The Physics Appreciator

Jason Romanof and Zain Glover

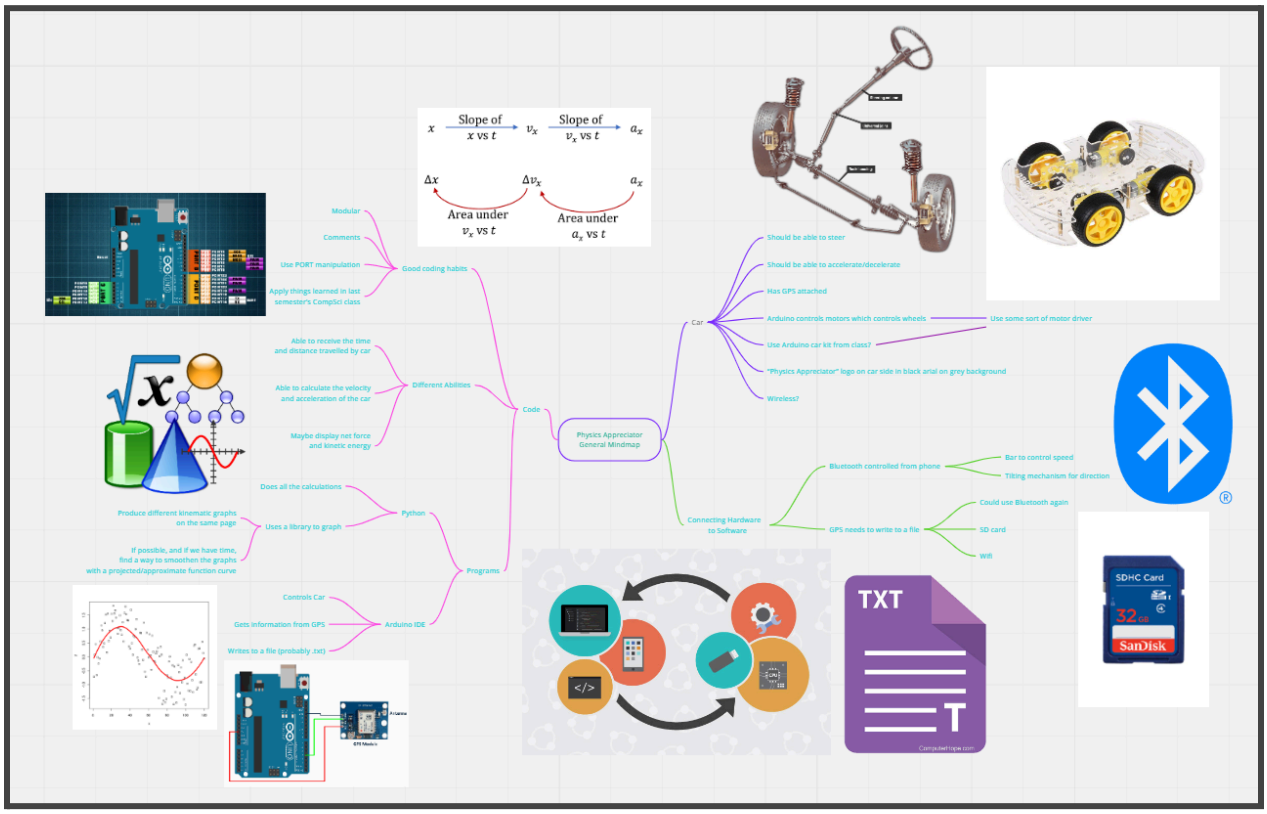
TEJ3M1/P-01

June 20, 2022

Cycle #1:

Step 1: Define:

The point of this cycle is to get an idea of the things we want to accomplish and set a base for ourselves. We want to create a program that analyzes the kinematics of a car.



https://miro.com/app/board/uXjVOsdWG9M=/?share_link_id=348358051988

Plan:

1. Draw a sketch of the first prototype.
2. Build and wire the Arduino car with GPS. We won't be able to use tinkercad since it doesn't have a GPS component but realistically the wiring shouldn't be too complicated.
3. Write the Arduino code; learn how to use the Adafruit library. We just need to make sure that the car is able to move smoothly.
4. Create the bluetooth control for the car.
5. Write the Python code; receive serial data and print it on to a Matplotlib graph using CoolTerm.

Step 2: Research:

Things to research (Blue = Zain, Purple = Jason):

- How to power an Arduino Wirelessly
- Different GPSs and how they work
- How to write data to the computer (Will we need to use an SD card?)
- How to use a phone and bluetooth to give input to an Arduino
- How to assemble car
- Python graphing libraries

Links:

- **https://www.youtube.com/watch?v=Q36NbjPMV5k**
- **https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/**
- **https://www.youtube.com/watch?v=kwk3qzaIcCU**
- **https://www.youtube.com/playlist?list=PLGs0VKk2DiYx6CMdOQR_hmJ2NbB4mZQn-**
- **https://www.youtube.com/watch?v=Iz_AETY9o5E&t=61s**
- **https://www.youtube.com/watch?v=sXs7S048eIo**
- **https://www.youtube.com/watch?v=OhnxU8xALtg**
- **https://www.youtube.com/watch?v=NXlyo0goBrU&t=610s**
- **https://www.youtube.com/watch?v=GulmSO4Iqk8**
- **https://docs.arduino.cc/learn/built-in-libraries/software-serial**
- **https://www.youtube.com/watch?v=dyjo_ggEtVU&t=1914s**
- **https://www.youtube.com/watch?v=UO98lJQ3QGI&list=PL-osiE80TeTvipOqomVEeZ1HRrcEvtZB_**

Things Established:

- We will be using the Adafruit Ultimate GPS [ADA746]:
    - Specs can be found at https://www.adafruit.com/product/746
- It looks as though the GPS can only get a fix when outside. We will need to do any testing at lunch!
- In terms of the frame of the car, we can use the one found in class. This design doesn't allow for conventional turning (like that of a car) so we will need to experiment with this aspect (maybe turn it like a tank?).
- Matplotlib is a very useful way to plot graphs using Python. There are other libraries that go hand in hand with Matplotlib but look a bit complicated in their concepts. We might consider using them in the future once we have a basis for our code established.
- We will use Roger Meier's CoolTerm software to receive serial data and then send it to a text file. The software is especially useful as we can use Python to directly read from the text file and produce graphs.
    - http://freeware.the-meiers.org/

Imagine:

- **Base design:** Arduino car with a GPS that can move forward and backward. Car is wired to a computer where its GPS data can be interpreted to analyze the Car's basic kinematics. These kinematics include the Car's acceleration, velocity at any point, and position. We can calculate this using the time interval between the data received and the speed given by the GPS using Euler's method.
- All other capabilities will be structured off the base design. Possible capabilities include:
    - Allowing the Car to steer/turn
    - Analyzing the kinetic and gravitational energy of the car
    - Analyzing the static/kinetic friction of the car
    - Building our own frame for the car (this would be extremely difficult)
    - Making the car wirelessly send back serial data
    - Add LEDs

- Later on in different cycles we can add certain capabilities: at this point we really care about whether we can create a foundation for our project.

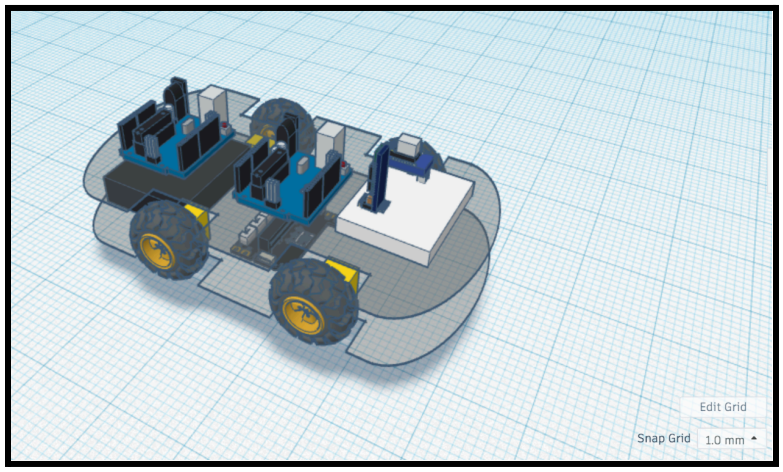
Step 3: Prototype:

Product List:

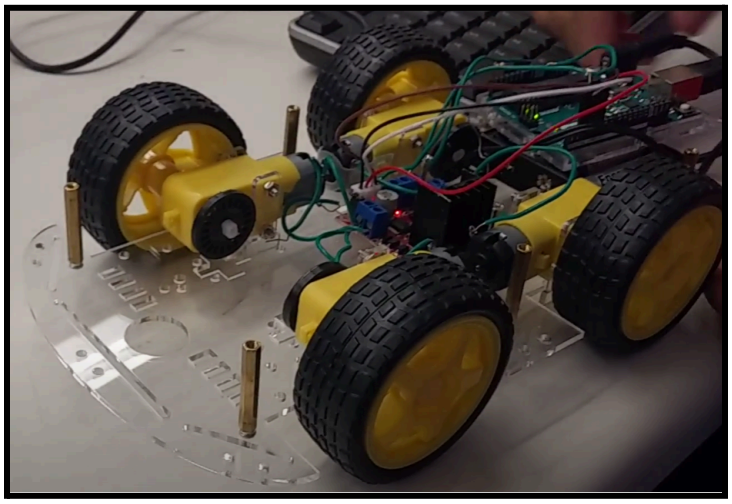
- 2 Arduino Unos
- 2 Car Plates
- 2 Arduino USB Connectors
- 2 Computers
- Wires
- 4 Wheels
- 4 Motors
- Screws and bolts
- Adafruit Ultimate GPS
- HC-05 Bluetooth Module
- Battery Pack
- Duct Tape
- Screwdriver
- Wirecutter

Sketches:

- https://www.tinkercad.com/things/3FsTNeFhrSM-schematic-1-tej301/edit?sharecode=mmPl_oCnOdbAghpbPlKAbgsOXn1vIawZ5TenAgzKlS8



Car Photos:



Step 4: Code:

Car/Bluetooth code:

```
#include <SoftwareSerial.h>

const uint8_t rxPin = 6;
const uint8_t txPin = 5;

const uint8_t motor1forward = 8;
const uint8_t motor1backward = 9;
const uint8_t motor2backward = 10;
const uint8_t motor2forward = 11;

char command;

SoftwareSerial BTSerial(rxPin, txPin);

void setup() {
  Serial.begin(9600);
  BTSerial.begin(9600);

  DDRD = 0b00100000;
  DDRB = 0b001111;
}

void loop() {
  while (BTSerial.available()) {
    command = BTSerial.read();
    Serial.write(command);
    if (command == 'F') {
      digitalWrite(motor1forward, HIGH);
      digitalWrite(motor2forward, HIGH);
      digitalWrite(motor1backward, LOW);
      digitalWrite(motor2backward, LOW);
    }
```

```
    if (command == 'B') {
      digitalWrite(motor1forward, LOW);
      digitalWrite(motor2forward, LOW);
      digitalWrite(motor1backward, HIGH);
      digitalWrite(motor2backward, HIGH);

    }

    if (command == 'L' or command == 'R') {
      PORTB = 0;
    }
  }
}
```

