Track Runner's Pacing Assistant

By
Ben Chang
David Creger
Gaurav Gunupati

Final Report for ECE445, Senior Design, Spring 2021
TA: AJ Schroeder

May 2021
Team 5

Abstract

This project aims to solve an issue that distance runners of all skill levels struggle with: pacing. We strived to create an affordable and easily accessible product to allow distance runners to train exactly at their goal pace. Our solution utilizes a cheap RC car as a base and IR sensors to follow the lane lines around a running track. The car drives at the speed entered by the user, and the runner is then able to follow behind it to have a perfectly paced workout. Workout settings are entered through a smartphone app and sent to the car via bluetooth. In this report, we outline our design decisions and components used to create this product.

Table of Contents

1 Introduction

1.1 Objective

One of the biggest problems that new distance runners face is learning to pace themselves. Whether you are a high school track or cross country runner or just a casual 5K runner, you have probably experienced this issue. The most effective way to run a race over 1500 meters is to keep a constant speed the entire time, and if you are not doing this, it will dramatically hurt your performance [1]. A lot of beginners will end up running too fast at the start of the race and have to slow way down by the end. While this issue affects beginners more dramatically, even very experienced runners have trouble with this as well. When looking to shave every possible second off of your time, perfect pacing matters a lot.

Our solution creates a miniature car that can maintain a precisely constant and adjustable speed on any standard running track in the world. It utilizes IR sensors to follow the lines around the track and connects to a smartphone app where the user can input distance, pace, and time. The runner is then able to run behind this car in order to maintain a constant pace throughout their run.

1.2 Background

Extensive research has been done on the subject, and researchers unanimously agree that maintaining a mostly constant speed throughout a distance race will lead to the fastest times [2]. There may be some slight exceptions depending on the race length and strategy, such as starting and ending the race faster than the middle [3]. Nonetheless, if a runner can develop the muscle memory for their desired race pace prior to the actual race, they can give themselves the best chance to run their fastest times.

In an effort to understand how important pacing is, even for Olympic level runners, we can analyze Haile Gebrselassie's world record attempts for the marathon. Gebrselassie broke the world record at the 2007 Berlin Marathon, but was determined to run even faster at the 2008 Dubai Marathon. Starting overly eager, he completed the first half of the race in 61:27 which was 30 seconds faster than he ran in Berlin [2]. Unfortunately, this had detrimental effects on the second half of the race, which took 63:26, and caused him to miss his own world record time by nearly 30 seconds [2]. This goes to show that even a small error in pacing dramatically affects the outcome of a race. On that day in Dubai, Gebreselassie may very well have had the aerobic capacity to break the world record again, but his eagerness to go out too fast hurt him in the end. Luckily for him, he was able to learn from this mistake and break the world record again at the 2008 Berlin marathon [3], this time with splits of 62:04 and 61:55.

This example shows why pacing devices such as GPS watches are so popular today, among serious and casual athletes alike. However, GPS watches can't provide very instantaneous feedback and are sometimes inaccurate, especially going around turns [4]. They also can't provide visual motivation like having a pace car in front of you would. These are the two issues our group was able to solve with our pacing assistant.

There is no shortage of athletes that would benefit from this technology. The NCAA estimates there are over 800,000 high school track and cross country athletes in the U.S. alone [5]. Aside from that there many more college and professional runners, not to mention the casual 5K runner that just wants to run their fastest times. Our pacing assistant can be an incredibly helpful tool to many thousands of runners across the world.

1.3 High Level Requirements

- The robot must have adjustable speed ranging from 5 to 10 mph, and be able to operate for at least 30 minutes at 6mph.
- The robot must follow all typical Olympic track lane markers at all times using IR sensors.
- The smartphone app must have a display showing set speed, distance travelled, and time elapsed. Distance, pace, and time must each be correctly displayed with an allowable error of 5%.

2 Design

Our design can be broken down into four main subsystems: speed control, steering control, power distribution and user interface.

The speed control subsystem consists of a hall effect sensor, motor driver circuit, and microcontroller. The hall effect sensor is used to measure the speed of the car, while the microcontroller calculates the duty cycle that should be sent to the DC motor to adjust the speed of the car.

The steering control subsystem includes the IR sensors, servo motor, and microcontroller. The IR sensors are used to detect the position of the lane line beneath the car. The microcontroller then adjusts the direction of the front wheels with the servo motor in order to keep the car on the line.

Finally, the user interface subsystem includes the smartphone app and bluetooth module where the user can input their desired workout settings which are sent to the RC car's microcontroller.

