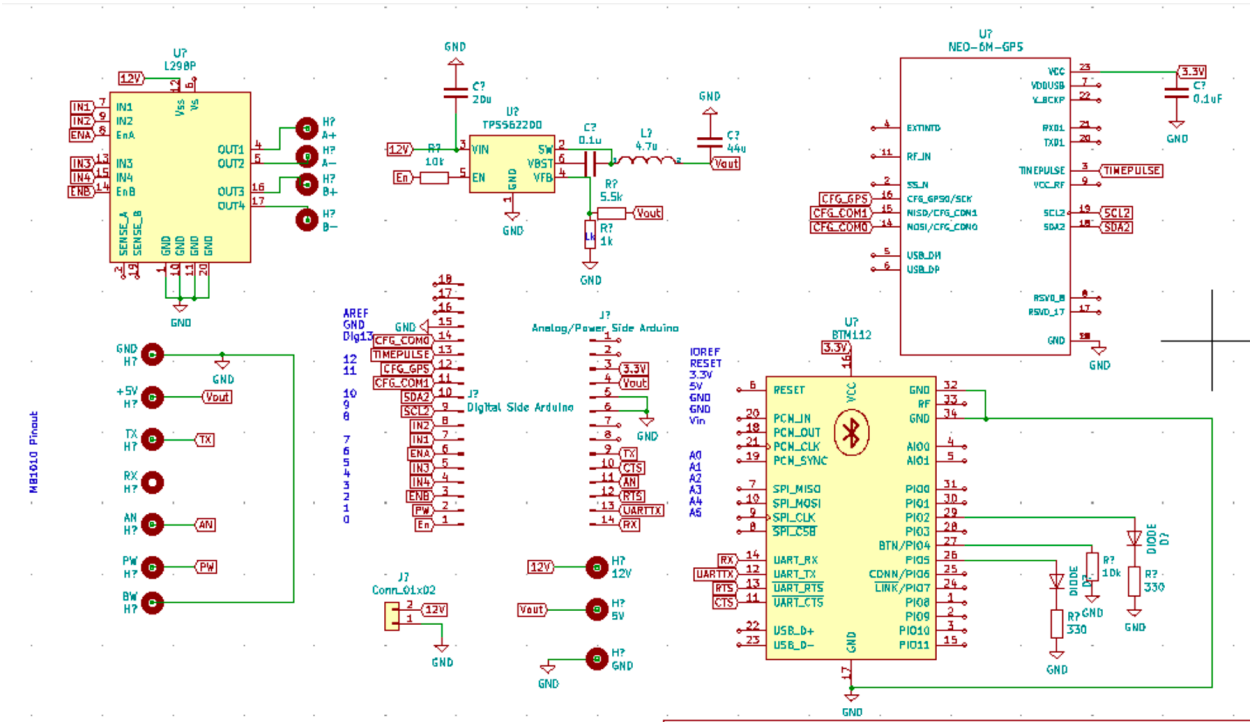


Figure 2.7 KiCAD schematic for regulation, navigation, and communication systems

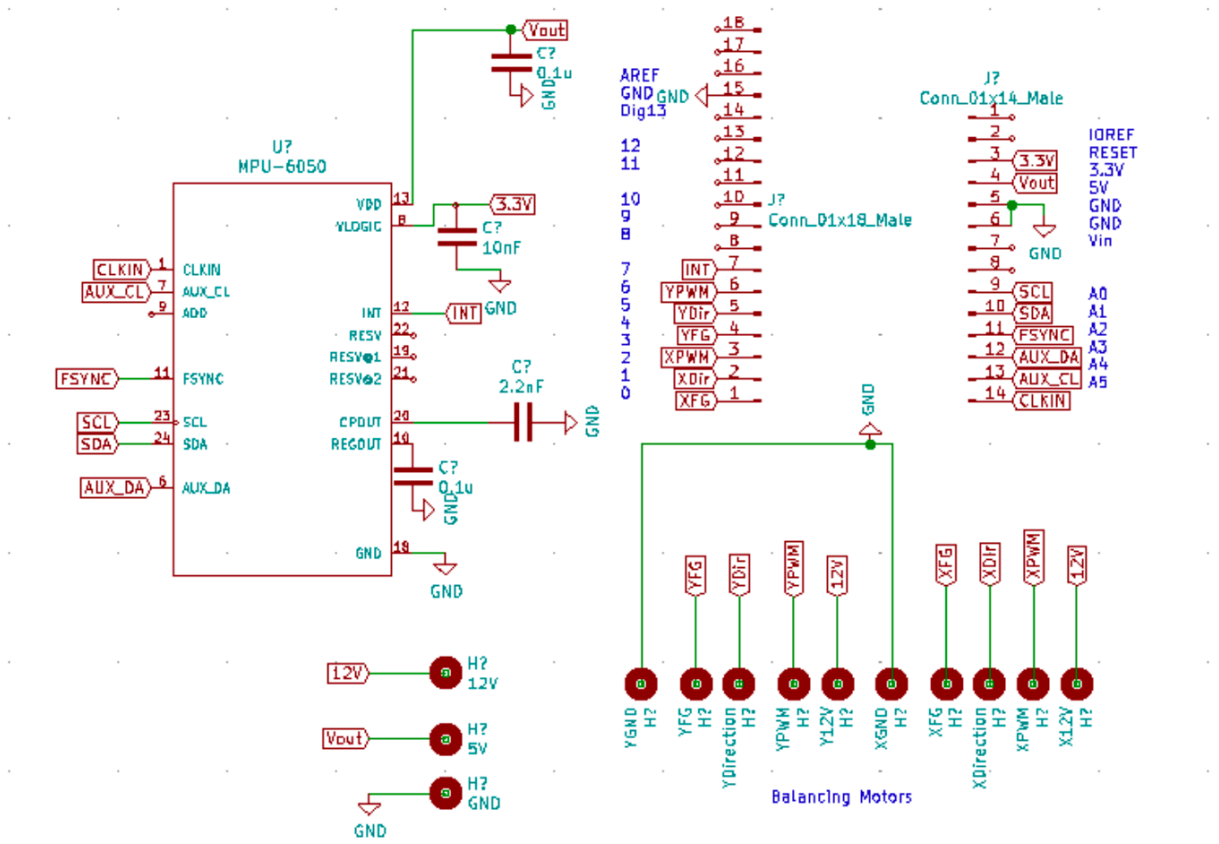


Figure 2.8 KiCAD Schematic for Balancing System

2.4 Center of Mass and Critical Angle Calculations and Estimates

Without a chosen pull cart, we estimated parameters such as weight and lengths after researching multiple pull carts. Then we assigned a 3-D coordinate system to use in our calculations.We placed the (0,0,0) origin directly in between the centers of the two side wheels. Then the z-axis would point directly up and perpendicular to the ground plane, the x-axis toward the back wheel, and the y-axis following the right-hand rule toward the right side wheel when looking from the +x direction. The total x dimension used for our estimate was 100cm, y dimension was 80cm, and then the z dimension was 117cm. These

$$CenterofMass = \frac{Mass_1 * Location_1 + Mass_2 * Location_2}{Mass_1 + Mass_2} \qquad (2.3)$$

were determined based on other common pull cart dimensions. Equation (2.3) shows the center of mass calculation for 2 objects in 1 dimension.