Python code and its output:

```
# Uncomment this code to fill the .txt file with random speeds. This shows the
# Capabilities of the matplotlib library.
'''
import random as r

speed_file = open("NMEA.txt", "w")

for i in range(1000):
    speed = r.uniform(0, 3) # Random float within range
    speed = str(speed)

    speed_file.write(speed)
    speed_file.write("\n") # Create a new line

speed_file.close ()
'''

# Uncomment this code to fill the .txt with a linear speed (or change the speed
# Equation to test other functions). It should be clear that speed is
# The derivative of distance, and acceleration the derivative of speed.
'''
speed_file = open("NMEA.txt", "w")

for i in range(1000):
    speed = 0.5 * i + 10
    speed = str(speed)

    speed_file.write(speed)
    speed_file.write("\n") # Create a new line

speed_file.close ()
'''


from matplotlib import pyplot as plt

speed_file = open("NMEA.txt", "r") # Open the GPS data as a read only file

time_axis = []

accelerations = []
speeds = []
distances = [0]  # Set inital position (reference point) as 0 m

d = 0 # Initial distance (reference point)
h = 0.01 # Step size or time interval between each outputed speed
         # From GPS which will be recieved through experiment.
         # Letter h is convention for small values; same as dt or dx.

ctr = 0
```

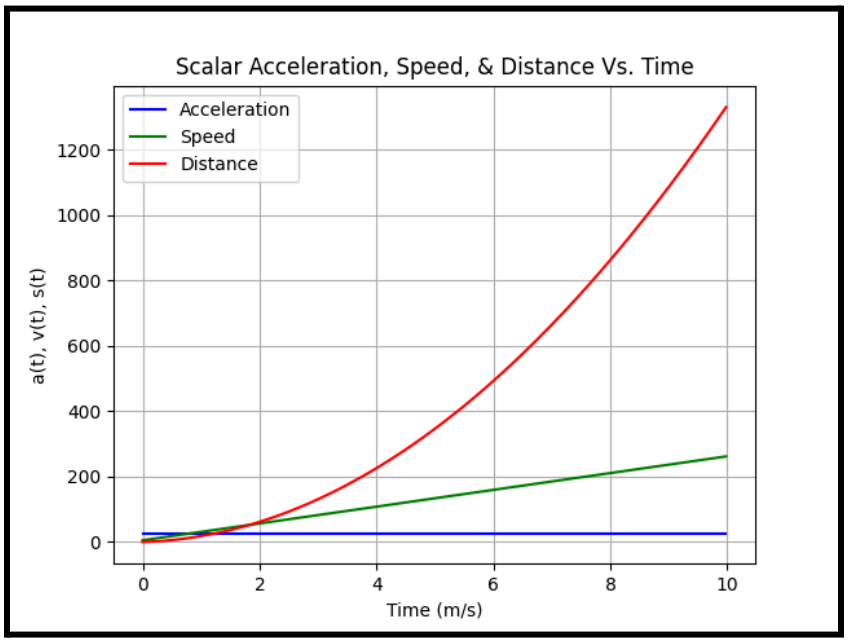
```
for v in speed_file:
    speeds.append(float(v) * 0.514444) # Convert knots to m/s

    time_axis.append(ctr * h)
    ctr += 1

speed_file.close()


# To find the accelerations, we compare two consecutive velocities and find
# The slope, (dv/dt) that connects them. Since the velocities will have a very
# Small time interval between them, the accelerations will appear to be
# Instantaneous. Since we have to compare two speeds for one acceleration value,
# The lenght of the acceleration array will be one less than that of the speeds.

for j in range(len(speeds) - 1):
    a = (speeds[j+1] - speeds[j]) / h

    accelerations.append(a)


# To find the distances, all we need to know is the initial position and how it
# Changes. This is enough to implement Euler's Method such that each particular
# Distance is built on the previous one. Since speed is found by comparing 2
# Distances, the lenght of the speed array will be one less than that of the
# Distances.

for v in speeds:
    d = d + h * v

    distances.append(d)


# We know that len(accelerations) = len(speeds) + 1 = len(distances) + 2 = len(time_axis) + 1
# So we will avoid plotting the last speed and last two distances so that we have
# The same amount of data for each line. We can do this using array slices. Here,
# All lines will be plotted onto one graph and have a label that will be displayed
# In the legend.

plt.plot(time_axis[:-1], accelerations, color="blue", label="Acceleration")
plt.plot(time_axis[:-1], speeds[:-1], color="green", label="Speed")
plt.plot(time_axis[:-1], distances[:-2], color="red", label="Distance")

plt.title("Scalar Acceleration, Speed, & Distance Vs. Time")
plt.xlabel("Time (m/s)")
plt.ylabel("a(t), v(t), s(t)")


plt.legend() # Create legend based on passed labels.

plt.grid() # Show grid lines because sometimes your mind plays tricks
           # On you and things that are actually straight look slanted!

plt.show() # We've created all the plots, now show them.
```

Code links in the order above:

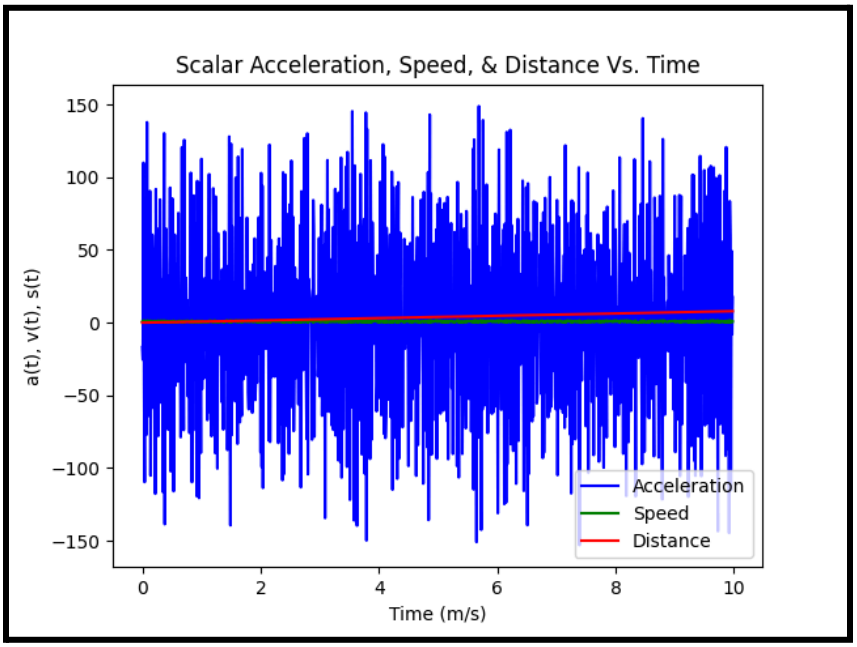
- **https://docs.google.com/document/d/1ftMsYc6vwiGA6Cf0LOvE6vPhDyZTxtYiwPvh2qxfFh0/edit?usp=sharing**
- **https://docs.google.com/document/d/1VUbV7Bk0WZig7V4zxJeqLJxgkRTOfNTRG5YWN5iexoo/edit?usp=sharing**

Python Code Outputs:

For linear speed:



For random speeds:



Step 5: Testing / Evaluating:

What went well	What did not go well	Areas to be researched
- The car moves with control from Bluetooth.	- Car can't turn because the wheels spin inwards/outwards. - Wheels spin very slowly. - Wheels need to be pushed in	-Turning mechanics with the car kit.

	order to spin.	
- Matplotlib works after 3 hours of trying to install it.	- Bluetooth disconnects VERY frequently. - We cannot accelerate the car right now because the GPS app that we are using doesn't have a slider.	-Other Bluetooth apps that might be better? (Currently using "Arduino BlueControl")
- Bluetooth code works smoothly and fast (some codes we ran before had huge delays).	- GPS does not consistently get detected. - GPS soldering damaged the GPS's board.	- How to solder neatly - How to connect the GPS so that it is consistently detected (this problem may be solved by **neat** soldering).
- Python code logically works and has a good design. - Python code was easy to create.	(Not really something that didn't go well but two areas we really need to expand on for code): - Real-time graph updates: we didn't do this since there was nowhere to get the real-time data from (GPS wasn't working). - Different graphs: there's a lot of data that we can portray. While the scalar functions are a good start, the vector functions and 3D plots are something we can definitely add.	-Matplotlib coding -Real-time code using .txt files -Coordinates and how they are organized in NMEA sentences to get the vector functions. (We have no idea how latitude and longitude work). -3D plots in general.
	- Wires come out very easily because the circuitry is quite disorganized.	- Wire Wrapping? We've heard of this but not sure exactly what it is.
- After soldering the GPS's VIN and GND pins worked perfectly so we know how to improve the connection (but it got damaged).	- We kind of didn't follow our mind map at all because of the sheer amount of technical issues that we encountered.	- Battery on back of GPS module: what exactly is its purpose?
	- Connecting RX and TX before starting the program prevents the code from being uploaded.	- How to solve the RX, TX problem on everyone's favorite: StackExchange
	- Design is currently poor because we are focusing on the functionalities.	

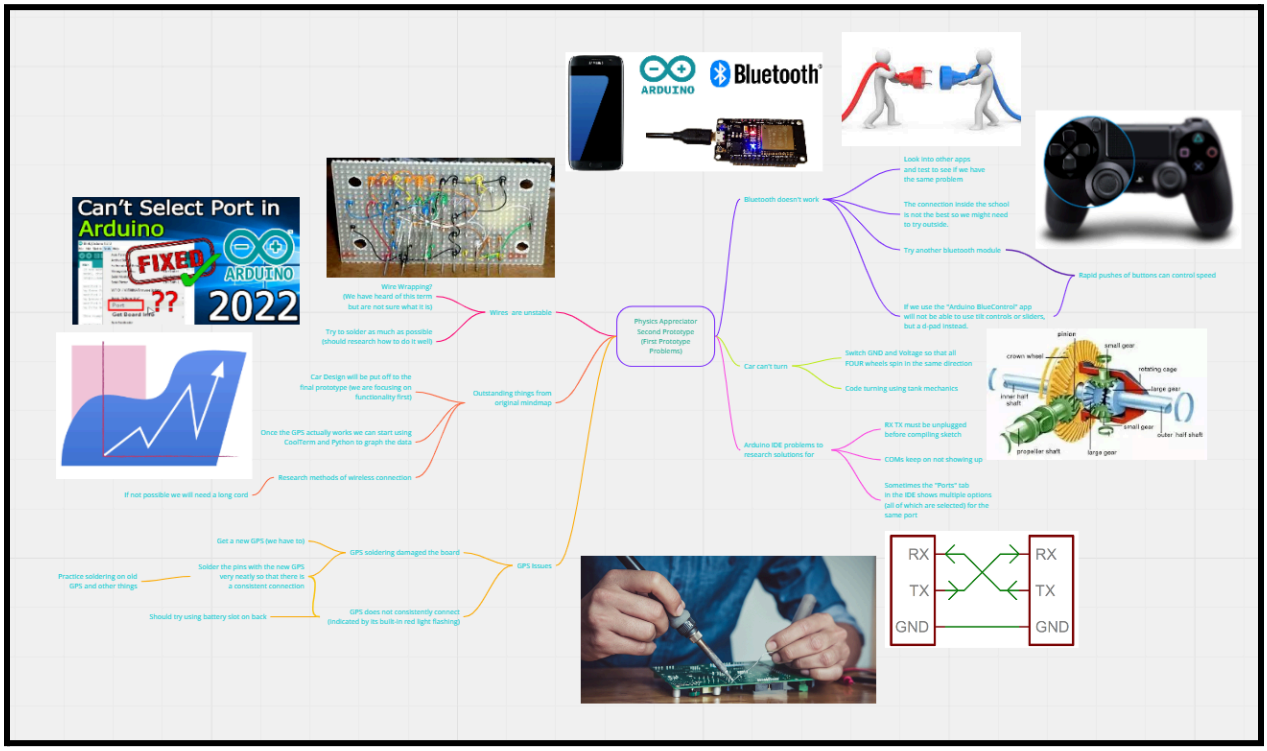
Car moving:

- **https://drive.google.com/file/d/1tBhoVrq97r6xsp70abvcatSVvoWQBKnL/view?usp=sharing**

Cycle #2:

Step 1: Define:

The point of this cycle is to troubleshoot all the problems we encountered before adding any other features.



https://miro.com/app/board/uXjVOrnfOh4=/?share_link_id=919190509547

Plan:

1. Do some planning/documentation.
2. Fix the wiring of the car since it broke and the wheels spin inwards/outwards.
3. Enhance the Arduino code for the Bluetooth control so that the car can turn.
4. Troubleshoot the GPS and the GPS code.
5. Do some more planning/documentation.

