Senior Design

Control System
Hyperloop Pod Project

New York Institute of Technology

Image extracted from Railway Gazette

Genecis Fernandez (gferna04@nyit.edu), Dept. of Electrical and Computer Engineering, SoECS, NYIT

Chelsey Toribio (ctorib01@nyit.edu), Dept. of Electrical and Computer Engineering, SoECS, NYIT

Mayameen Mushairib(mmushair@nyit.edu), Dept. of Biomedical Engineering, SoECS, NYIT

Nicolas Picon Lopez (npiconlo@nyit.edu), Dept. of Electrical and Computer Engineering, SoECS, NYIT

Introduction:

The world is in one of the most fascinating moments in the age of technology. High tech innovation is happening in different sectors with Hyperloop technology being the latest innovation in the transportation sector. The Hyperloop industry is advancing more than ever before as different engineers and developers have joined the race to come up with the most complete and advanced prototype to form the basis for future development and production[1,5]. Hyperloop is the concept of a way of transportation that goes at really high speed and it levitates from the ground in a sealed pressurized tube, negating the friction force and opening new opportunities to reach even higher speeds. Currently, Elon Musk's Hyperloop plans have been met with a significant dose of skepticism by the scientific community and the media . The Evacuated-Tube Rail (ETR) team started working on this project and developed their design of a Hyperloop pod, however, their design is missing some key components to be able to be a fully working prototype. The current race from various groups of developers is to improve the notable kinks in the existing pod's prototypes. For our senior design project, we will follow up and work with the previous ETR team that has been working on their Project to build a Hyperloop tube pod system [4]. We plan to solve this before graduating, to leave a working prototype of a hyperloop pod to our school. With this prototype, we plan to start an organization at our school to encourage undergraduate and graduate students with the help of some professors to create and develop a bigger design to compete in the SpaceX Hyperloop competition of 2022.

Problem Statement:

The problem is that the design of the ETR team has no control system to make the prototype autonomous while also being able to make decisions while operating. One of the key elements that the SpaceX competition judges look at is the control system and how quick it is able to make a decision, for example, braking in case an emergency happens. As well, there is a need for telemetry equipment to send data back to a main station, that could be a computer, to supervise if everything is running well.

Background:

In the last few years, different parties have taken part in the SpaceX Hyperloop pod competition. Among the key competitors include the MIT team, Althen, Hyperlynx team, and AZLoop team. The competition rests on the design and development of the system of the pod based on Elon Musk's Hyperloop idea. These players have made significant development sensors and the entire control system. MIT has developed more than 35 sensors that serve various functions including telemetric sensing, data collection, and active control of the pod [1]. Among the 35 sensors, there are sensors for collecting pod's position, velocity, roll, pitch, temperature, power, height, acceleration, yaw, and ride height information. These sensors are controlled by five different microcontrollers. Each microcontroller is responsible for different groups of sensors and actuators. Further, MIT sensors are both analog and digital output sensors that collect and send data to a master computer for health monitoring and piloting. Analog sensors 4-20mA standard, an industry-standard used since the 1950s [1]. In digital analog, data is digitally converted and transmitted. On the other hand, analog sensors produce a continuous signal that is proportional to the quantity being measured. Digital analog produces a discrete output signal, digital representation of quantity being measured, and produces binary output in the form of ones and zeros [3].

Althen laser sensors for the Hyperloop include both high-speed laser sensors, draw wire sensors and acceleration sensors [3]. The high-speed laser sensors are used to measure the pod's position, relative to the maglev track over which the pods hovers. They also used for the validation of the vehicle model, measurement of the lateral position of the chassis, and tuning of the suspension. The draw-wire sensors are used for the position and control of the brake system. HyperLynx pod has a control system with sensors that collect information on velocity, acceleration, altitude, pressure, speed, temperature, and power consumption. The custom software code is developed to execute systems commands based on sensor input data. The power control sensors have a power input of 250 watts. The AZ Loop final design captures RPM, temperature, force, proximity, laser and optical, pressure, and flow sensors. These sensors facilitate pod control through data gathering and communication. Sensors are connected to the data logging and

communication controller that communicate relay information to the black box and telemetry stream.