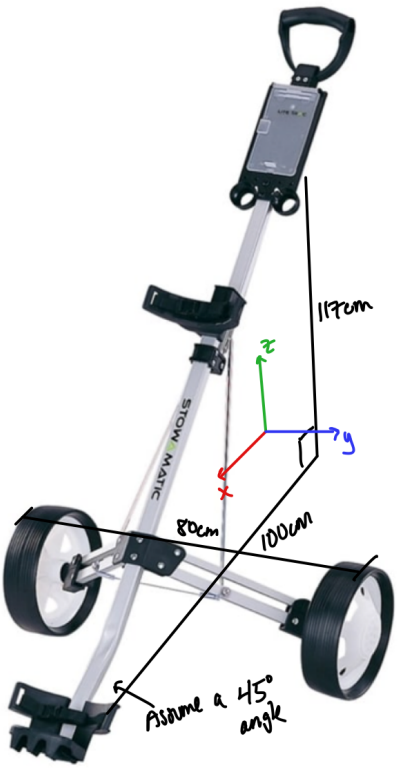
For 3-D systems, this calculation becomes more complicated where each of the 3 coordinates x, y, and z will have a separate center of mass that make up the whole center of mass.Using a standard length golf bag and mass of 13kg, and pull cart dimensions stated previously, the center of mass for the cart and bag system is estimated to be about (0, 0, 43.6cm). The y center of mass is assumed to be zero based on

symmetry of the bag and cart. Then the x center of mass is estimated to be about zero which is centered between the 2 side wheels. Lastly the z center of mass is of most importance to this problem based on the chosen coordinate system. The value of 43.6cm came from more estimations of both the bag and cart's centers of mass.

Then the next step in this process was to determine a critical tilt angle for the cart's tipping point. Assuming a tilt angle θ, a critical angle equation can be determined using trigonometric functions and triangles.

$$43.6cos(90-\theta) = 40cos(\theta) \tag{2.4}$$

This equation solves to give θ=42.5°. This value means that the cart should be able to stay standing when placed on any horizontal angle less than 42.5°. We expected a value much closer to our high-level requirement angle of 15°. We do not expect the value of 42.5° to hold up if we had the chance to work with an actual golf bag and pull cart. This critical angle of 42.5° was also found using multiple estimations which means physical experiments may be needed to develop the most accurate critical angle for the system.



COM: (-3cm, 0, 54.4cm)

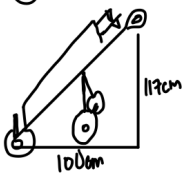
Cart Weight: 7kg

Avg. Heavy Bag Weight: 13kg

Total without counterweights: 20kg

- Bag will be on 45° angle w.r.t. ground
- Solve for COM_x (center-of-mass), COM_y, COM_z

① $COM_y = 0$ (symmetry)



$\frac{\sqrt{100^2+117^2}}{2} = 77cm$

$\sin(45°) = \frac{z}{77}$

$z = 54.4cm$

② $COM_z = \frac{13(54.4)+7(.2)(1.17)}{13+7}$

$= 43.6\,cm$

③ $COM_x = \frac{13(COM_{x\,bag}) + 7(COM_{x\,cart})}{13+7}$

· $COM_{x\,cart} \approx x = 10cm$

$COM_x = \frac{13(COM_{x\,bag}) + 70}{20} = \frac{13(-10)+70}{20}$

$= -3cm$

(just towards handle)

Figure 2.9 Diagram and calculations for determining the center of mass of the cart with a bag on it.

3. Second Project Conclusions

3.1 Implementation Summary

After our Design Review and TA meeting, we felt it was best to move forward with 4 major implementations: the A* algorithm, pseudocode for our balancing system, a KiCAD schematic for our PCB, and a physics breakdown to assess the proper counterweight size. Each one of these contributes to at least our high-level requirements and was an integral part of the overall design. Since certain aspects of the design had already been 'done' for us (for example, getting the raw angle of inclination from the gyroscope/accelerometer), it was more important to focus on the deeper algorithms and reasoning behind the design choices that we made. It is easy to get the pull cart to drive in a straight line, but how would it know to drive in a straight line? Questions like these are the ones we asked ourselves when determining what would be most fitting to implement in the past two weeks. While it's certainly only a small fraction of what we would have liked to completed, the figures in Section 2 show a clear proof of concept that could be expanded upon with the right tools and materials.