Step 2: Research:

Things to research (Blue = Zain, Purple = Jason):

- Arduino Wifi
- Wireless transmitters and receivers (is that a thing?)
- Bluetooth apps
- Arduino IDE problems (RX/TX, Arduino board not showing up in COMMS)
- Soldering neatly
- Wire Wrapping
- Probably some Adafruit GPS coding/troubleshooting since we didn't *really* have the chance to do that in cycle 1.
- How to use matplotlib

Links:

- **https://www.youtube.com/watch?v=Sl5acslCLGI**
- **https://www.youtube.com/watch?v=37mW1i_oEpA**

- **https://www.youtube.com/playlist?list=PL-osiE80TeTvipOqomVEeZ1HRrcEvtZB_**
- **https://forum.arduino.cc/** (for all sorts of problems)
- **https://www.youtube.com/watch?v=D271p2E2_o4**
- **https://www.youtube.com/watch?v=8UIwOQ0S5fk**
- **https://www.youtube.com/watch?v=YYMsS50x1UY**
- **https://docs.arduino.cc/retired/boards/arduino-uno-wifi#arduino-uno-wifi-firmware-updater**
- **https://duino4projects.com/make-gps-transmitter-hc-12-transceiver/**
- **https://fossbytes.com/arduino-remote-control-apps-android/**

Things Established:

- Wire Wrapping isn't exactly something that would be relevant to our project. We should prioritize soldering instead.
- Pins 0 and 1 need to be disconnected while uploading sketches because the Arduino Uno uses those pins for communication with the processor. A way to bypass this annoyance is to just use the SoftwareSerial library. (We imported this library in our code for the GPS before but since it wasn't working, we had to run tests in which we ran empty code and had the connection to pins 1 and 0. From now on we should use the SoftwareSerial library during tests as well).
- The reason why we had problems with COM port connections is because we had to update our drivers (we can just follow the youtube tutorials to do so).
- Arduino wifi is actually built into the Arduino UNO and can be accessed by following the steps on the Arduino doc linked above.
- The HC-12 is a wireless transmitter and receiver that could be very useful for sending back GPS data to a computer. It works up to 1 km in range from its receiver so it would definitely suit our needs. The only problem is if we were to use this component we would need to buy it ourselves. An alternative to using a wireless transmitter in general is just to use a very long USB connector, though a wireless transmitter would allow more range, mobility, et cetera.
- The most common app found for Bluetooth control would be the Arduino Bluetooth Controller. This app includes a controller that resembles a game joystick which is great for steering a car. There are a couple of other Bluetooth apps that include wifi capabilities which are also useful (other apps in link above).
- We should first get used to how to actually solder before we do anything on the GPS. Once we feel comfortable soldering, a good strategy is to place the pins into the breadboard and the GPS so that the GPS does not move. We should also try to use a very minimal amount of solder to prevent pins from touching each other.
- To have the wheels spin the right way (and not inwards/outwards), we can swap the black and red wires for the problematic motors.

Imagine:

- We can have an antenna-type rod to elevate the GPS and Bluetooth modules to help with the connectivity.
    - Use the hot glue gun and metal rods.
- Use a d-pad to control the car's movement.
    - Output for each button can be preset in the settings of the app we are using.

- Car will turn using tank mechanics.
    - Left wheels move back and right wheels move forward to turn left.
    - Right wheels move back and left wheels move forward to turn right.
- We are “imagining” that we will have to get a new Adafruit GPS off amazon for the new cycle in advance. We will re-solder it correctly now that we have some experience.
- Once we get the GPS (it probably won’t arrive this cycle), we can use it to get the data needed to implement Euler’s method for the scalar graphs.
    - Time interval (step size h in python code) can be determined from the rate we set the GPS at (1 Hz, 5 Hz, or 10 Hz).
    - Speed can be parsed from the NMEA sentences.

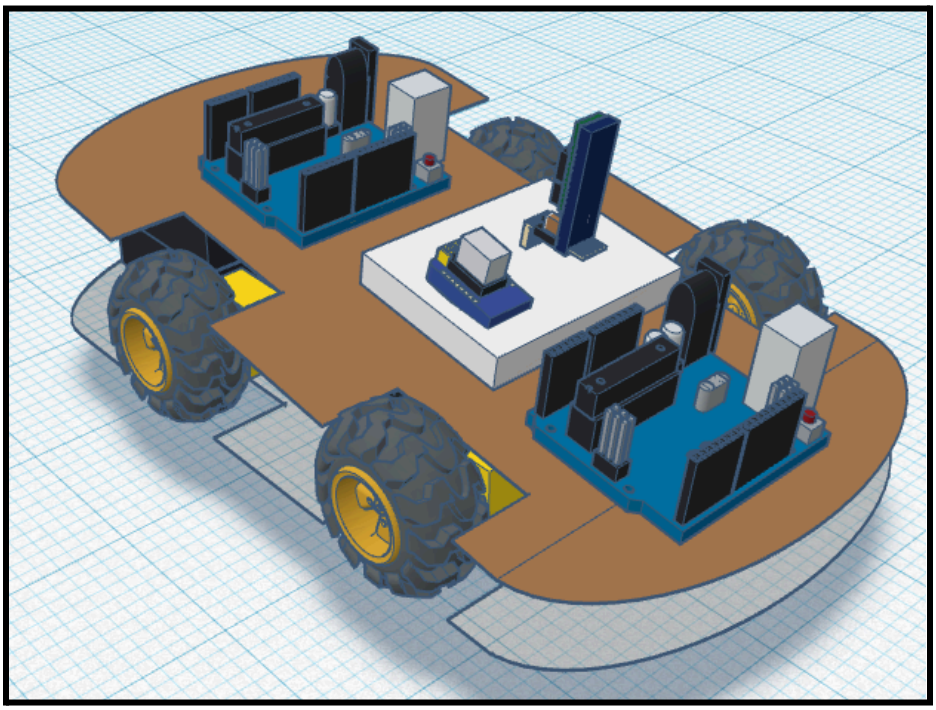
Step 3: Prototype:

Product List:

- Previous prototype items
- Soldering gun
- Wires (need more)
- Extra screws/bolts for antenna
- Hot Glue Gun

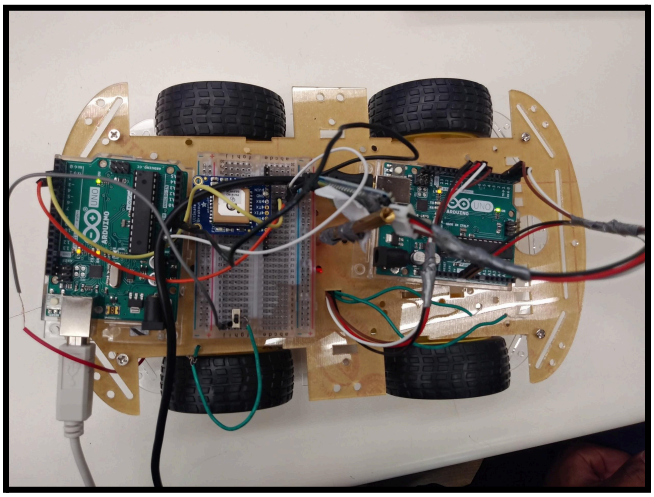
Sketches:

- https://www.tinkercad.com/things/geCSGO4K0II-copy-of-schematic-3-tej301/edit?sharecode=u-kV56b2hSXwG8h9aJpO3nUyhCESJGsGYHeu5hf9qag



(Antenna was planned but not used)

Car Photos:



Step 4: Code:

```
#include <Adafruit_GPS.h>   // Other Adafruit libraries available?
#include <SoftwareSerial.h>

SoftwareSerial GPS_Serial(3, 2); // RX, TX from perspective of Arduino
Adafruit_GPS GPS(&GPS_Serial);   // Create GPS object

String NMEA_Sentence;
char c;

void setup(){
  Serial.begin(115200);
  GPS_Serial.begin(9600);

  GPS.sendCommand("$PGCMD,33,0*6D");              // Turn off annoying antenna update
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);  // We only need the RMC sentence with provides the speed and time
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
}

void loop(){
  NMEA_Sentence = GPSread();
  Serial.println(NMEA_Sentence);
}

String GPSread(){
  // In the future we will have to clear old/bad data
  // But for now we just want to get the GPS working.

  while(!GPS.newNMEAreceived()){
    c=GPS.read();
    }

  String NMEA = GPS.lastNMEA();

  return NMEA;
}
```

```
#include <SoftwareSerial.h>

const uint8_t rxPin = 2;
const uint8_t txPin = 3;

const uint8_t LeftWheelsforward = 8;
const uint8_t LeftWheelsbackward = 9;
const uint8_t RightWheelsforward = 10;
const uint8_t RightWheelsbackward = 11;

char command;

SoftwareSerial BTSerial(rxPin, txPin);

void setup() {
  Serial.begin(9600);
  BTSerial.begin(9600);

  DDRD = 0b00100000;
  DDRB = 0b001111;
}
```

```
void loop() {
  while (BTSerial.available()) {
    command = BTSerial.read();
    Serial.write(command);
    if (command == 'F') {
      digitalWrite(LeftWheelsforward, HIGH);
      digitalWrite(RightWheelsforward, HIGH);
      digitalWrite(LeftWheelsbackward, LOW);
      digitalWrite(RightWheelsbackward, LOW);
    }
    if (command == 'B') {
      digitalWrite(LeftWheelsforward, LOW);
      digitalWrite(RightWheelsforward, LOW);
      digitalWrite(LeftWheelsbackward, HIGH);
      digitalWrite(RightWheelsbackward, HIGH);
    }
    if (command == 'L') {
      digitalWrite(LeftWheelsforward, LOW);
      digitalWrite(RightWheelsforward, HIGH);
      digitalWrite(LeftWheelsbackward, HIGH);
      digitalWrite(RightWheelsbackward, LOW);
    }
    if (command == 'R') {
      digitalWrite(LeftWheelsforward, HIGH);
      digitalWrite(RightWheelsforward, LOW);
      digitalWrite(LeftWheelsbackward, LOW);
      digitalWrite(RightWheelsbackward, HIGH);
    }
  }
}
```

Code links in the order above:

- https://docs.google.com/document/d/1MhmpQmYR4gEtoJh6Cryvxd3j-pRfycQX2MaEXOs8X9A/edit?usp=sharing
- https://docs.google.com/document/d/1eJirTH0Yb2OYzAVGEhNj9vPLPSK6BpFFFseVJnddd Vc/edit?usp=sharing

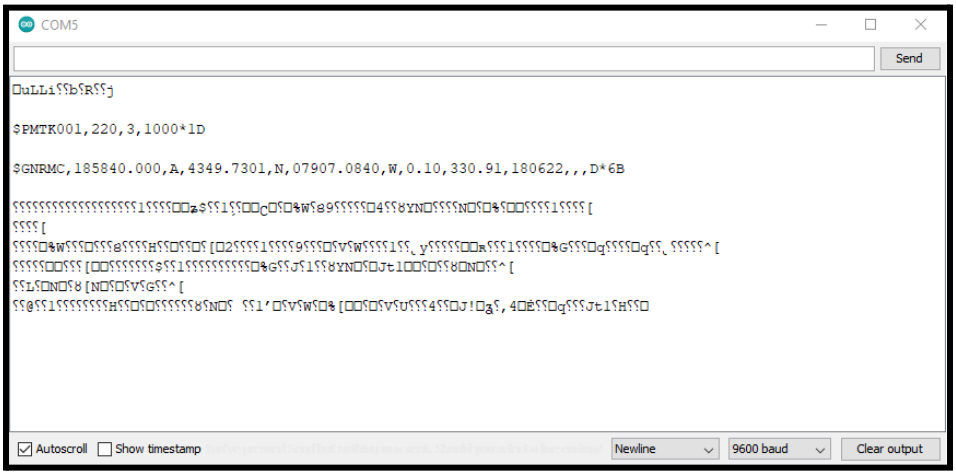
Step 5: Testing / Evaluating:

- Unfortunately, right after having tested the prototype and being about to film it the car stopped working. We think a wire got disconnected meaning that we would have to open up the entire thing again so we just moved on to the next cycle. For that reason, we don't have a video but it's really the same as the first one since the antenna didn't stick. Sorry about that.
- In terms of the COM problem we found that if it is able to work on one computer, don't unplug it. If it doesn't work, just try again later. Since we don't have permission to download the drivers to solve the problem, this is all we can do.