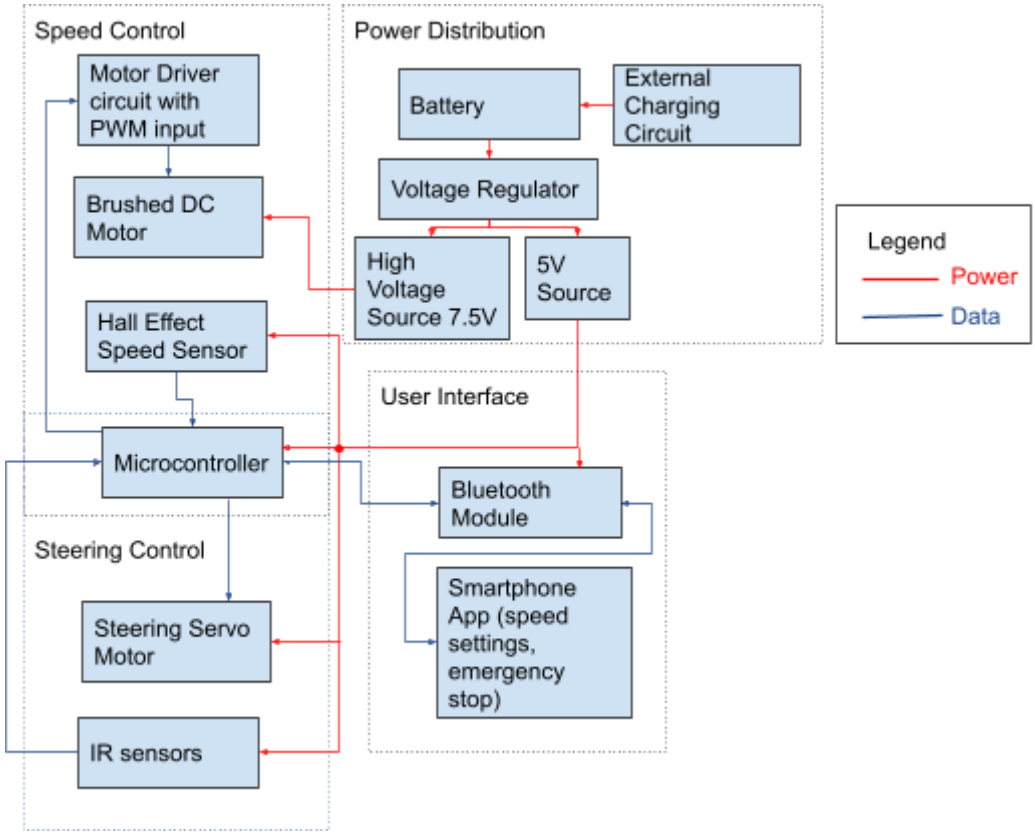

Figure 1. System Overview Block Diagram

2.1 Speed Control

2.1.1 Hall Effect Sensor

A hall effect sensor and magnet is used to measure the RPM of the car wheels and calculate the linear speed of the car. We created our own magnetic encoder by embedding a neodymium magnet in one of the rear wheels, and hanging the hall effect sensor just outside. An image of this mechanism is shown in Figure 2.The magnet passes the sensor exactly 1 time for each rotation of the wheels, so the period between each pulse from the sensor is measured and the RPM and speed are calculated. We are using omnipolar switch hall effect sensor DRV5033-Q1. Figure 4 shows the state of the output pin vs the magnetic field sensed. Figure 3 shows the circuit diagram we are using.

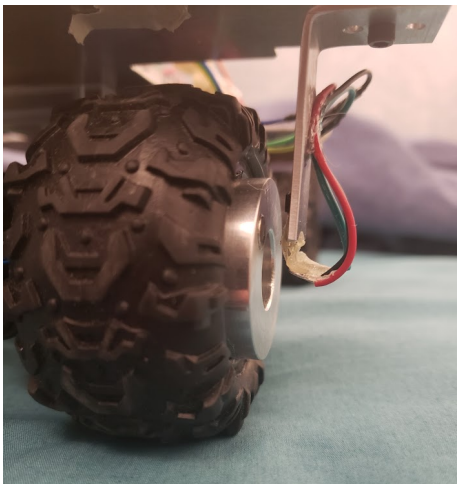

Figure 2. Hall Effect Sensor and Magnet

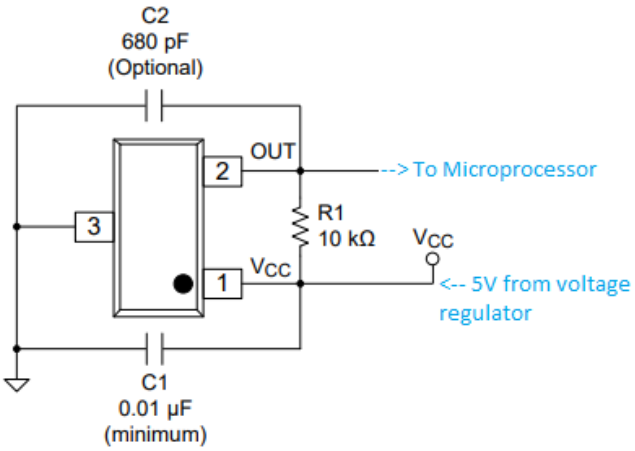

Figure 3. Circuit Diagram for Hall Effect Sensor

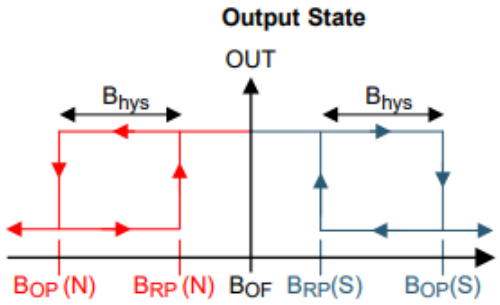

Figure 4.Output State of DRV5033-Q1

2.1.2 Motor Driver

We are implementing a single direction brushed motor control circuit to control the DC motor on the car. It is a simple circuit consisting of a MOSFET and flyback diode, which is shown in Figure 5. We are using the FQP30N06L N-Channel MOSFET which can handle up to 60V and 30A. Using this circuit, a small PWM signal from the microcontroller is amplified and sent to the DC motor. The speed of the DC motor is proportional to the duty cycle of the PWM signal, so it

is easily controllable by the microprocessor. The flyback diode prevents back EMF from the motor from damaging other parts of our circuit.

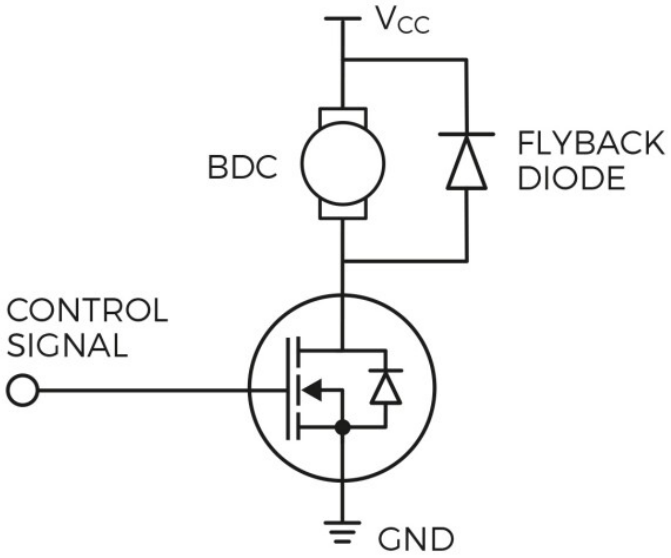

Figure 5. Circuit Diagram of motor driver[6]

2.1.2 PID control loop

The previous two subsections described how to measure the speed of the car and how to manipulate the speed of the motor. This subsection will cover how we are intelligently controlling that speed so that the car drives at the exact speed we want it to. To do this, we are using a PID feedback loop. This works by continuously calculating an error value, which is the measured RPM minus the desired RPM. We then apply correction factors based on the proportional, integral, and derivative terms of the error function. We can tune the controller with Kp, Ki, and Kd gains in order to give different weights to each of these correction terms. A basic summary of this procedure is shown below in Figure 6. This method allows for very accurate control of the RPM of the car, such that it drives at the exact speed we set it to.