3.2 Unknowns and Uncertainties

Most of the hardware aspect of our design was hard to complete without both access to the lab and any of the modules we needed for data processing. Without any possibility for hands-on testing, building an entire Arduino project utilizing all of the libraries seemed fickle. As none of us have any personal experience with an MPU-6050, NEO-6M, or the others, all of the code that would have been written for testing would have been pulled straight from a library or online tutorial. As it felt redundant to copy-and-paste entire sections of tutorial code, we decided it was best to 'dumb down' the software engineering that was planned for our project. Instead of implementing an entire Apple Maps API with access to nationwide golf courses and data, it was best to focus on the A* algorithm itself and how we could simulate different situations for the cart to determine what it's solution path would be.

In addition to the issues with implementing the software that we designed, we were obviously not able to build any of the hardware components without access to the lab. Since soldering tools amidst others needed for the assembly of the balancing system were not at our disposal, we were only able to design the PCB in KiCAD as a hypothetical.

Had the laboratory been available, the PCB would have been ordered as soon as possible to minimize the chance of an issue happening with the shipment from overseas. While waiting on the PCB, the prototype described by *Figure 1.3* would have been built as soon as the modules came in. Once assembled, the prototype would be used for testing and learning how the different modules work and can interact with each other. Communication between them would have been built one at a time to ensure individual functionality. After we had felt comfortable with each module and how to control it, we would have moved forward with the actual logic specific to our pull cart. This would ultimately result in a combination of the pseudocode above and the libraries of each module which would reliably balance and control our cart autonomously.

3.3 Ethics and Safety

One of the biggest safety concerns that we see in creating this project is making sure that the cart is always operating in a safe manner. Golf clubs and the things that people put in their bag can be very expensive (phones, wallets, etc) and we would not want the autonomous cart to cause damage to the user's belongings in some way. More importantly we do not want to risk injuring the user or others. Any occurrence of the above would be a violation of IEEE Code of Ethics, #9: "to avoid injuring others, and their property" [5]. Therefore, we must ensure that the cart is operating at a safe speed and can accurately detect objects with the ultrasonic sensor to avoid damage to the user, property or the cart itself. We believe that maximizing the speed of the cart to 4 mph, the speed of a brisk walk, will ensure overall safe operation.

A worst case scenario would be the cart not detecting a lake, and driving into it with all of the user's possessions while they weren't paying attention. Due to these considerations, we will need to make sure that the GPS data being received and processed by the microcontroller is done very accurately, and draw 'safe zones' around these potential hazards to ensure that the cart never comes close to interacting with them.

In addition, we must make sure that we deliver on our promises. We would not want the user to purchase this autonomous cart and have to pull it around as if it were a regular cart due to it not working as intended. Therefore, we must ensure that the cart operates with the levels of accuracy mentioned in the requirements. Providing inaccurate data to the user would be a violation of IEEE Code of Ethics, #3: "to be honest and realistic in stating claims or estimates based on available data" [5].

3.4 Project Improvements

The first thing that we would start improving if we had more time and lab access would be more in depth and accurate center of mass and critical angle calculations. Many assumptions and estimates were needed in section 2.1.4 that led to a probably inaccurate value. We would need time to experiment with a physical pull cart and golf bag to determine the balancing capabilities of the standalone cart and bag system with no added balancing. Then we could ultimately determine first if the added balancing subsystem is even necessary for expected golf course inclines and second what weights would increase the critical angle to a region that fits with our project requirements.

One improvement that we'd really like to make is to add a 'Ball-Finding' subsystem. This would be the third mode of the cart, in addition to the Automatic and Manual modes. When this mode is selected by the user, they specify a circle of area on the map where they believe their ball is. The cart then traverses the area, and begins scanning the grass directly underneath the cart with a fisheye lens to see a wider portion of the ground without being too high into the air. Using computer vision, this camera waits to see a large cluster of white pixels on the ground, signifying a golf ball, since golf course's are specifically designed to avoid coloring things white (especially against the grass) so as to not confuse players. If the cart thinks it found a ball, it sends a notification to the user and stops moving so that they can go check for themselves. We believe this feature would be extremely useful as it can sometimes be difficult to find a golf ball in certain situations.