What went well	What did not go well	Areas to be researched
-The car's wheels move exactly like they're supposed to.	-They need to be pushed to spin and don't actually move when placed on a surface (sometimes they move slightly).	-Why don't the wheels spin on the ground? - Lack of power? - Friction issues? -What are the black circles on the motors for? (We just took them off)
-GPS code works exactly like we wanted (even though we were able to complete it because of the bad data we received)	- GPS sends garbage data -99% chance that it's damaged	-Obviously, once we get the new GPS we are going to have to do some research on how it works (coding, coordinates, NMEA sentences; things we did to some extent when it wasn't working)
-The COM problem has been somehow miraculously fixed.		
-RX TX problem has been fixed	-Bluetooth still keeps disconnecting.	-HC-05 Troubleshooting and connectivity issues -We could try doing some tests where we don't have the arduino connected to anything but the bluetooth module. We can try different locations because maybe the school environment is what is causing the bad connection. -This could also be related to the power problem.

-The wires are stable now.	-Wires coming out of Arduino are very disorganized -We keep on finding random tiny pieces of wire coming out of our prototype (kind of concerning?!).	
	- The “antenna” was really foldy since we could not hot glue the metal rods together. We ended up just not using an antenna at all.	

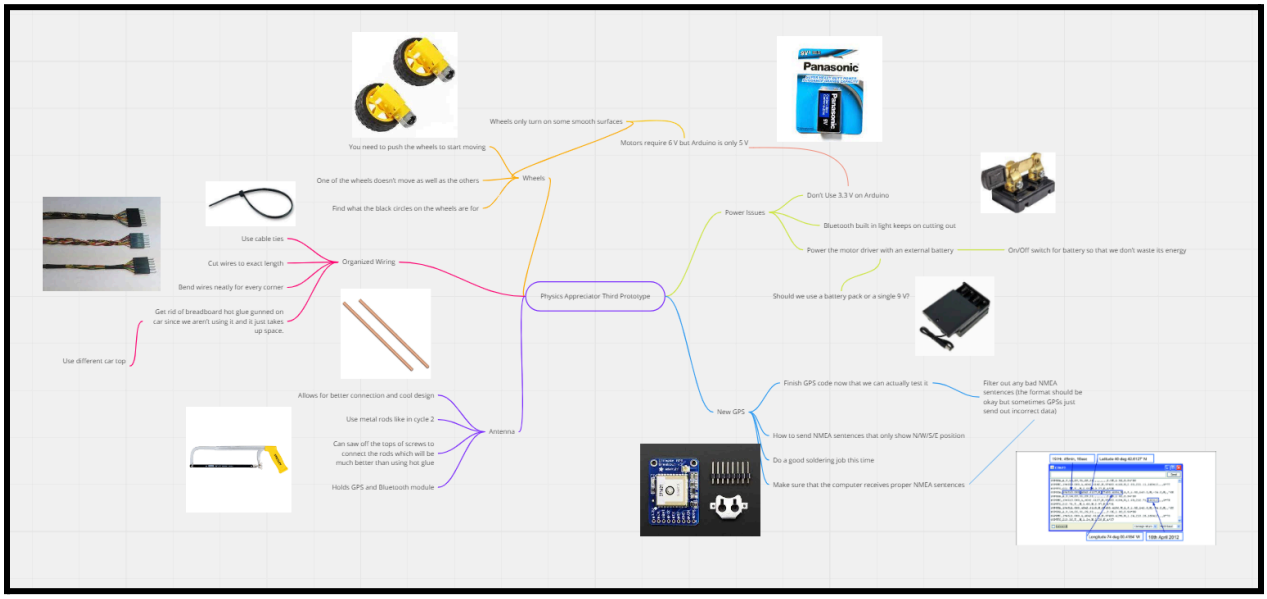
GPS Random Output:



Cycle #3:

Step 1: Define:

The point of this cycle is to continue troubleshooting while trying to upgrade the quality of the prototype in its structure, code, and capabilities.



https://miro.com/app/board/uXjVOrnfiyU=/?share_link_id=840399240882

Plan:

1. Do some documentation.
2. Organize the wiring in the car and add the new battery source.
3. Do some testing using multimeters and motors.
4. Use tape to secure the build and put everything together.
5. Finish off outstanding GPS code, test it, and troubleshoot it if it doesn't work.
6. Build the antenna.
7. Test the prototype.
8. Do some more documentation.

Step 2: Research:

Things to research (Blue = Zain, Purple = Jason):

- Why do the wheels only turn on certain surfaces?
- What are the black circles attached to the motors for?
- How should we parse the GPS code?
- Why is there a read c value for the GPS example codes?
- Why does one of the wheels not move all the time?
- Why do the wheels need to be manually pushed before they start turning?
- Is there a way to have an on/off switch for the battery pack?
- Motor Driver power supply

Links:

- https://forum.arduino.cc/t/robot-wheels-are-not-turning-enough/419469/11
- https://www.quora.com/What-batteries-type-and-voltage-to-use-with-L298N-and-Arduino

- **https://electronics.stackexchange.com/questions/390531/9v-battery-versus-6xaa-batteries**
- The grade 11 physics textbook since we are conveniently learning about motors in our current unit.

Things Established:

- Motors not only need a substantial amount of voltage, they also need a LOT of current. A 9V battery isn't going to cut it since the current will be way too low. This is because of the type of battery (ie. what chemical reaction takes place), which have varying internal resistances. The l298n is said to work on 6V, but it really should get at least 9V. The motor driver should be powered with 6-8 AA batteries connected in series.
- There was no found reason for why there are black circles on the ends of the wheels. We will take them off for now as it seems to interfere with the interior surface of the car. If a purpose/problem is found later on we will consider reattaching them.
- The "read c" in the GPS codes means to read character. Basically, all information the GPS gives us is through reading this character.
- The wheels only turning after a push is likely again related to lack of power, but if this is not the case then maybe something is coming in contact with the motor. We will have to check this out.
- We can use any electrical circuit switch as a on/off switch for the battery pack. The most common option found for this is a Knife Switch, however it is also possible to use a pushbutton. The only problem with using a button is that it may be hard to tell when the circuit is on or off since the button comes back to the same position regardless of how many times you press it.

Imagine:

- Since we now know that motors need huge amounts of current, we can try using a multimeter to measure the current and voltages and see if it's enough.
- We can try using a couple more batteries than needed just in case since some of them will be old and not provide their listed voltage anymore.
- Use a knife switch to turn the batteries on and off.
    - This is very important since we found that if we left the batteries on for even a couple of minutes too long they would become very hot and take a really long time to cool down.
- We can use a different top/plate since we saw some that look cooler than the clear one.

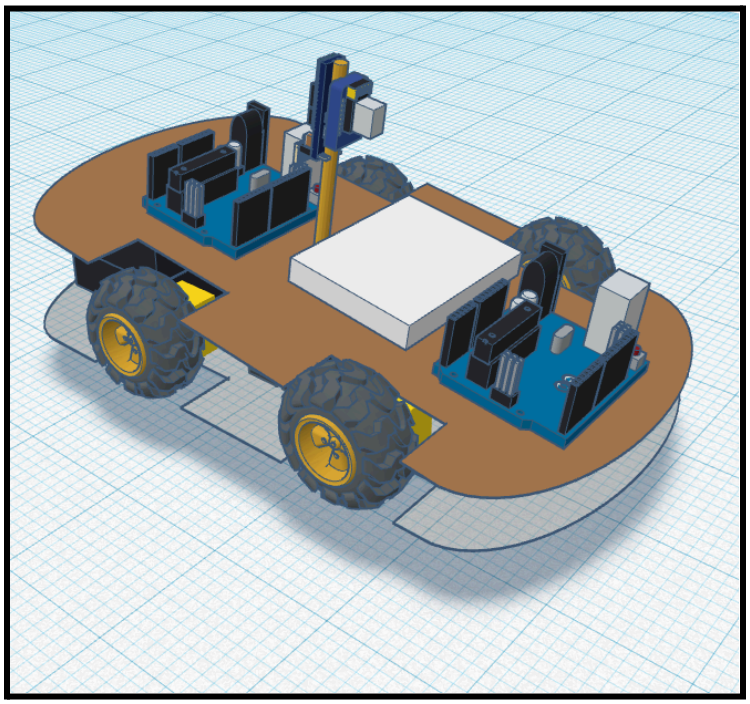
Step 3: Prototype:

Product List:

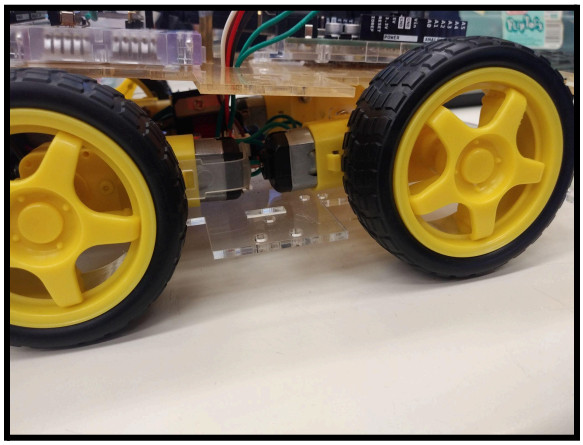
- Previous prototype items
- New Adafruit Ultimate GPS
- AA Batteries
- Battery Pack
- Multimeter
- Switch/Button (whatever's available)
- Extra screws/bolts for antenna

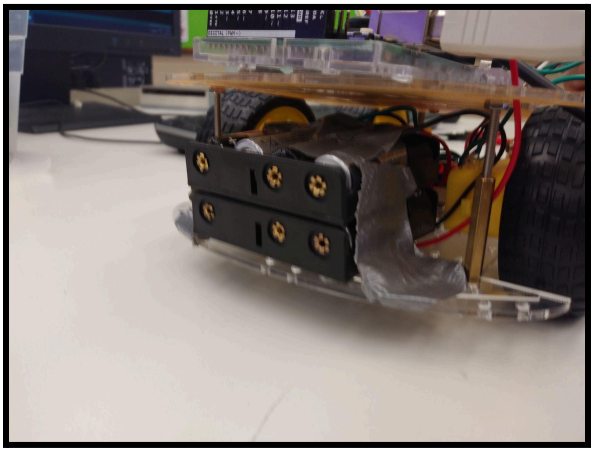
Sketches:

- https://www.tinkercad.com/things/fbLEhx9J040-copy-of-schematic-1-tej301/edit?sharecode=wwHMN9ywQncwHECPJ0vIamKakBWwXG51GgudEBnDyno



Car Photos:

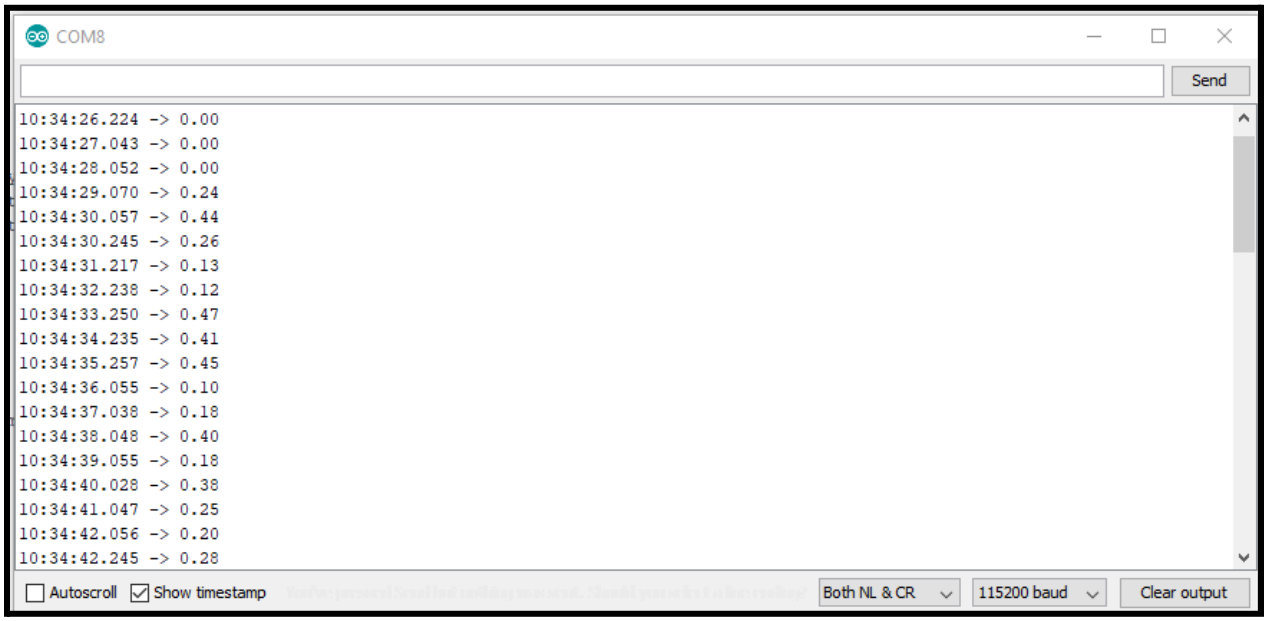




Step 4: Code:

```
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>

SoftwareSerial GPS_Serial(3, 2); // RX, TX from perspective of Arduino
Adafruit_GPS GPS(&GPS_Serial);   // Create GPS object

String NMEA_Sentence;
char c;

void setup(){
  Serial.begin(115200);   // Make sure you set the serial baud rate to 152000
  GPS_Serial.begin(9600); // Or else it will print out random characters!

  GPS.sendCommand("$PGCMD,33,0*6D");             // Turn off annoying antenna update
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY); // We only need the RMC sentence with provides the speed
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);     // Set update rate to 1 Hz (it's good enough for us)

  delay(1000); // The GPS actually needs a second to process the commands
}

void loop(){
  GPSread();

  if (GPS.fix = true){         // If the GPS can get a proper connection,
    Serial.println(GPS.speed);     // Print the speed to the serial for the
  }                                // Python code to graph
}

void GPSread(){
  while(!GPS.newNMEAreceived()){ // While a full NMEA sentence is not available,
    c=GPS.read();                  // Read the partial NMEA sentence's characters
    }

  if (GPS.parse(GPS.lastNMEA())){ // Organize the last read NMEA sentence into callable traits
    return;
  }
}
```

https://docs.google.com/document/d/1PiWN5Jhu3KC65LWxwqv6QwXED8tcYWjjV0hp-_lXT4c/edit?usp=sharing



Step 5: Testing / Evaluating:

What went well	What did not go well	Areas to be researched
-Bluetooth connection is flawless since we dealt with the power problem	-COM problems started just as we were about to record the prototype.	-We have already tried to research this. There isn't much we can do since we don't have

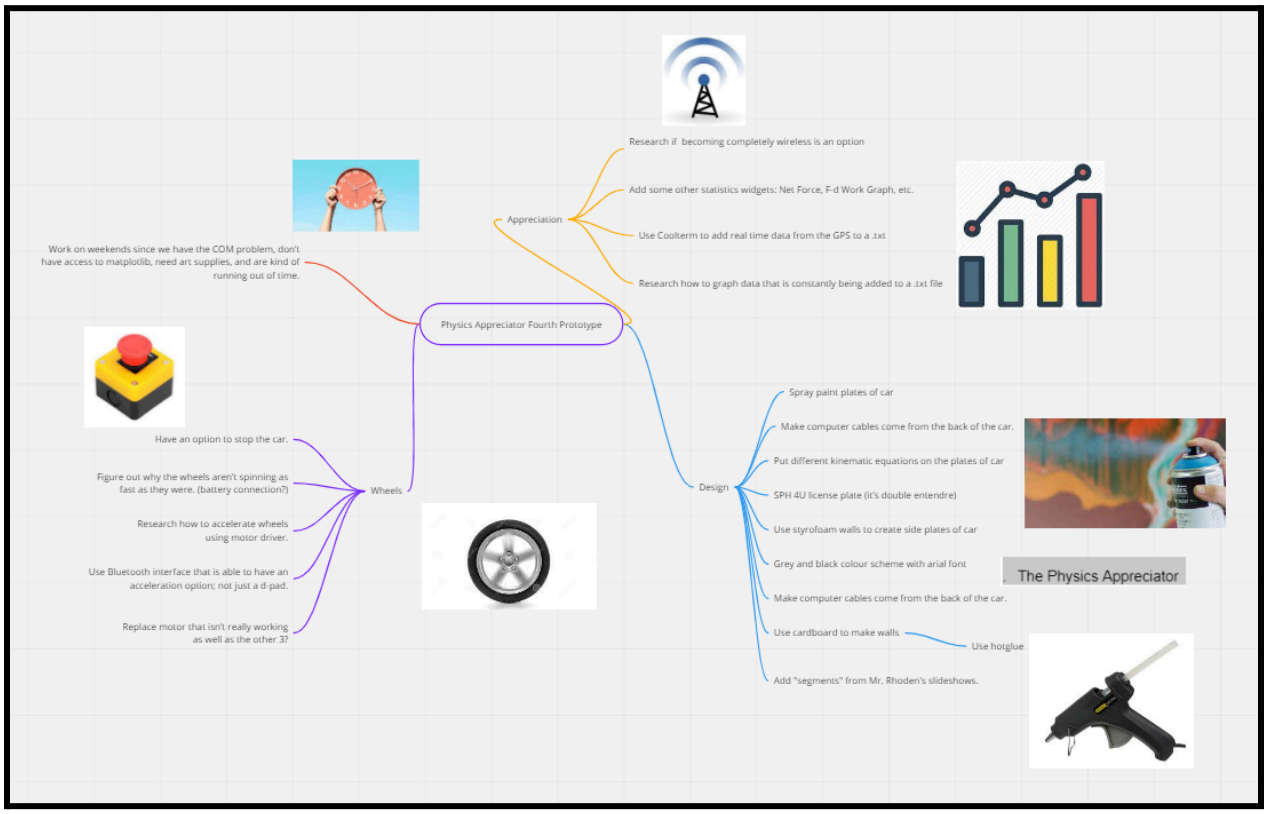
		administrator access.
-The new GPS was soldered properly and we were able to get it running how we wanted it in minutes.	-Design is lacking.	-How we can make side plates for the car and how to paint/colour the car.
-While building the prototype the wheels span SUPER well now that we dealt with the power problem.	-Wheels didn't spin very well during testing -One of them doesn't really spin at all	-Acceleration for the motor driver and wheels.
-New car plates fit nicely together. We now have a good foundation for the final prototype. Wiring is neater.	-The stickiness of the Arduino boards could be better. -The computer cables could be better placed	- What adhesive sheets to use to stick the components together. - How we can neatly connect the two arduino cables to the two computers.
-Switch works conveniently.		
-Car can turn.		
-Antenna is solid. The ends of screws were hacksawed off to allow us to connect multiple metal pieces together.		
-We found out that one of the reasons that the COM problem appears (it came back briefly right when we were trying to take a video of the prototype) is because we try to run a program while we already have one running. It is best to have one sketch open at a time and close the window before running different code.	-Documentation is currently really messy and we are missing certain segments.	-Not much to research, we just have to organize this paper.

Car Moving:
https://drive.google.com/file/d/16HapfwrZ_UmR9_t5HouwKlz-2rke4WID/view?usp=sharing

Cycle #4:

Step 1: Define:

The point of this cycle is to improve on the main aspects of our car which are lacking, now that the tiny nuisances are gone.



https://miro.com/app/board/uXjVOrmuU78=/?share_link_id=77010334241

Plan:

1. Draw a sketch of what the car's new design should look like.
2. Find out how we can accelerate the car.
3. Actually make the design for the car.
4. Code the acceleration.
5. Do some documentation.

Step 2: Research:

Things to research (Blue = Zain, Purple = Jason):

- Really just review the things we already looked at but didn't implement for whatever reason. (For example real-time graphing was never done since there was no real-time data to plot). (Both)
- How to accelerate wheels.
- Art stuff
- Not really research but just review from Mr. Fong's CS class last semester:
    - https://docs.google.com/document/d/1s-KSb0U0W7o6SE_OP18pPCGDjzcddLK9RZVHueJm3xw/edit?usp=sharing

Links:

- https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/
- https://www.youtube.com/playlist?list=PL-osiE80TeTvipOqomVEeZ1HRrcEvtZB_
- https://www.youtube.com/watch?v=Iz_AETY9o5E&t=61s
- A bunch of StackExchange and Youtube videos that didn't help with the Python code problems (see below)

Things Established:

- Acceleration is controlled using the ports to the sides of the direction and on/off ports on the L298N.
- We will have to put off real time graphing for the moment as making Python read off a text file live is a whole other thing.
- Art will be done via spray paint, and cutting and hot glueing cardboard.

Imagine:

- We can colour the car and improve on its design. We are close to finishing the functionalities of the car so we may as well finish what the final sketches of the car should look like.
- The plan is to spray paint car plates to our gray and black color scheme. Then make cardboard cutouts to put on the side and top of the car to hide any wiring.
- We can also print out kinematic equations and put them on the cardboard cutouts, and possibly some "interesting moments" from Mr. Rhoden's slideshow, as this project is inspired by his Kinematics Unit.
- Another idea for the design is to add a sort of sun roof to the top cardboard piece.