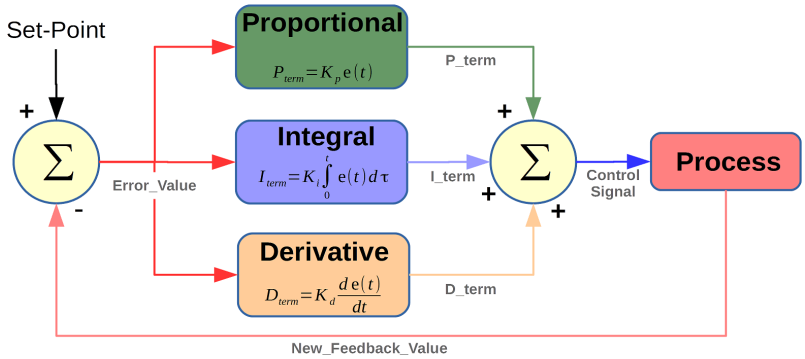

Figure 6. Basic PID diagram

This PID control loop was tested and tuned using Arduino's serial plotter. The three gain values were systematically adjusted until the measured speed settled very quickly at the desired speed without oscillations. An image of our final tuned controller is shown below in Figure 7. The red line shows the desired RPM while the blue line shows the measured RPM. We can see it has a quick settling time and almost no oscillations.

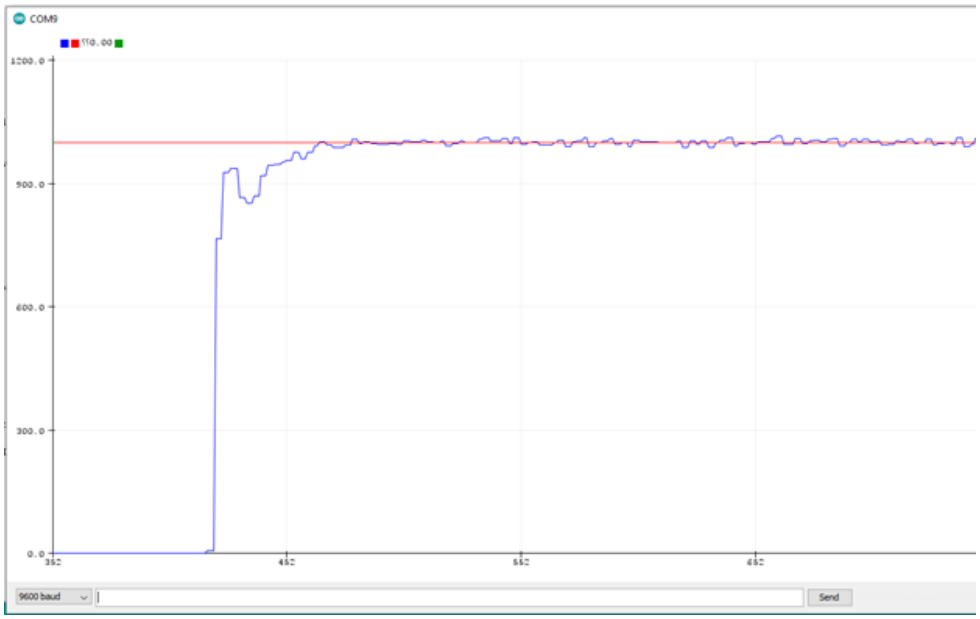

Figure 7. Serial Plotter of Measured RPM (blue) and Desired RPM (red)

Since the PID controller operates in RPM, we must first convert the incoming linear speed in mph to rotational speed in RPM. The circumference of the wheel was measured and a simple unit conversion was performed using the following formula.

$$Circumference\ of\ Wheel\ =\ 210\ mm$$

$$RPM\ = \frac{X\ mi}{hr} * \frac{1\ hr}{60\ min} * \frac{1.609E6\ mm}{mile} * \frac{1\ rotation}{210\ mm} \qquad \text{Eq. 1}$$

2.2 Steering Control

Having made line following cars in ECE110, our group knew we wanted to use a similar technique for this project. We knew we needed to use IR sensors to evaluate where the lane line was in relation to the car, and be able to correct it's course. However, we didn't know how many to use. The simplest solution was to use two IR sensors like we did in ECE110, but we weren't sure if that would suffice for our faster moving car. In an effort to be as cost effective and as simple as possible, we began testing with only 2 IR sensors, and quickly realized it would not

be sufficient. We ended up needing at least 4 IR sensors, and using the algorithm described in the following subsection.

2.2.1 IR sensors

We decided to use IR reflectance sensors to differentiate between the white lane line and dark track surface. They work by emitting IR light towards the ground and measuring how much comes back. Lighter colors reflect more light, so we can measure the analog output value from each sensor to determine whether it is above the lane line or not. The sensors we used were the SEN-11769 ROHS which are pre-packaged IR sensors. This allowed us to easily adjust the number of sensors as well as the spacing between them until we found the optimal setup. The machine shop helped us affix the sensors in the front of the car hovering just off the ground. Images of our setup are shown below in Figure 8.

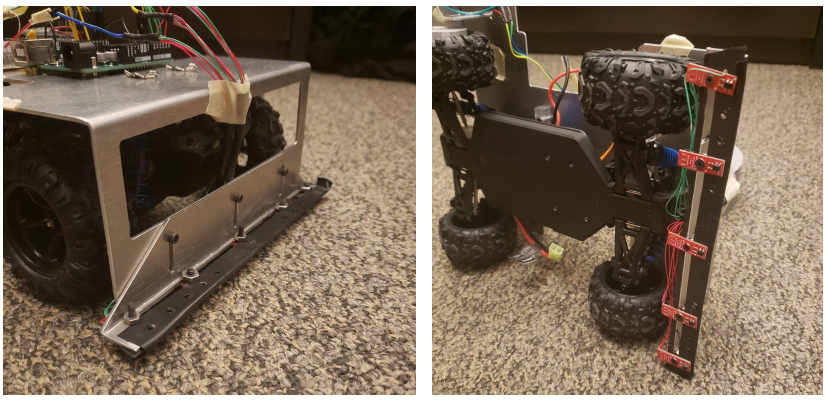

Figure 8. IR sensor Arrangement

2.2.2 Steering Algorithm

A basic line following car can be made with only two IR sensors. It would work by turning the wheels left if the left sensor saw the line, and turning right if the right sensor saw the line. This method works fine for very slow moving cars, but we quickly realized it would not work for our case. Since our car is moving quite fast, during testing the car would blow right past the lane line without having time to correct its course. We knew we needed more sensors to be able to sense the line over a wider area. Then, we could have the car start to correct earlier when it deviates from the line.