In addition to the computer-vision focused 'Ball-Finding' subsystem, we don't believe it would be particularly difficult to add in a 'fore-caddying' mode to the cart as well. When instructed by the user,

the cart will move off to the edge of the fairway of the next hole, and then turn to face the tee box of that hole. One more camera would be mounted to the front of the cart and, once the ultrasonic sensor ensures nothing is in front of the cart and it has a clear view of the tee box (could also be confirmed by the user before hitting their tee shot), it waits for the user. Once the user hits, it processes the first few frames of the shot to estimate the velocity and launch angle of the ball. Once estimated, it picks a 'guess' of where the ball landed, shows the user where that zone is, and begins to traverse to it. If its estimate is incorrect, the user can easily revert to automatic mode to allow the cart to return its focus to the user instead.

4. First Project Progress

For our first project, ‘Guitar Learning and Feedback Tool,’ we created the entire skeleton for the FSM / menu system that would be controlled with the two buttons and knob by the user in addition to the KiCAD schematic for the PCB. The FSM was implemented using python where the back/enter buttons were the ‘A’ / ‘D’ keys, respectively, and the scroll knob was simulated by the ‘W’ / ‘S’ keys. For each state, the user is given a list of options (or a set of information, if we’re in PLAYBACK or EVALUATE), and the pressing of the ‘A’ / ‘D’ key determines their choice. A set of screenshots from the simulated FSM can be found below, from the output of the python console.

```
PAUSED
>Select Song
Settings
Resume Playback

LENGTH
Current Measures:
1 - 6

LIST
>SongName1
SongName2
SongName3
SongName4

PLAYBACK
Song: N/A
40BPM
Measures: 1 - 6

SETTINGS
>Change Tempo
Adjust Measure Seq.
New Song
```

Figure 4.3. Screenshots of the Python console during the operation of five different states, each labeled at the top of its section.