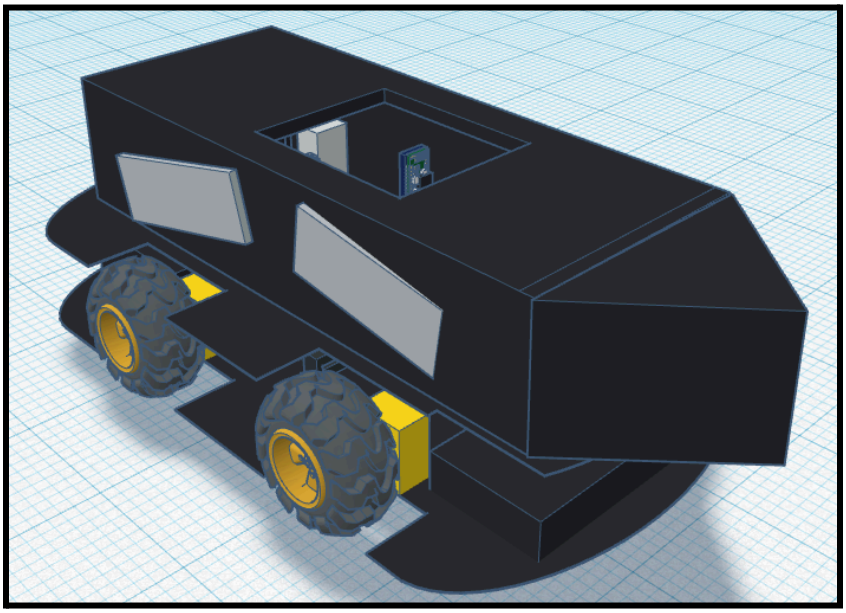
Step 3: Prototype:

Product List:

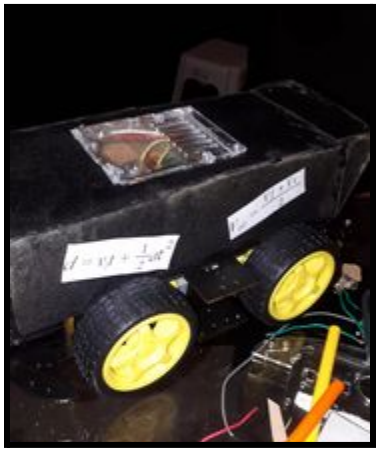
- Previous prototype items
- Black and gray spray paint
- Cardboard
- Scale
- Paper
- Printer

Sketches:

- https://www.tinkercad.com/things/bDnaCCEnUBY-copy-of-schematic-4-5-tej301/edit?sharecode=rh5Pf6We4RZ0_yiJQ4fHU2_1NqTMxB8tN1-OA-M9CN0



Car Photos:



Step 4: Code:

Using Coolterm:

```
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation

open('NMEA.txt', 'w').close() # This is supposed to clear the file
                              # But this gives me an access denied error!

speed_file = open("NMEA.txt", "r") # Open the GPS data as a read only file

time_axis = [0]

accelerations = []
speeds = []
distances = [0]  # Set inital position (reference point) as 0 m

d = 0 # Initial distance (reference point)
h = 1 # Step size or time interval between each outputed speed
         # From GPS which will be recieved through experiment.
         # Letter h is convention for small values; same as dt or dx.


def get_speed():
    new_speed = ""

    while (new_speed == "" or new_speed == "\n"):
        new_speed = speed_file.readline()

    new_speed = float(new_speed) * 0.514444

    speeds.append(new_speed)
    time_axis.append(time_axis[-1] + h)

def get_acc():
    a = (speeds[-1] - speeds[-2]) / h

    accelerations.append(a)

def get_distance():
    d = distances[-1] + h * speeds[-1]

    distances.append(d)
```

```
def plot_graphs(self):
    get_speed()
    get_distance()
    get_acc()

    plt.cla()

    plt.plot(time_axis[:-1], accelerations, color="blue", label="Acceleration")
    plt.plot(time_axis[:-1], speeds[:-1], color="green", label="Speed")
    plt.plot(time_axis[:-1], distances[:-2], color="red", label="Distance")

    plt.title("Scalar Acceleration, Speed, & Distance Vs. Time")
    plt.xlabel("Time (m/s)")
    plt.ylabel("a(t), v(t), s(t)")


    plt.legend() # Create legend based on passed labels.

    plt.grid() # Show grid lines because sometimes your mind plays tricks
               # On you and things that are actually straight look slanted!



get_speed()
time_axis = [0]
get_distance()

# The following line of code creates a graph object by looping through the plot_graphs(self)
# Function every 500 ms. Even though the GPS is set to 1 Hz, it doesn't actually send data
# With a time interval of 1 second on the dot (even though it says it does when you print out
# The time!). For this reason, and also because the code takes time the run, the interval should
# Be set at a time less than 1 s (or 1000 ms).
# plt.gcf() passes in the figure (window) to animate on (gcf = "get current figure").

graph = FuncAnimation(plt.gcf(), plot_graphs, interval=1000)

plt.show() # We've created all the plots, now show them.
```

https://docs.google.com/document/d/1oQx0lZLyJ0GexEjErCMbAYplvrw4qPoPTRet_TMHCRQ/edit?usp=sharing

Using Pyserial:

```
#-------------------------------------------------------------------------------------------------#
                                    ### Libaries ###
#-------------------------------------------------------------------------------------------------#

# All of the below must be downloaded using the command prompt and pip:

import serial                                   # Allow Python to read serial data
from matplotlib import pyplot as plt            # Graphing library
from matplotlib.animation import FuncAnimation  # Allows real-time graphing

# Quick Note: In this program we will calculate "instantaneous" values using other "instantaneous"
#             Values. These values are not actually instantaneous; that would be impossible to
#             Measure. But since the time interval between the speeds that the GPS sends out
#             Is very small, they can be treated as such for a very good approximation.

#-------------------------------------------------------------------------------------------------#
                                    ### Functions ###
#-------------------------------------------------------------------------------------------------#

# This function gets the speed that is being printed by the Arduino code in the serial monitor:

def get_speed():
    while (speed_data_serial.inWaiting() == 0): # While there is no data to receive:
        pass                                        # Do nothing.

    new_speed = speed_data_serial.readline() # After data is available, read it.
    new_speed = float(new_speed)             # Convert coded byte to floating point.

    speeds.append(new_speed)
    print(new_speed)
    time_axis.append(time_axis[-1] + h)      # Record the time that correlates to the speed.


# This function calculates instantaneous acceleration by comparing changing speeds:

def get_acc():
    a = (speeds[-1] - speeds[-2]) / h # From definition of a derivative,
                                      # f'(x) = [f(x+h) - f(x)] / h where a(t) = v'(t).
    accelerations.append(a)


# This function calculates the distance traveled by the car at a point in time by using the
# Change in distance and the initial position:

def get_distance():
    d = distances[-1] + h * speeds[-1] # Derived from Euler's method which can calculate any value
                                       # Of f(x) as long as f'(x) and some point (in this case the
    distances.append(d)                # Previous one) on the line f(x) are known. v(t) = d'(t).
```

```
# This function plots a new graph after getting new data:

def plot_graphs(self): # Real time animated plot object calls itself.
    get_speed()
    get_acc()
    get_distance()

    plt.cla() # Clear the previous graph.

    # The number of times we have stored will be equal to the number of speeds we have. Since we
    # Need to compare two speeds to find acceleration, we will have one less acceleration stored
    # Than speed. Likewise, since speed is the comparison of two distances, we will have one less
    # Speed than distances. To plot the same amount of accelerations, speeds, distances, and times,
    # The following array slices have been used:

    plt.plot(time_axis[:-1], accelerations, color="blue", label="Acceleration")
    plt.plot(time_axis[:-1], speeds[:-1], color="green", label="Speed")
    plt.plot(time_axis[:-1], distances[:-2], color="red", label="Distance")

    plt.title("Scalar Acceleration, Speed, & Distance Vs. Time")
    plt.xlabel("Time (m/s)")
    plt.ylabel("a(t), v(t), s(t)")


    plt.legend() # Create legend based on passed labels.

    plt.grid() # Show grid lines because sometimes your mind plays tricks
               # On you and things that are actually straight look slanted!

#-------------------------------------------------------------------------------------------------#
                                    ### Main Code ###
#-------------------------------------------------------------------------------------------------#

time_axis = [0] # The reason why this is not empty is that the function get_speed() appends to
                # This array using a recursive formula equivalent to
                # (# of iterations already completed) * h. There is no initial value specified,
                # However, since it would be more efficient to do this manually and avoid the
                # Program looping through an if statement that will always evaluate to false
                # After the first iteration. Really, the intial value we put in this array can be
                # Anything: we just need to put something so that the program doesn't break. After
                # The first iteration time_axis = [0] as code further down below.

accelerations = []
speeds = []
distances = [0]  # Set initial position (reference point) as 0 m.

h = 1 # Step size or time interval between each outputed speed from GPS which has been
      # Set in the Arduino code (options of 1 Hz, 5 Hz, 10 Hz; we chose 1 Hz).
      # Letter h is convention for small values; same as dt or dx.


com_num = input("COM: ")                                    # Once the user enters the COM port
speed_data_serial = serial.Serial(f"com{com_num}", 115200) # Python can start reading data and the
                                                            # Rest of the program can begin.

#speed_data_serial.read_all()           # Both of these commands should clear the serial
#speed_data_serial.reset_input_buffer() # Buffer, yet none of them do!!!
```

```
get_speed()     # Get the intial speed.
time_axis = [0] # Set the intial time for that speed.

# After getting one speed manually, the next time we get a speed we can compare them to get an
# Acceleration value. Thus, we can now loop through filling our kinematic arrays. (This would not
# Have been possible had we not received one speed before the looping since there would not be
# Two speeds to calculate the first acceleration value on the first iteration).

# Before beginning looping, we must also manually get another distance value so as to prevent
# The error we would get by the array slice on line 68 since the array would not be long enough:

get_distance()

# The following line of code creates a graph object by looping through the plot_graphs(self)
# Function every 500 ms. Even though the GPS is set to 1 Hz, it doesn't actually send data
# With a time interval of 1 second on the dot (even though it says it does when you print out
# The time!). For this reason, and also because the code takes time the run, the interval should
# Be set at a time less than 1 s (or 1000 ms).
# plt.gcf() passes in the figure to animate on (gcf = "get current figure").

graph = FuncAnimation(plt.gcf(), plot_graphs, interval=500)

plt.show() # We've created all the plots, now show them.
```

https://docs.google.com/document/d/1i5fryHIfjygCP9mHFEYftn_K6KMbpfTP8u04oXc1FNE/edit?usp=sharing

Step 5: Testing / Evaluating:

What went well	What did not go well	Areas to be researched
-Design looks EPIC	I cannot clear my .txt file of old data	Why can't I write to my .txt file ([ERRNO 13]: PERMISSION

		DENIED)
-Bluetooth connects consistently	Pyserial's commands is not clearing the serial buffer.	Why are pyserial's commands not working?
-GPS connects consistently	My scale can't measure very small weights so I cannot calculate force which is mass x acceleration.	
-Besides the wheels not turning, we are pretty much on our final prototype. Obviously small things can be tinkered and improved on but we are happy with what the car is right now.	-Wheels are still not moving properly. -We were not able to do 60% of the other stuff on the mind map because of this. We need to go and fix the wiring.	-Well we finished the design part so there is not anything to research in this area. We just need to open up the prototype and look into the wiring without damaging anything.
-Arduinos have been screwed down so as to not move during motion.		
- Coolterm works no problems and is good for debugging	- It's kind of complicated to run all the programs at once (2 Arduinos, Python, Coolterm) since they need to start in a specific order.	