The solution we settled on was to use four IR sensors and have increasingly extreme turn radii the farther the line was from the center of the car. That way, the car would start correcting its course slightly if it was only slightly off the line, and correct its course more drastically if the line was all the way off to the right or left. This solution offered us a very smooth path that stayed on the lane line almost the entire time. It was implemented using a lookup table for servo position values depending on which IR sensors saw the line. Servo position can be easily controlled using Arduino's servo library. After testing, we determined that a servo position of 136

positioned the wheels straight. Values greater than 136 turned the wheels left and values less than 136 turned the wheels right. Figure 9 shows the lookup table that was used. A 1 indicates a sensor above the line, while a 0 indicates a non-line.

IR Sensor Values	1000	1100	0100	Everything Else	0010	0011	0001
Servo Position	140	139	138	136	134	133	132

Figure 9. Lookup table for IR Sensor Value vs Servo Position

2.3 Power Distribution

Our power distribution system consists of a 2 cell lithium polymer battery and a step down converter. The high voltage from the battery is fed straight to the motor driver circuit, while a 5V source is required to power the microcontroller, IR and hall effect sensors, as well as the bluetooth module and the steering servo.

2.3.1 Battery

A 2s 850mah LiPo battery is used to power the entire system. It's voltage measures between 8.4 and 7.0 volts depending on the level of charge.This battery is connected directly to the voltage regulator and the motor driver circuit. The battery is removable and rechargeable via an external USB charger. We used the stock battery that came with the RC car which was enough to meet our requirements.

2.3.2 Voltage Regulator

Most of our components run off of 5V, but since our battery voltage is 7.0 - 8.4V, we need a voltage regulator to step the voltage down. The best choice for us was to use an LDO, which is small, simple, and cheap compared to a switching buck converter. The model we chose to use was the NCP1117 5V LDO from Mouser. It can handle up to 1A of current which was more than enough for our needs. A schematic of the voltage regulator circuit is shown below.

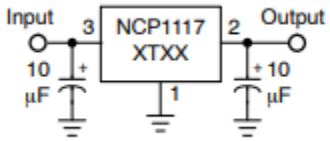

Figure 10. NCP1117 Voltage Regulating Circuit

2.4 User Interface

The UI was implemented using a simple phone app and bluetooth receiver module. This is the main way for the user to interact with the RC car. The user is able to input their workout plan using the phone app. The time left and current distance travelled are displayed on the app and an emergency stop button is available in case the user would like to abort their current workout.

2.4.1 Bluetooth Module

A HC - 05 bluetooth receiver is used as a gateway between the RC car and our smartphone app. The receiver communicates directly with the ATmega328P Microcontroller to allow a two way communication path. The workout plan is inputted through the UI and sent to the microcontroller while workout feedback is sent from the microcontroller to the UI.

2.4.2 Phone app

There were a few different options for how we wanted to control the RC car. These were either to use a remote controller, use a smartphone app, or to add speed adjustment buttons to the RC car itself.

We decided to use the smartphone app because of a few large advantages. One of which is remote operation. Having some way to remotely stop the car is essential for both the safety of the user, and those that share the same track. It allows the user to stop if they are unable to keep up. The smartphone app also has the advantage of being more accessible than a remote control, as most users would already be carrying a smartphone in their pocket.

The smartphone app itself was created using the online tools available at MIT app inventor[8]. It allows users to create smartphone apps with a very low barrier of entry. It offers a similar structure to SCRATCH, where you can click and drag different code segments to the corresponding elements on the UI to add functionality.

For the app itself, there are four main parts to the UI:

- Emergency stop: This is an important part of the project that focuses on an essential safety feature. It is one simple button. In the rare case that the RC car goes off track or is enroute to colliding with someone/something, this button can be used to cut power and force the car to stop.
- Workout settings: The user can set their desired speed by inputting either minutes per mile, or miles per hour. The user can also input their preferred workout time, and the car will automatically stop when the time is reached.
- Total distance: This is a feedback system that comes from the car's speed sensor. This information is used to keep track of how far you have already traveled in your workout.

The code for the app can be found in *Appendix C*

Some issues that occurred when implementing the smartphone app were that the microcontroller only accepts one byte at a time. In order to solve this problem, the smartphone app will send a "\n" character to the RC car whenever it is done sending a message, in this case, the set speed. The microcontroller then keeps track of the current message being input, and finalizes the input when this character is seen.

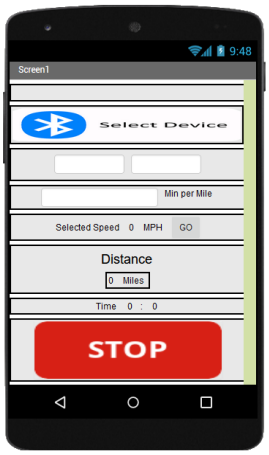

Fig 11. Layout of the smartphone app

2.5 Printed Circuit Board

Our group planned to use a PCB to interface all the subsystems described above into one neat package. We did all of our testing on solderless breadboards and attempted to transfer the design to our PCB at the end. Unfortunately, it was not a smooth transition, and we were unable to get the PCB operating by the time of our demo. This was due to several setbacks including having limited lab access and no track access. With a little more investigation, we would likely be able to diagnose the issue and correct it. Nonetheless, the schematics and images of our PCB are shown below.