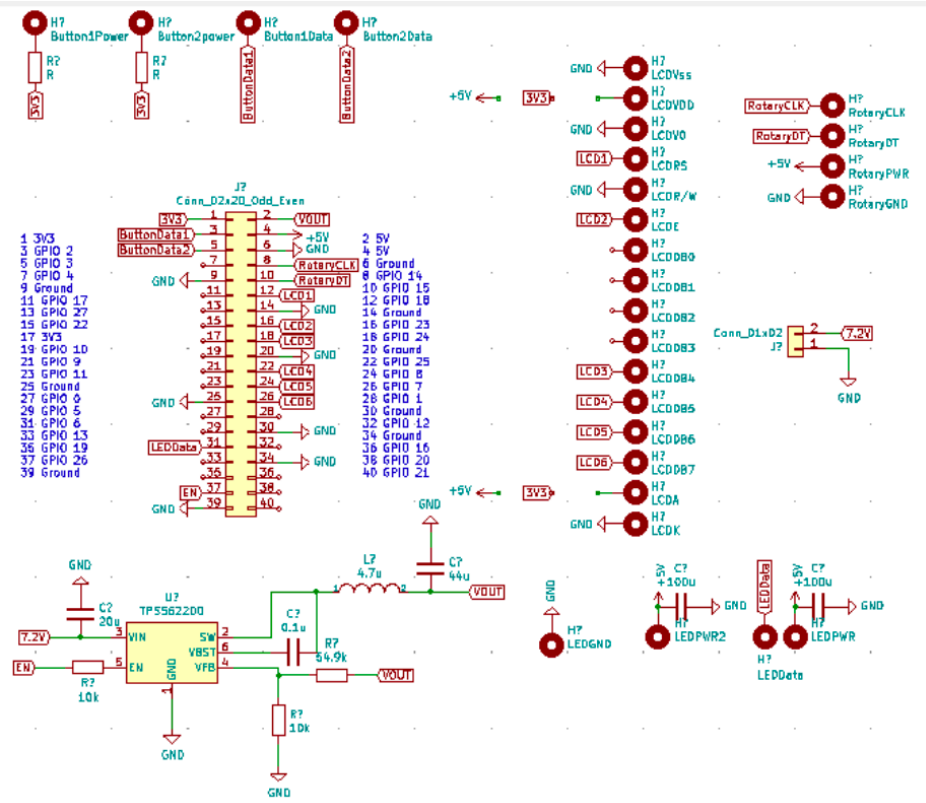


Figure 4.4. KiCAD Schematic for Guitar Learning Tool

5. References

[1] “Golf Industry - Statistics & Facts,” Statista. [Online]. Available: https://www.statista.com/topics/1672/golf/. [Accessed: 3-April-2020].

[2] “Popularity of Golf Around the World,” Topend Sports. [Online]. Available: https://www.topendsports.com/world/lists/popular-sport/sports/golf.htm. [Accessed: 3-April-2020].

[3] “Alphard eWheels Club Booster Electric Push Cart Conversion Kit,” Motogolf.com. [Online]. Available: https://www.motogolf.com/products/. [Accessed: 3-April-2020].

[4] “CaddyTrek Mobile Autonomous Robotic Golf Cart Caddy,” Robot Shop. [Online]. Available:https://www.robotshop.com/en/caddytrek-mobile-autonomous-robotic-golf-cart-caddy-black.html. [Accessed: 3-April-2020].

[5] “Arduino Uno R3” Amazon.com [Online] Available: https://www.amazon.com/Arduino-A000066-ARDUINO-UNO-R3/dp/B008GRTSV6. [Accessed: 14-April-2020].

[6] “MPU6050 Sensors,” Mouser.com. [Online]. Available: https://www.mouser.com/Sensors/. [Accessed: 17-April-2020].

[7] “NEO-6M GPS Module — An Introduction,” Electro Schematics. [Online]. Available: https://www.electroschematics.com/neo-6m-gps-module/. [Accessed: 17-April-2020].

[8] “SparkFun Bluetooth Modem - BlueSMiRF Silver,” SparkFun. [Online]. Available: https://www.sparkfun.com/products/12577. [Accessed: 17-April-2020].

[9] “MB1010 LV-MaxSonar-EZ1,” MaxBotix. [Online]. Available: https://www.maxbotix.com/Ultrasonic_Sensors/. [Accessed: 17-April-2020].

[10] “Greartisan DC 12V 1000RPM Gear Motor High Torque Electric Micro Speed Reduction Geared Motor Eccentric Output Shaft 37mm Diameter Gearbox,” Amazon. [Online]. Available: https://www.amazon.com/Greartisan-Electric-Reduction-Eccentric-Diameter/. [Accessed: 17-April-2020].

[11] “Kuman L298N Motor Drive Controller Board DC Dual H-Bridge Robot Stepper Motor Control & Drives Module for Arduino Smart Car Power Mega2560 Robot K48,” Amazon. [Online]. Available: https://www.amazon.com/Controller-H-Bridge-Stepper-Mega2560-Duemilanove/dp/B01BWLICV4?ref_=fsclp_pl_dp_2. [Accessed: 17-April-2020].

[12] “LM2575-5.0WU-TR,” [Online]. Available: https://www.digikey.com/product-detail/en/microchip-technology/. [Accessed: 17-April-2020].

[13] “TalentCell Rechargeable 12V DC Output Lithium ion Battery Pack for LED Strip/Light/Panel/Amplifier and CCTV Camera with Charger, Multi-led Indicator Black (3000mAh),” Amazon.com. [Online]. Available: https://www.amazon.com/TalentCell-Rechargeable-Amplifier-Multi-led-Indicator/. [Accessed: 21-April-2020].

[14] “IEEE Code of Ethics,” *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 3-April-2020].