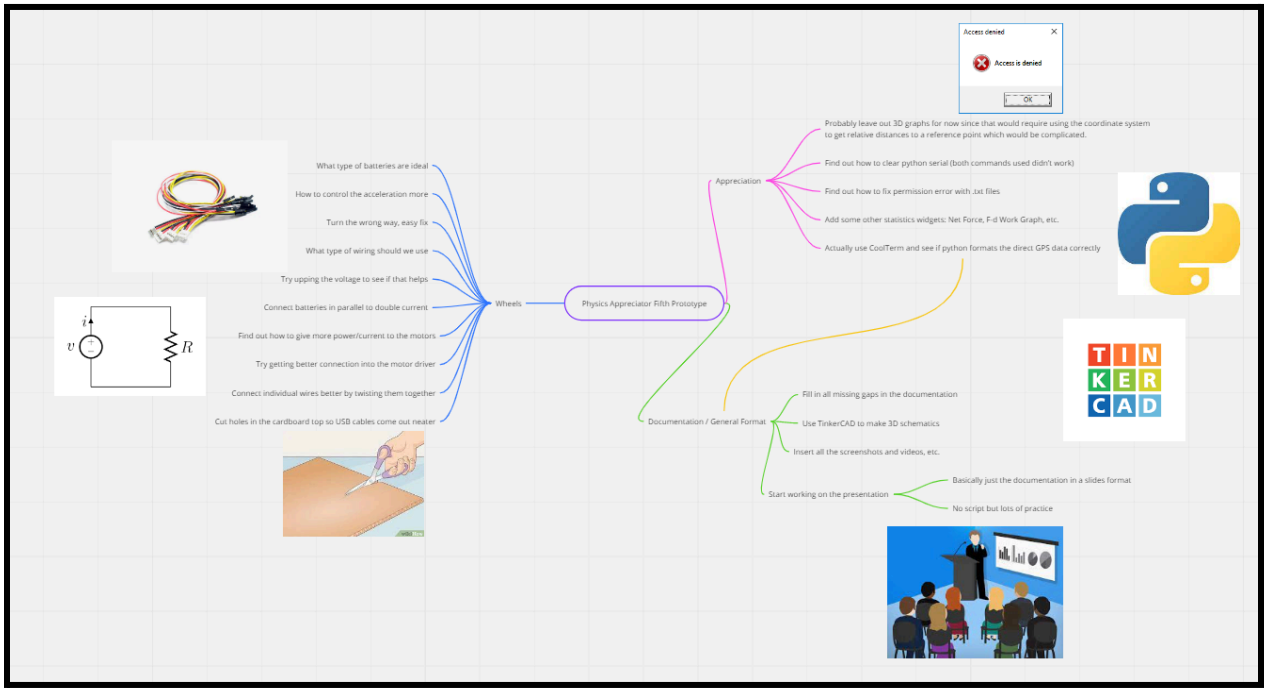
Car Moving:

https://drive.google.com/file/d/1ebfN6OFMoDoiSgphf_PE0MPEO5DABZYR/view?usp=sharing

Cycle #5:

Step 1: Define:

The point of this cycle is to make sure the wheels can actually run. Everything else seems to be working fine but without the car actually moving properly, there is no point for the other components to exist.



https://miro.com/app/board/uXjVOrmqNAE=/?share_link_id=516155403783

Plan:

1. Find out how we can power the motors more efficiently.
2. Fix any python code issues.
3. Fill in some missing parts in the documentation and start thinking about the presentation.

Step 2: Research:

Things to research (This cycle we researched together):

- Parallel and series circuit
- What batteries and amount of batteries to use
- Find out how to fix the python problems

Links:

- https://www.youtube.com/watch?v=dyZolgNOomk&t=123s
- StackExchange
- VARIOUS videos on Youtube about python problems (at least 10)
- The grade 11 physics textbook
- Mr. Rhoden
- https://www.youtube.com/watch?v=hgTgx_h5QOk

Things Established:

- Batteries should actually be connected in series since the motor driver needs a minimum voltage and the batteries already give off 3 times the current needed for the motors!

- Pyserial will by default restart the Arduino (and thus the code) such that there is no reason to use any commands!
- We need to uncover the 2 pins on the motor driver and send an analog voltage to control the speed.

Imagine:

- Really, we just need to wire the car the way it was before (cycle 4)
    - We spent too long thinking of how we can run the car with a parallel circuit instead of sticking to the basics and using a series.
- For the documentation:
    - Put borders around pictures
    - Add in all the screenshots
    - Add links to all the code so it is easy to copy and paste
    - Just organize and format in general
- For the slideshow:
    - Use the documentation as a base and just add more detail into the mental struggle we had to endure just to get a motor to spin.
    - Don't use a script, too long (though we can still practice)
    - Take turns every other slide
    - Use slides from slidesgo portuguese language template
    - Talk about injuries (potentially)
    - Talk about what we would do differently; what we learned.

Step 3: Prototype:

Product List:

- All previous items
- Tweezers
- Multimeter
- Kitchen Scale

Sketches:

- No changes to the structure of the car and thus no blueprints for this cycle.

Car Photo:

- These are more photos from prototype 4, as the design is practically the same.

Step 4: Code:

```
#-------------------------------------------------------------------------------------------------------#
                                        ### Libaries ###
#-------------------------------------------------------------------------------------------------------#

# All of the below must be downloaded using the command prompt and pip:

import serial                                    # Allow Python to read serial data
from matplotlib import pyplot as plt             # Graphing library
from matplotlib.animation import FuncAnimation   # Allows real-time graphing

# Quick Note: In this program we will calculate "instantaneous" values using other "instantaneous"
#             Values. These values are not actually instantaneous; that would be impossible to
#             Measure. But since the time interval between the speeds that the GPS sends out
#             Is very small, they can be treated as such for a very good approximation.

#-------------------------------------------------------------------------------------------------------#
                                        ### Functions ###
#-------------------------------------------------------------------------------------------------------#

# This function gets the speed that is being printed by the Arduino code in the serial monitor:

def get_speed_and_KE():
    speed_data_serial.write('x'.encode())

    while (speed_data_serial.inWaiting() == 0): # While there is no data to receive:
        pass                                        # Do nothing.

    new_speed = speed_data_serial.readline() # After data is available, read it.
    new_speed = float(new_speed)             # Convert coded byte to floating point.

    speeds.append(new_speed)
    time_axis.append(time_axis[-1] + h)      # Record the time that correlates to the speed.

    kinetic_energies.append(0.5 * car_mass * new_speed ** 2)


# This function calculates instantaneous acceleration by comparing changing speeds:

def get_acc_and_force():
    a = (speeds[-1] - speeds[-2]) / h # From definition of a derivative,
                                      # f'(x) = [f(x+h) - f(x)] / h where a(t) = v'(t).
    accelerations.append(a)
    forces.append(car_mass * a) # From Newton's 2nd law of motion


# This function calculates the distance traveled by the car at a point in time by using the
# Change in distance and the initial position:

def get_distance():
    d = distances[-1] + h * speeds[-1] # Derived from Euler's method which can calculate any value
                                       # Of f(x) as long as f'(x) and some point (in this case the
    distances.append(d)                # Previous one) on the line f(x) are known. v(t) = d'(t).
```

```
# This function plots the kinematic functions on the 1st (left side) graph:

def plot_kinematics():
    kinematics_graph.cla() # Clear the previous graph.

    # The number of times we have stored will be equal to the number of speeds we have. Since we
    # Need to compare two speeds to find acceleration, we will have one less acceleration stored
    # Than speed. Likewise, since speed is the comparison of two distances, we will have one less
    # Speed than distances. To plot the same amount of accelerations, speeds, distances, and times,
    # The following array slices have been used:

    kinematics_graph.plot(time_axis[:-1], accelerations, color="blue", label="Acceleration")
    kinematics_graph.plot(time_axis[:-1], speeds[:-1], color="green", label="Speed")
    kinematics_graph.plot(time_axis[:-1], distances[:-2], color="red", label="Distance")

    kinematics_graph.set_title("Scalar Acceleration, Speed, & Distance Vs. Time")
    kinematics_graph.set_xlabel("Time (s)")
    kinematics_graph.set_ylabel("a(t), v(t), s(t)")

    kinematics_graph.legend() # Create legend based on passed labels.

    kinematics_graph.grid()


# This function plots the Force-distance graph on the 2nd (right side) graph:

def plot_Fd_Work():
    Fd_Work_graph.cla() # Clear the previous graph.

    Fd_Work_graph.plot(distances[:-2], forces, color="purple", label="Force")

    Fd_Work_graph.set_title("Force-Distance Work Graph")
    Fd_Work_graph.set_xlabel("Distance (m)")
    Fd_Work_graph.set_ylabel("Force (N)")

    Fd_Work_graph.legend() # Create legend based on passed labels.

    Fd_Work_graph.grid()


# This function plots the Kinetic Energies with respect to time:

def plot_KE():
    KE_graph.cla() # Clear the previous graph.

    KE_graph.plot(time_axis[:-1], kinetic_energies[:-1], color="orange", label="Kinetic Energy")

    KE_graph.set_title("Kinetic Energy Vs. Time")
    KE_graph.set_xlabel("Time(s)")
    KE_graph.set_ylabel("Kinetic Energy (J)")

    KE_graph.legend() # Create legend based on passed labels.

    KE_graph.grid()
```

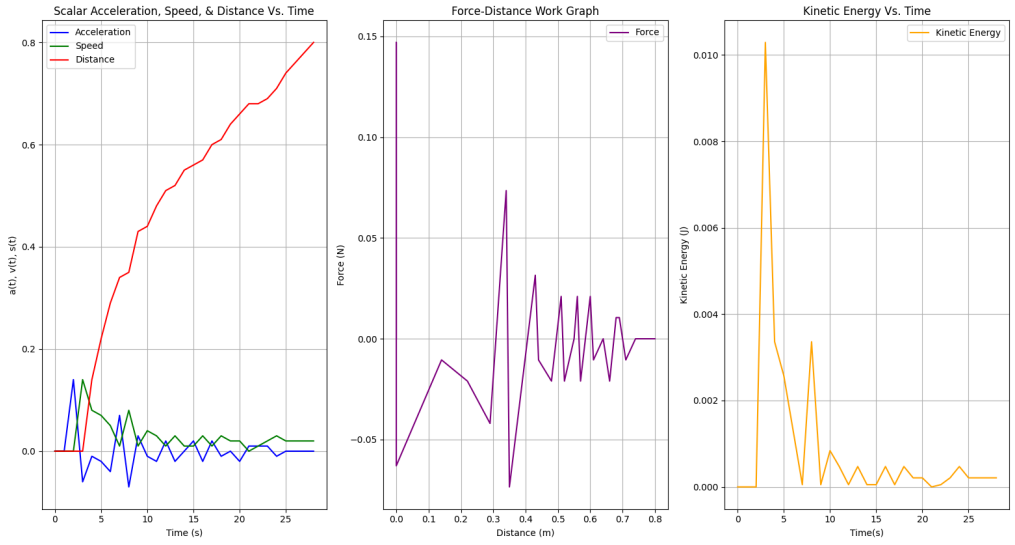
```
# This function plots the new graphs after getting new data:

def plot_graphs(self): # Real time animated plot object calls itself.
    get_speed_and_KE()
    get_acc_and_force()
    get_distance()

    plot_kinematics()
    plot_Fd_Work()
    plot_KE()

#-------------------------------------------------------------------------------------------------------#
                                        ### Main Code ###
#-------------------------------------------------------------------------------------------------------#

car_mass = 1.05 # Car was scaled to be this many kilograms

time_axis = [0] # The reason why this is not empty is that the function get_speed() appends to
                # This array using a recursive formula equivalent to
                # (# of iterations already completed) * h. There is no initial value specified,
                # However, since it would be more efficient to do this manually and avoid the
                # Program looping through an if statement that will always evaluate to false
                # After the first iteration. Really, the intial value we put in this array can be
                # Anything: we just need to put something so that the program doesn't break. After
                # The first iteration time_axis = [0] as code further down below.

accelerations = []
speeds = []
distances = [0]  # Set initial position (reference point) as 0 m.

forces = []
kinetic_energies = []

h = 1 # Step size or time interval between each outputed speed from GPS which has been
      # Set in the Arduino code (options of 1 Hz, 5 Hz, 10 Hz; we chose 1 Hz).
      # Letter h is convention for small values; same as dt or dx.


com_num = input("COM: ")                                        # Once the user enters the COM port
speed_data_serial = serial.Serial(f"com{com_num}", 115200) # Python can start reading data and the
                                                                # Rest of the program can begin.

window, (kinematics_graph, Fd_Work_graph, KE_graph) = plt.subplots(nrows=1, ncols=3)

get_speed_and_KE()     # Get the intial speed.
time_axis = [0] # Set the intial time for that speed.

# After getting one speed manually, the next time we get a speed we can compare them to get an
# Acceleration value. Thus, we can now loop through filling our kinematic arrays. (This would not
# Have been possible had we not received one speed before the looping since there would not be
# Two speeds to calculate the first acceleration value on the first iteration).

# Before beginning looping, we must also manually get another distance value so as to prevent
# The error we would get by the array slice on line 68 since the array would not be long enough:

get_distance()
```

https://docs.google.com/document/d/18NW1odWNBz6mOlL7RrKyBhW2f0LM5zs9mEipy-cgPAY/edit?usp=sharing