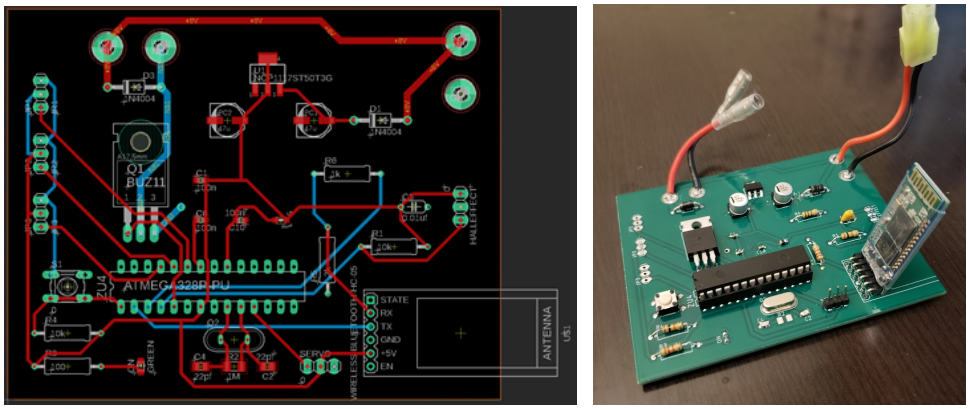

Fig 12. Custom PCB images

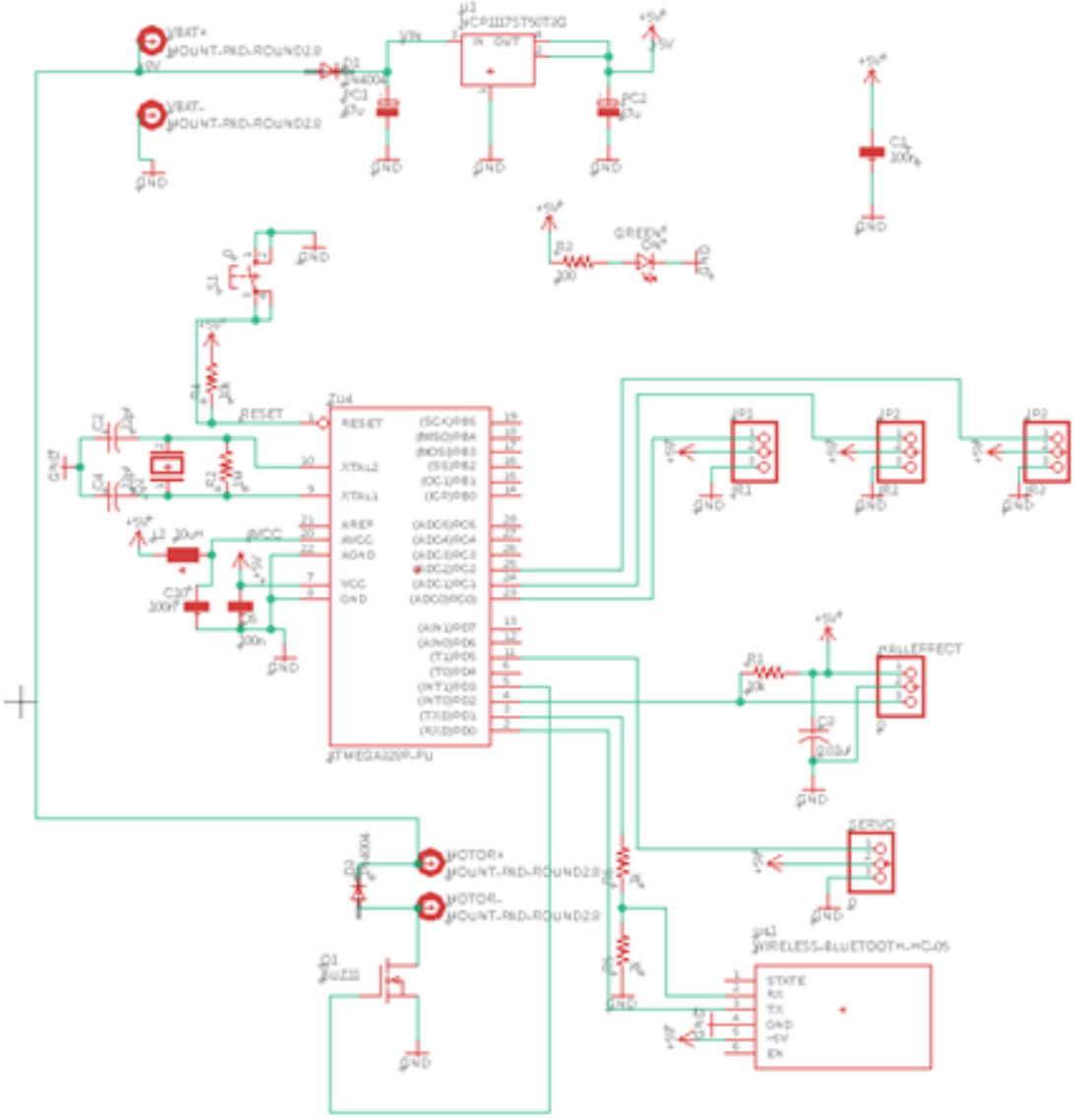

Fig 13. Custom PCB Schematic

3 Design Verification

The table in *Appendix* A highlights the various requirements and verifications that we needed to complete in order to consider this project a success. There were some slight alterations to the requirements table from when we originally submitted in the design document. One of these is the adjustment from speeds between 1 and 10 mph, to between 5-10 mph. The average walking speed is between 2 and 4 mph[7], so we felt that 5 mph was the lowest speed that we could consider jogging. A few methods of verification were also changed, but still accomplished the task of checking if the requirement was met or not.

3.1 Speed Control Verification

Our first high level requirement states we want the car's speed to be adjustable between 5 and 10 mph, and to be accurate within 5% of the desired value. Our initial plan was to test this on a running track, and time how long it took the car to travel 100m. However, we were never able to get access to a running track, so we completed the test on 35m of white tape in an ECEB hallway. The results of this test are shown in the table below. We can see that the speeds were accurate within 4%, which satisfies our requirement.

Speed	Calculated Time	Actual Time	Percent Error
3 mph	26.10 sec	26.91 sec	3.1%
4 mph	19.57 sec	20.06 sec	2.5%
5 mph	15.66 sec	16.27 sec	3.9%

Table 1. Speed Verification

3.2 Steering Control Verification

Our second high level requirement states that we want the car to follow all Olympic lane markers at all times. As stated previously, we were never able to get access to a running track to perform these tests on, so we resorted to testing on a piece of white duct tape in the ECEB. Figure 14 shows an image of our testing setup. During our tests, the car was able to follow the curved piece of tape for the entire length of the hallway, when it's speed was under 6mph. However, it started to become less reliable when we increased the speed of the car past 6mph. The car was still able to mostly follow the line, but would occasionally drift off it. We suspect that the line following would be much more reliable on an actual running track since the curves are extremely broad, and also the rubber surface would provide more traction for the car. Nonetheless, we consider this test a success.

Fig 14. Steering Verification Test

3.3 Battery Test

The final test that was conducted was a battery test. Our high level requirements states the car must be able to travel at least 3 miles on a single charge. Our group's plan was to use the stock battery that came with the RC car for testing, and upgrade to a larger battery later if need be. Luckily the stock battery was more than enough to meet this requirement. The car was set up on a test stand, and set to run for 30 minutes at 6mph, while the voltage of the battery was monitored. The below screenshots show the car was successfully able to travel 3 miles in 30 minutes. At this time, the voltage of the battery had only dropped to 7.4V, indicating there was more charge in the battery. However, the test was stopped after 30 minutes of running.