Step 5: Testing / Evaluating:

What went well	What did not go well	Areas to be researched (More like things to be considered at this stage)

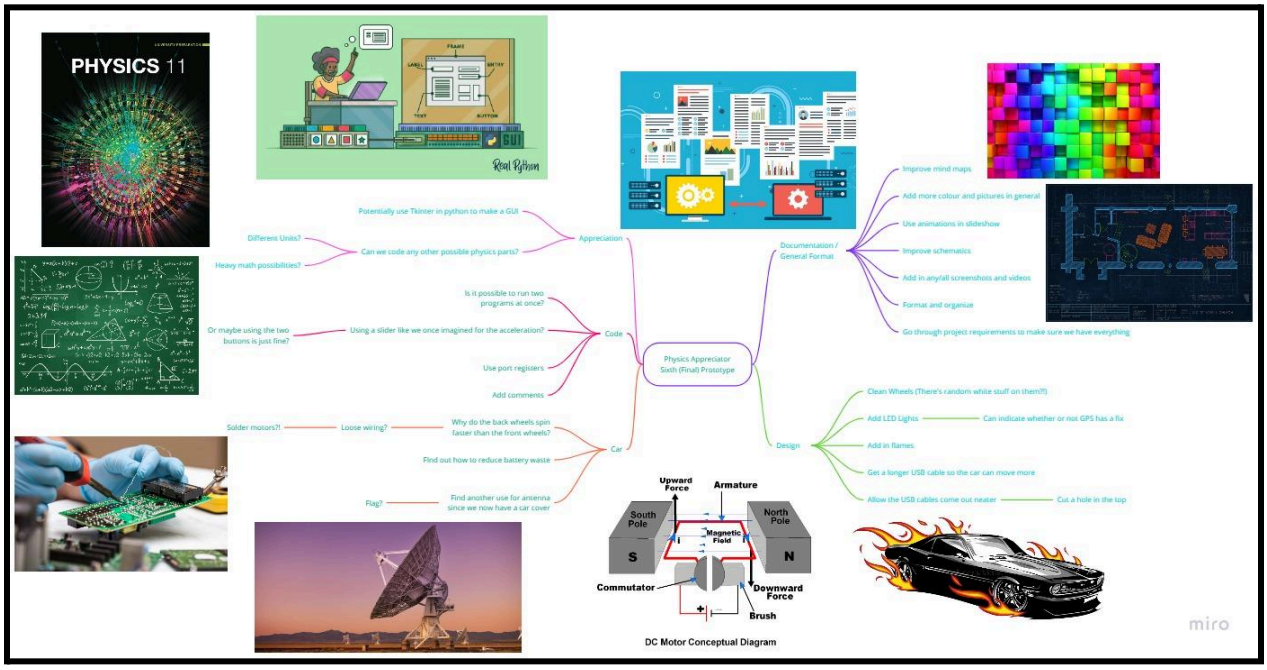
- Circuit works well and can actually get the car moving well	- You need to disconnect the GND from the Arduino connected to the battery after you finish using the car since the battery will actually power the Arduino and waste its juice.	
- Found out that pyserial resets the Arduino so you don't have to clear the serial buffer!	- USB cables do not fit through the cardboard top, which forces the top upwards	- How we can fit the cables out of the top of the car neatly
- Graphs look really good - Graphs are accurate	- USB cables are not very long so we can't really move the car more than ~2 metres	- Since we can't use wireless connection, what type of cables would be the best to use
- GPS is accurate	- Computers shut off while trying to videotape the prototype: SO ANNOYING. -Waste a lot of time with switching back to series connection of batteries.	- What we should do for the additional physics analysis. - Should we stick with the physics we currently have or try to add to it.
	- Metal connections on motors keep breaking!	- How we can securely connect wires to the motor



Cycle #6:

Step 1: Define:

Since all the components are now all working, this cycle is mainly to tidy things up.



https://miro.com/app/board/uXjVOrmpWvk=/?share_link_id=485739510398

Plan:

1. Find a way to get the USB cables to come out nicer as well as extend them.
2. Add some final design to the car.
3. Finish all documentation and the presentation.

Step 2: Research:

Things to research (This cycle we researched together):

- Why does the car make a detonating-type beeping noise after reducing the speed of the wheels?
- Can you run two Arduino sketches at the same time?
- Mind Map Websites (used Google Drawings before)

Links:

- https://arduino.stackexchange.com/questions/67505/l298n-beeps-and-malfunctions-when-receiving-certain-analogwrite-values#:~:text=So%2C%20what%20is%20the%20motor,heard%20as%20a%20continuous%20beep.
- https://www.digikey.ca/en/maker/projects/multi-tasking-the-arduino-part-1/b23d9e65c4d342389d20cbd542c46a28#:~:text=The%20Arduino%20is%20a%20very,load%20and%20run%20multiple%20programs.
- https://venngage.com/features/mind-map-maker

Things established:

- When the motor does not have enough energy to overcome the friction, it will create vibrations in the motor, which causes a PWM signal which can be heard as a beep.

- Arduino cannot run multiple programs at once.

Imagine:

- We create RGB headlights and use the antenna as a flag.
- Comment Central for the code.
- Soldering (!?) will ensure that we never break a motor.
- Clean design, clean wheels; that's just to be expected.
- Wow factor to the documentation.
    - We can add more colour to the documentation.

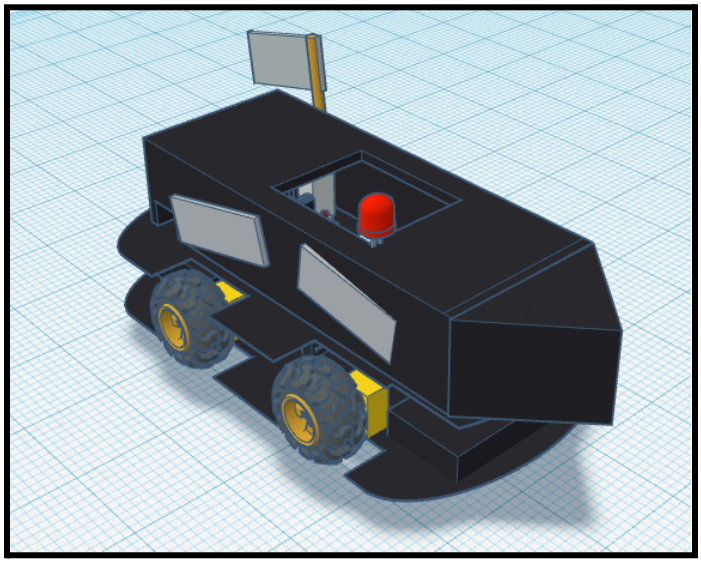
Step 3: Prototype:

Product list:

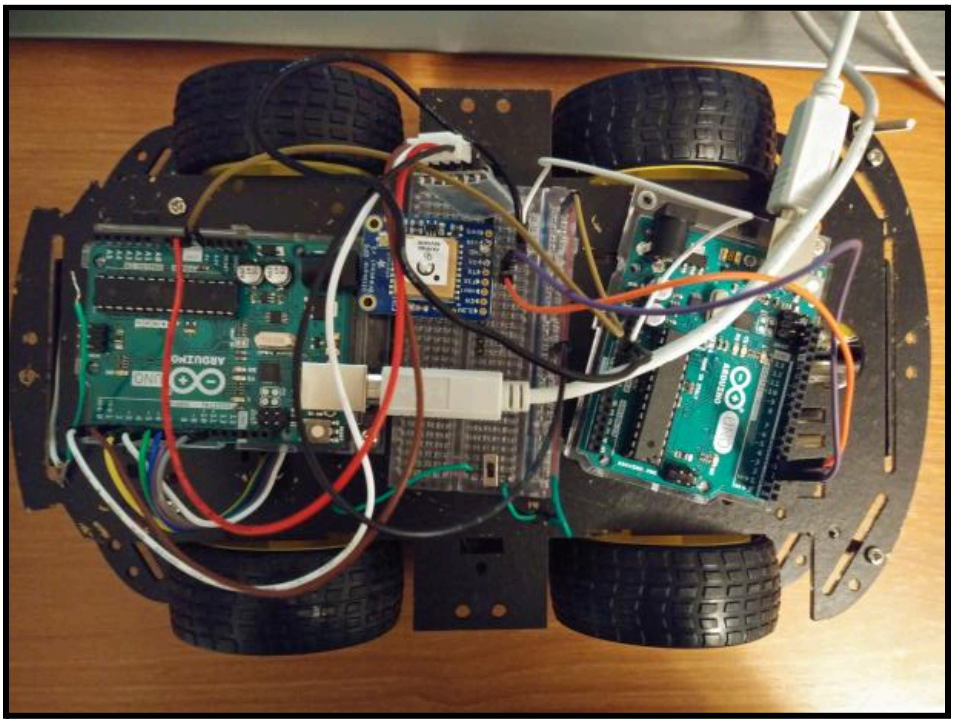
- Previous prototype items
- Extension cord for the USB cables?
- 2 LEDS

Sketches:

- https://www.tinkercad.com/things/ftgGguD4mKX-copy-of-copy-of-schematic-4-5-tej301/edit?sharecode=iyr5eKXTDi2OTjMgtTQc7asRX8rXYBeRgWcFHX_40MI



Car Photo:



Step 4: Code:

```
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>

SoftwareSerial GPS_Serial(3, 2); // RX, TX from perspective of Arduino
Adafruit_GPS GPS(&GPS_Serial);   // Create GPS object

String NMEA_Sentence;
char c;

void setup(){
  Serial.begin(115200);   // Make sure you set the serial baud rate to 152000
  GPS_Serial.begin(9600); // Or else it will print out random characters!

  DDRB = 0b000001;

  GPS.sendCommand("$PGCMD,33,0*6D");              // Turn off annoying antenna update
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);  // We only need the RMC sentence with provides the speed
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);      // Set update rate to 1 Hz (it's good enough for us)

  delay(1000); // The GPS actually needs a second to process the commands
}

void loop(){
  GPSread();

  if (GPS.fix = true){          // If the GPS can get a proper connection,
    Serial.println(GPS.speed);     // Print the speed to the serial for the Python code to graph

    PORTB = 0b000001;           // Turn LED light on
  }
  else{
    PORTB = 0;                  // Light will blink since anytime the GPS is not sending data to
  }                             // The Arduino, this line will run. If the GPS has no fix for an
}                               // Extended period of time, it will remain off.
```

```
void GPSread(){
  while(!GPS.newNMEAreceived()){ // While a full NMEA sentence is not available,
    c=GPS.read();                   // Read the partial NMEA sentence's characters
    }

  if (GPS.parse(GPS.lastNMEA())){ // Organize the last read NMEA sentence into callable traits
    return;
  }
}
```

https://docs.google.com/document/d/1uuSWRgVueNq_fddvxpIPyKdsAVvz5GLWFU1tE0e0t1M/edit?usp=sharing

Step 5: Testing / Evaluating:

What went well	Room for Improvement	Areas to be researched
- Accomplished basically everything from original mind	- Going completely wireless (though we don't know how we	- There is nothing to research as we have practically completed

map	would do this)	our original plan.
- Python code works splendidly - GPS code works precisely -Bluetooth code works amazingly -Interface is fine and easy to use on phone	- Use better motors	
- Developed our skills: Soldering, multimeters, presentations	- Stronger wires (green ones look so strong but snap if you whisper too loudly near them)	
- Car works as how we first planned	- Do vector graphs - 3D graphs	

Car Moving:
https://drive.google.com/file/d/1Y7ZPaYs2kkBjaCmXO9PDWr98_hquoOaP/view?usp=sharing

Real-Time Graphing:
https://drive.google.com/file/d/1VUZ00gpvI7ttpzTeiB4psKmjG6lDFEfT/view?usp=sharing