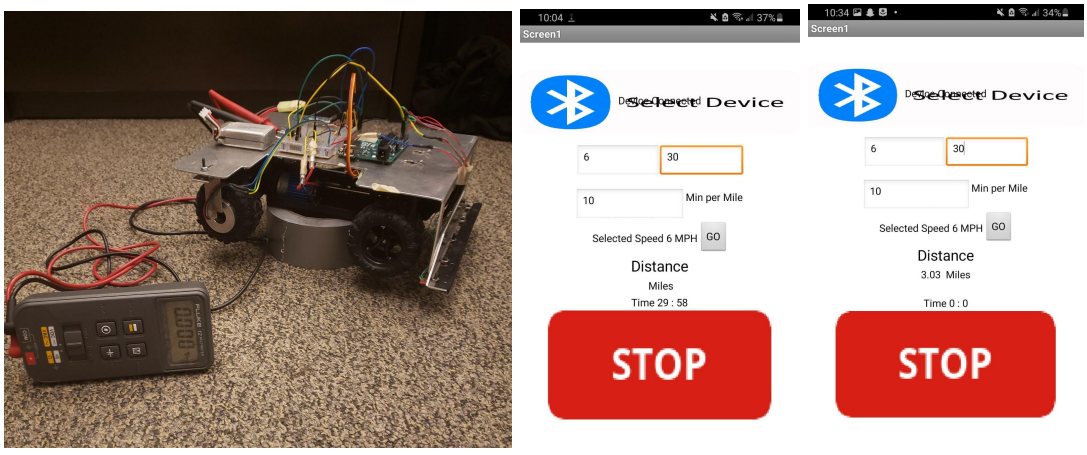

Fig 15. Battery Test

4 Costs

The costs include both the cost of parts, as well as the cost of Labor.

4.1 Parts

The cost of parts is shown in the table below

Part	Cost
DEERC RC Cars 9300 High Speed Remote Control Car	$ 79.99
HC - 05 Bluetooth module	$ 7.99
SEN-11769 ROHS IR sensor	$ 2.95 x 5
DRV5033-Q1 Hall Effect Sensor	$ 0.73
ATmega328P Microcontroller	$ 2.29
COM-10213 ROHS MOSFET	$ 0.95
ILSB0805ER100K Inductor	$ 0.24
1N4007G Diode	$ 0.23 x 2
NCP1117ST50T3G LDO	$ 0.56
EEE-FK1V470AL Capacitor	$ 0.45 x 2
C0603C104K5RAC3121 Capacitor	$ 0.28 x 3
ECS-160-18-4X-CKM 16Mhz Crystal	$ 0.53
C0805C220J4GACTU Capacitor	$ 0.24 x 2
Total	$110.71

Table 2. List of parts and prices

4.2 Labor

Our labor costs are estimated to be $40/hr and 10 hours/week for 3 people. Since the semester is 16 weeks long, the total labor cost is as follows

$$\left(\frac{3\ people}{}\right)\left(\frac{16\ weeks}{}\right)\left(\frac{\$40}{hour}\right)\left(\frac{10\ hours}{week}\right)(2.5) = \$48,000$$

Eq. 2

4.3 Total

Adding together the costs of labor and parts brings our project to a grand total of $48,110.71

5 Conclusion

5.1 Accomplishments

The project is fully functional in a lab environment and satisfied two out of the three high level requirements. The car is able to vary speeds between 5mph and 10mph while having the battery last at least 3 miles. There is a fully functioning android phone app that allows any user to control the car by setting a particular speed and workout time. The app also displays distance covered in real time. The second requirement was not satisfied as track access had been denied throughout the semester. To compensate, a 35m white tape was set on the floor in the ECEB and the car worked perfectly here so we arguably hit all three requirements. Although not a typical olympic track, the car is able to follow a white line on a marble floor.

Both the hardware and software aspects of the project were relatively new to all of us. We did not have much experience in app making, bluetooth integration or even steering and motor controls. However we divided up the work and got a head start on it which allowed us to complete the project as hoped. By the end of it we not only fully understood our individual components but also comfortable with any other aspect of the project.

The project was very rewarding and taught us a lot more than just the technical aspects. We learned to collaborate and find solutions when we had differences in opinions and overall learnt how to work as a team, a very important skill to have.

5.2 Ethical considerations

There were inherent risks to having a project which utilizes its own power system. The biggest is the method we store power for the operation of our robot. We chose to use a Lithium-Ion battery, however, these batteries come with inherent risks if precautions are not taken [9]. Lithium-ion batteries may explode if handled improperly. In order to reduce risks, we tested our charging circuit to ensure that the battery does not achieve voltages higher than the manufacturer's specifications. The battery was placed towards the end of the car so that padding would reduce impact if the robot were to run into objects or people, in order to prevent damages due to physical impact. Other problems could have also occurred if the battery reached temperatures so high which could have led the battery to fail or catch on fire. In order to mitigate this risk, we monitored the temperature of the battery at each stage to ensure that the temperature did not exceed 45°C. A feature that was not implemented but should have been is a real time tracker to check the battery temperature and report back to the user instantly.

Although the design is autonomous, it did not utilize machine learning or artificial intelligence, instead it solely relied on predefined use cases. However, this is still an autonomous vehicle and we must mitigate harm [10]. The biggest consideration is to prevent the vehicle from running into other people that are also running on the track. To do this, we implemented an emergency stop button for the user to press in the case the robot veers into harm's way. In addition we also added a case that will cause the car to stop in case it runs off the track. A final feature that can be added are ultrasonic sensors to signal the robot to stop if it detects anything within 2 meters in front of it.

We followed IEEE's code of ethics #1: "To uphold the highest standards of integrity, responsible behavior …" [11]. In doing so we tried to mitigate inherent risks of autonomous vehicles and reduce the chance of injury as much as possible.

5.3 Future work

Although the car perfectly hit all the requirements given our resources, it is by no means perfect. There are a lot of improvements that can be added to further better this project. The first improvement would be to clean up the aesthetic of the robot by switching out the breadboard with a PCB. This will not only make the car look better but also the soldered wires will be a lot sturdier and it will be less likely to break.

The next step would be to improve upon the IR sensors and steering controls. Right now the steering algorithm is very basic and might not be able to handle tracks with turns steeper than a typical olympic track. To do this, more IR sensors need to be added to have a much smarter steering control algorithm. Moreover, the car is not automatically able to calibrate the IR sensor cutoff values based on the track color. A future feature would be to add a calibrate option to allow the user to set the sensor values from the phone app itself without having to do it manually.

There are a few software fixes that can be implemented to make the app better. The first fix would be to add a pause feature. This would allow the user to pause their workout at any time and continue from that point on. Finally there is a glitch that causes text to overlap once the bluetooth device is selected. Fixing this would be the last straw in making the project close to perfect.

References

[1] J. Gaudette, “Learn to Pace Like a Pro,” Runner's World, 18-Oct-2012. [Online]. Available: http://www.runnersworld.com/advanced/a20847773/learn-to-pace-like-a-pro/. [Accessed: 05-Mar-2021].

[2] Ross Tucker, “Haile Gebrselassie: 2nd fastest time ever // World marathon record eludes Gebrselassie in Dubai,” The Science of Sport, 20-Jan-2008. [Online]. Available: https://sportsscientists.com/2008/01/haile-gebrselassie-2nd-fastest-time-ever/#:~:text=His%20first%2010km%20was%20run,than%20in%20Friday's%20Dubai%20race.

[3] “Haile shatters own world record in Berlin,” BMW-Berlin-Marathon, 28-Sep-2008. [Online]. Available: http://www.bmw-berlin-marathon.com/en/news-center/news/detail/haile-shatters-own-world-record-in-berlin/. [Accessed: 05-Mar-2021].

[4] “How reliable are GPS watches in tracking YOUR Pace? A look at the scientific literature,” 09-May-2016. [Online]. Available: https://runnersconnect.net/reliable-gps-watches-running/. [Accessed: 05-Mar-2021].

[5] “Estimated probability of competing in college athletics,” 16-Apr-2020. [Online]. Available: http://www.ncaa.org/about/resources/research/estimated-probability-competing-college-athletics#:~:text=Nearly%20eight%20million%20students%20currently,the%20professional%20or%20Olympic%20level. [Accessed: 05-Mar-2021].

[6] “Brushed DC motors and how to drive them.” [Online]. Available: https://www.diodes.com/design/support/technical-articles/driving-brushed-dc-motors/. [Accessed: 05-Mar-2021].

[7] S. Paul, “What Are the Right Walking and Running Speeds?,” Runner's World, 07-Mar-2013. [Online]. Available: https://www.runnersworld.com/runners-stories/a20844065/what-are-the-right-walking-and-running-speeds/. [Accessed: 05-Apr-2021].

[8]“With MIT App Inventor, anyone can build apps with global impact,” MIT App Inventor | Explore MIT App Inventor. [Online]. Available: https://appinventor.mit.edu/. [Accessed: 05-Apr-2021]

[9] Occupational Safety and Health Administration. (1970). Preventing Fire and/or Explosion Injury from Small and Wearable Lithium Battery Powered Devices (shib 011819). Available: https://www.osha.gov/dts/shib/shib011819.html [Accessed: 17- Feb- 2021].

[10] C. E. Berecz and G. Kiss, "Dangers in autonomous vehicles," 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 2018, pp. 000263-000268, doi: 10.1109/CINTI.2018.8928189.

[11] “IEEE code of ethics.” [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 05-Mar-2021].

Appendix A
Requirement and Verification Table

Table 3 System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
1. Car drives at target speed within 5% accuracy a. Hall effect and motor driver and PID control loop working together in conjunction 2. Operates at speeds between 5-10 mph	A. Place the car onto a premeasured track and time the car different speeds a. Using a 35m track, we measured the time it took for the car to reach the end at 4.5 mph, and 3.5 mph B. Set the user defined speed at 5 to 10 mph, then check if the RPM matches the speed	Y
1. The robot is able to follow track lane lines. a. Tests the IR sensor system as well as the steering algorithm	1. Place the RC car onto an example track a. Run the car on the predefined track at the minimum speed b. Check to see if the car stays on the line	Y
1. The car is able to drive for 3 miles on a single battery charge	1. Place the car on a stand, and run the car at 6 mph for 30 mins. a. Check if the battery is dead by the end of the test b. Check if the phone app properly reads 3 miles driven	Y
1. The phone app is able to properly start and stop the car 2. The speed inputted is properly sent to the RC Car 3. Accurately receives the distance traveled value from the RC car	1. Read the input from the microcontroller using Serial.print() 1. Set a low time, and see if the RC car stops when the timer hits 0 2. Place car on a premeasured track and see the distance recorded when the car reaches the end	Y

Appendix B Microcontroller Code

```
#include <Servo.h>
Servo servo;
int position = 137;        //Value of Servo motor for straight wheels
int Left2, Left1, Center, Right2, Right1;    //IR sensors
int Line = 670;            //Analog value separating line from non-line
int print = 0;

char Incoming_value[7] = "0.0000";          //Initializes incoming bluetooth value so that desired speed = 0

volatile long EncoderCount = 0;                     //variable used to track distance car has travelled
float RPM = 0, RPM_desired = 0;                       //initializing measured RPM and desired RPM
unsigned long t = 0, t_prev = 0, DeltaT = 50000;         //initializing RPM timing variables. DeltaT set so we don't get a divide by 0
error
int safety = 0;
float DutyCycle = 0;                                 //initialize duty cycle variable
char SR;                //serial read variable
int x = 0;
float speed;             //speed in mph


float error, error_prev = 0, integ_err, integ_prev = 0;   //initializing PID variables

float kp = 0.5;              //PID gain values
float ki = 0.1;
float kd = 0;

void TimerInterrupt() {        //measure how long in between each successive pulse from hall effect sensor
  t = micros();
  if (t - t_prev > 30000){     //Some interrupts would happen twice in a row. Probably due to jitter from the sensor. Enforce that pulses
are at least 30ms apart.
    DeltaT = t - t_prev;
    t_prev = t;
    EncoderCount++;
    print = 0;
  }
}


void setup() {
  pinMode(2, INPUT);            //Hall effect sensor input
  pinMode(3, OUTPUT);           //Motor Driver PWM signal output
  attachInterrupt(digitalPinToInterrupt(2), TimerInterrupt, RISING);     //goes to interrupt function TimerInterrupt() on each rising edge
of the hall effect output
  delay(1000);
  Serial.begin(9600);
  servo.attach(5);              //sets servo PWM input pin
  servo.write(position);            //set value for straight wheels
}


void loop() {


while(Serial.available() > 0){            // Checks whether data is coming from the serial port
  SR = Serial.read();
```

```
  if (SR != '\n'){              //checks for end character
    Incoming_value[x] = SR;
    x++;
  }
  else {
    Incoming_value[x] = '\0';     //adds end character to string
    x = 0;
    speed = atof(Incoming_value);   //converts character array to floating point number
    if(speed < 2){              // sets motor speed equal to 0 if speed is less than 4mph
      speed = 0;
    }
    if(speed > 15){
      speed = 15;
    }
    RPM_desired = speed*0.2682233333*476;
    break;
  }
}


RPM = 60000000 / DeltaT;              //calculates RPM from DeltaT

if(micros() - t_prev > 500000){       //sets RPM to 0 if we get no pulses in 0.5 seconds
  RPM = 0;
}

error = RPM_desired - RPM;                //calculates error
integ_err = integ_prev + (0.0000001*DeltaT * ((error + error_prev) / 2));     //trapezoidal method to calculate integral of error function

if(RPM_desired == 0){       //enforces that integral is 0 when desired rpm is 0 so that the car stops
  integ_err = 0;
}

DutyCycle = kp*error + ki*integ_err + (kd * (error - error_prev) / (DeltaT*0.0000001)) ;    //Calculates duty cycle based on P I and D
gains and error

if (DutyCycle > 255){            //Anti-wind up. Caps duty cycle at 255 and maintains error integral so it doesn't keep climbing to infinity
  DutyCycle = 255;
  integ_err = integ_prev;
}

if (DutyCycle < 0){              //prevents negative duty cycles
  DutyCycle = 0;
}



analogWrite(3, DutyCycle);        //writes duty cycle to motor driver circuit

integ_prev = integ_err;          //stores current integral as previous
error_prev = error;              //stores current error as previous


Left2 = analogRead(2);              //reads IR sensor values
Left1 = analogRead(1);
Center = analogRead(4);
Right1 = analogRead(3);
Right2 = analogRead(0);
```

```
if (Left1 < Line && Left2 > Line && Right1 > Line && Right2 > Line){        //Turns wheels right if right sensor sees line
position = 139;
}
else if (Left1 < Line && Left2 < Line && Right1 > Line && Right2 > Line && Center > Line){        //Turns wheels right if right sensor
sees line
position = 140;
}
else if (Left1 > Line && Left2 < Line && Right1 > Line && Right2 > Line && Center > Line){        //Turns wheels right if right sensor
sees line
position = 141;
}

else if (Left1 > Line && Left2 > Line && Right1 < Line && Right2 > Line){        //Turns wheels right if right sensor sees line
position = 134;
}
else if (Left1 > Line && Left2 > Line && Right1 < Line && Right2 < Line && Center > Line){        //Turns wheels right if right sensor
sees line
position = 133;
}
else if (Left1 > Line && Left2 > Line && Right1 > Line && Right2 < Line && Center > Line){        //Turns wheels right if right sensor
sees line
position = 132;
}

else{
  position = 136;
}


servo.write(position);


if (EncoderCount%25 == 0 && print == 0){
  Serial.println(EncoderCount*0.000130488, 2);      //EncoderCount*0.000130488
  print = 1;
}




}
```

Appendix C Smartphone app Code

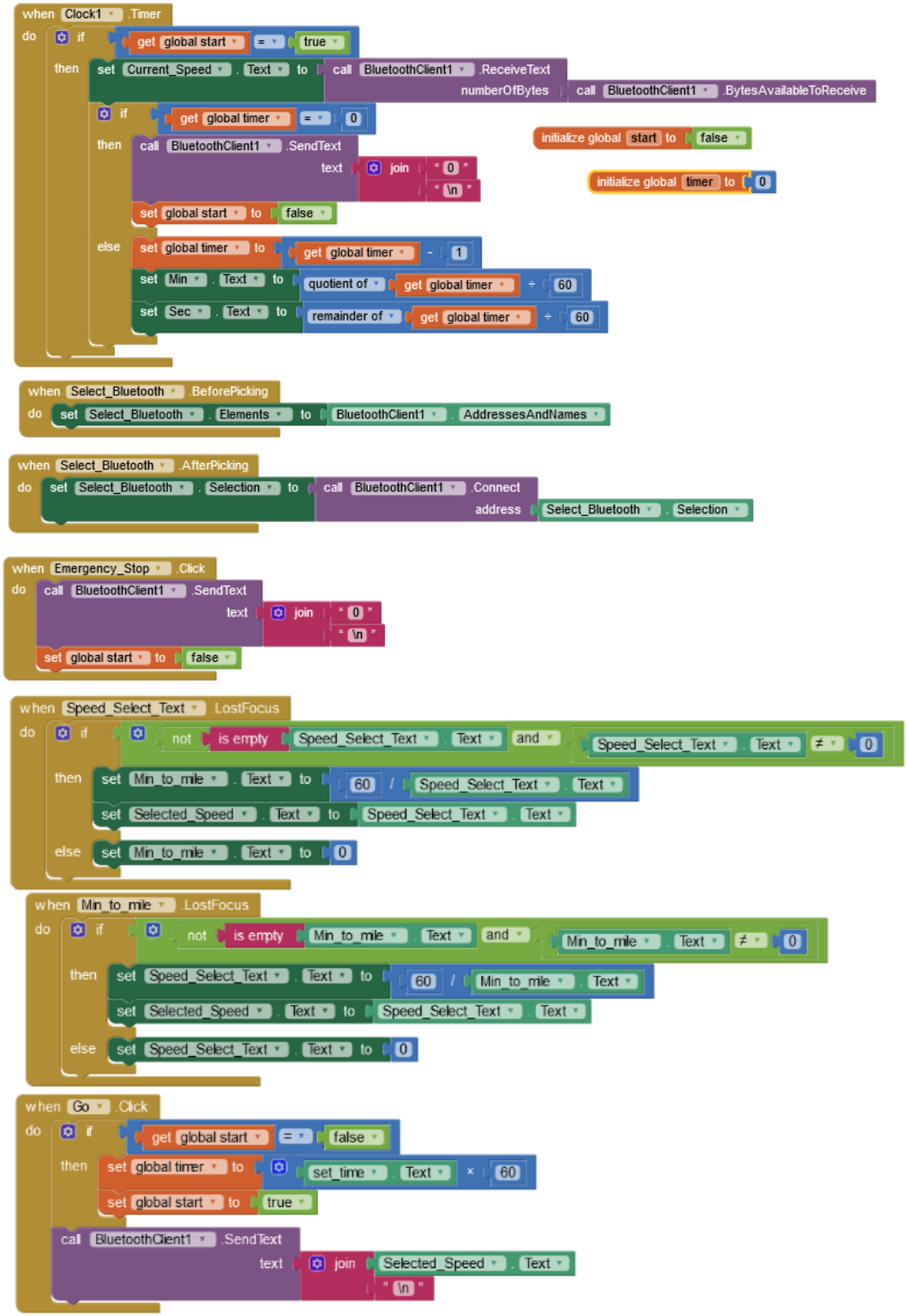

Figure 16. Smartphone app Code