Solution:

In order to get the hyperloop prototype working, there is a need for a control system. We would have different sensors to give feedback for different variables to our control system. We would have a temperature sensor; its task is to monitor the pod temperature while on the track. The temperature sensor would reassure the precise surface temperature for the different components of the pod. It is essential for the controller to not be exposed to a hot surround's environment. There would also be a pressure sensor that will be in charge of measuring the pressure inside the tube and check how much air is inside the pod, to calculate how much air resistance the pod will be experiencing. As well, with this sensor we will be monitoring if there is an air leak in any section of the track, so we can fix it. Another sensor that will be used is the infrared range sensor. Infrared range sensors will be used to measure the hyperloop pod gap between the track and the pod. We can also use it as a measurement of the lateral position of the motor vehicle's base frame in new designs. There will also be an inertial measurement unit(IMU). The inertial measurement unit includes an accelerometer, gyroscope, magnetometer, and a barometer. The accelerometer will measure linear acceleration, and the gyroscope helps indicate the orientation of the pod with respect to the earth. The magnetometer will be used to measure the magnetic field's strength. The barometer will be our backup if our pressure sensor fails. As well, we would use a raspberry pi to control all the sensors. We will code and program the raspberry pi to receive feedback from the sensors and report back the pod's status; if there are any issues or leaks. With this information, the raspberry pi should be able to take decisions on-flight and decide if it is needed to decelerate or brake. Also, it will be in charge of controlling the speed of each of the four motors and with this, its acceleration and the position of the pod.

System Architecture:

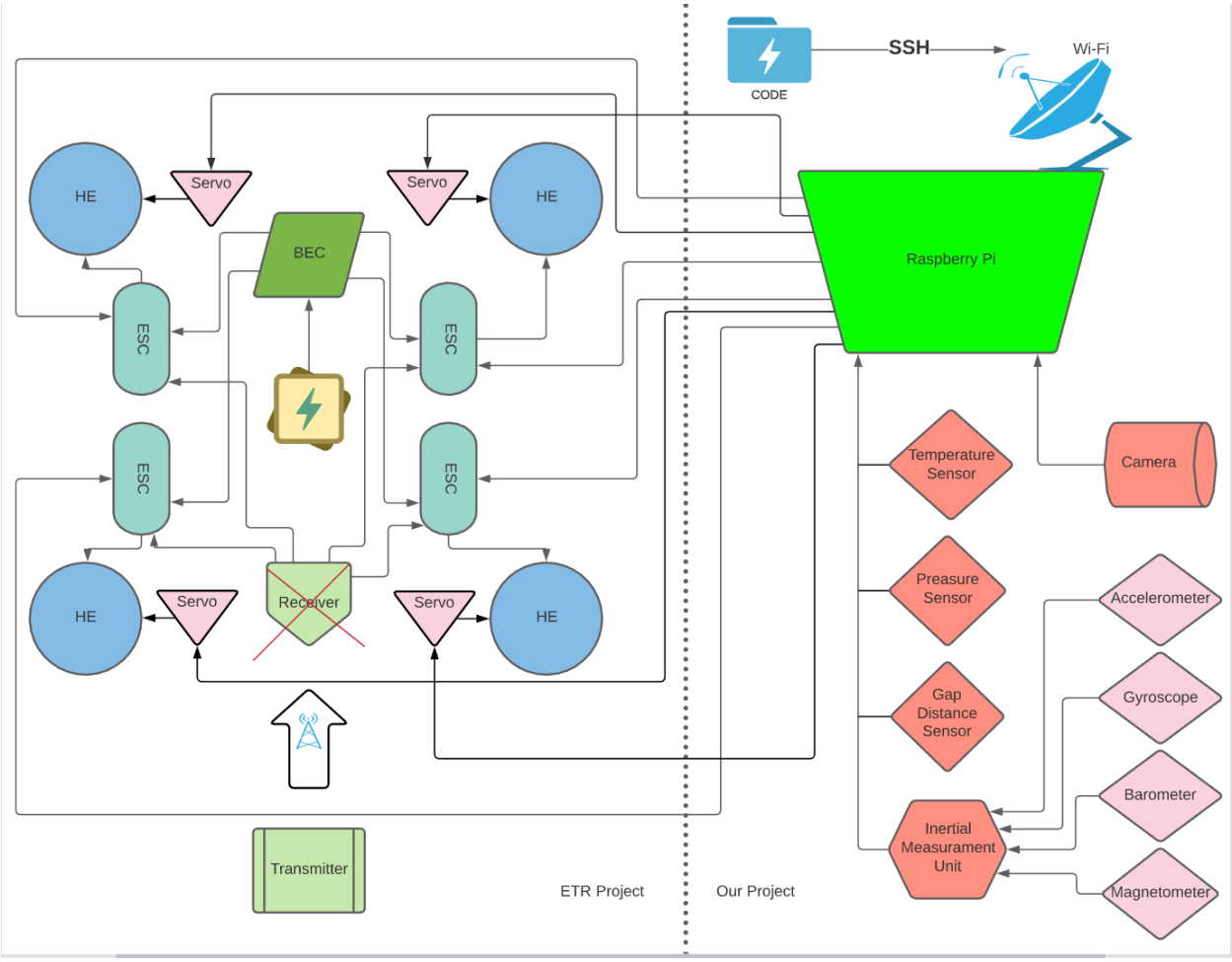


Figure 1 - System Architecture Block Diagram

The figure above shows the system architecture of our project. The figure is divided by a dotted line in the middle that its intention is to separate the ETR Project system architecture from our system architecture. The left half side is the ETR project system architecture for the pod, which consists of the 4 Hoover Engines (HE) that are the ones in charge of producing the electromagnetic field that will make levitate the pod, The Electric Speed Controllers (ESC) are in charge of controlling the RPMs of the HE and are powered by the Battery Emitter Controller (BEC) which is in charge of distribute the energy of the battery to the 4 ESC. The battery is in the middle of the pod and it is indicated by a lighting and right below it is the receiver that is used to receive the signal of the transmitter. Finally, the last part of their project are the servos

which are going to be used to tilt the hoover motors in a certain configuration to produce a forward, backwards or side motion.

In the right half of the image is our project which is going to be centered in a Raspberry Pi that is going to be in charge of receiving data and controlling the pod based on that data. For that first we will need to configure the Raspberry Pi to be able to access it through wifi with an SSH connection and then we will upload our code in it. The raspberry pi will have a lot of input sensors that are going to help the raspberry pi control the pod depending on the command of the user. These sensors are going to be: a temperature sensor, a pressure sensor, a gap distance sensor, a camera and an Inertial Measurement Unit (IMU). The IMU includes 4 different sensors: The accelerometer, the gyroscope, a barometer, and a magnetometer. The raspberry pi will replace the receiver and transmitter of the ETR project and will be in charge of controlling the servos to change to the desired motion configuration of the user and the ESC to be able to keep the pod levitating.

The motion configurations are the following:

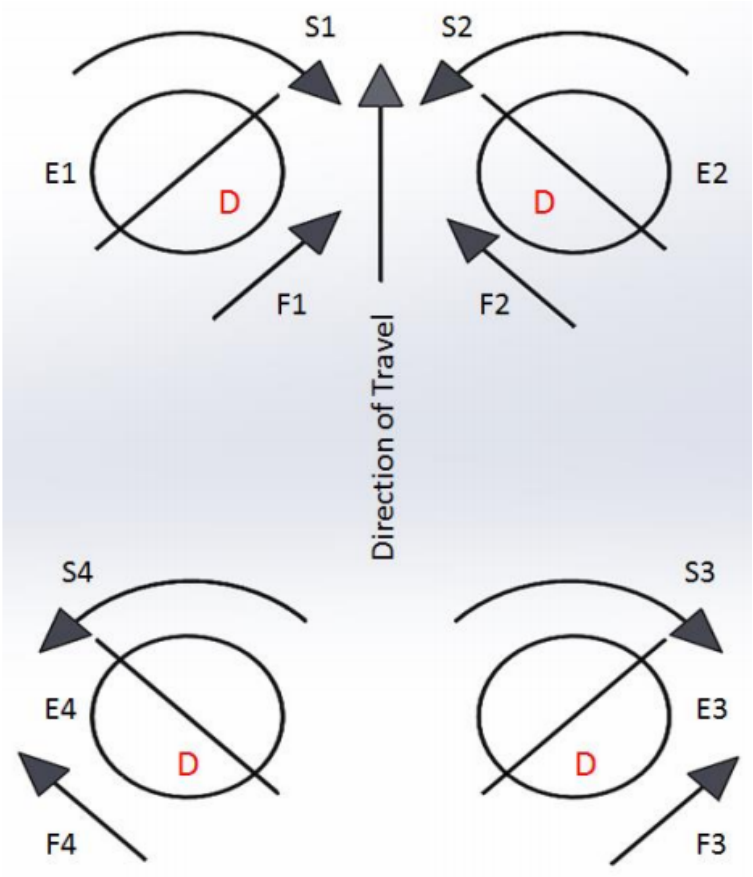


Figure 2.1 - Net Forward Motion Configuration

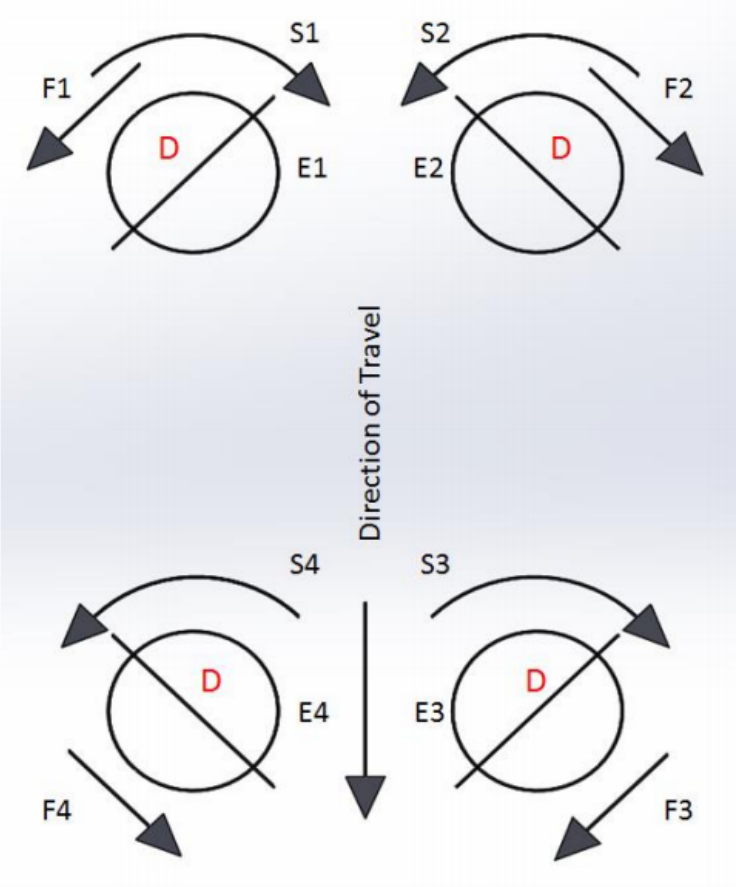


Figure 2.2 - Net Backward Motion Configuration

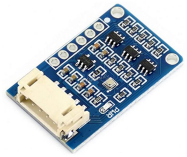
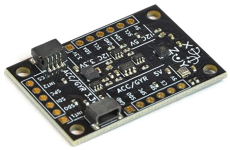
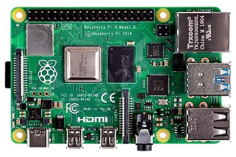
The 4 circles in figure 2.1 and 2.2 are the hoover motors which are going to be mounted in a base connected to the servos and this base is going to be tilted 45 degrees with a rotation possibility to the right and left. The servos are in charge of tilting simultaneously the bases with the hoover engines in the direction where the “D” in red is in the configuration figures on top. The neutral configuration will be the hoover engines not being tilted at all and be parallel to the track.

Code Flowchart:

Figure 3 - Code Flowchart Block Diagram

The figure above shows the code flowchart of our project. First, we would power it on, then turn on the hoover engine. After the hoover engines are one, we would calibrate the sensor. To calibrate the sensors, we would have to compare the measurements with our standards and get an offset. After we calibrate the sensors, there would be two inputs, the user input, and the sensor input. There would be four user inputs options. One of the user inputs is to accelerate. For the accelerate option, it would tilt servos for forward motion configuration. Another user input is decelerate. For the decelerate option, it would tilt servos to backward motion configuration; it would stop when velocity reaches zero. Another user input is the brake; to break, it will start decelerating, it will tilt servos to neutral motion configuration. The last user input is Turn Off; we will use the code to break to stop the Hoover Engine. There would be five sensor inputs. The first is the temperature input; it would monitor the sensor. While monitoring the sensor, it would make sure the temperature is not too high; if it is too high, it will turn off. The premature input would as well send data to the user. Another sensor input is pressure input. The pressure input would monitor the sensor; it would ensure the pressure is not out of range. If it is out of range, then it will warn the user. The pressure input would also send data to the user. Another sensor input would be the inertial measurement unit (IMU) input. The IMU has three sensors. One of those sensors is the gyroscope; it would monitor the sensor. When monitoring the sensor, it would measure the rotational motion. It would as well send data to the user. Another sensor that the IMU has is the accelerometer; it would monitor the sensor. While monitoring the sensor, it would measure the acceleration and calculate the position. The last sensor the IMU has is the magnetometer; it will monitor the sensor. When monitoring the sensor, it will measure the magnetic force. As well, it will send data to the user. Another sensor input is the gap distance input; it will monitor the sensor. When monitoring the senor, it will see if the gap distance is correct or low. If the gap distance is low, then it will increase RMP. The gap input would also send data to the user. The last sensor input is the camera input; it would record and send data to the user.

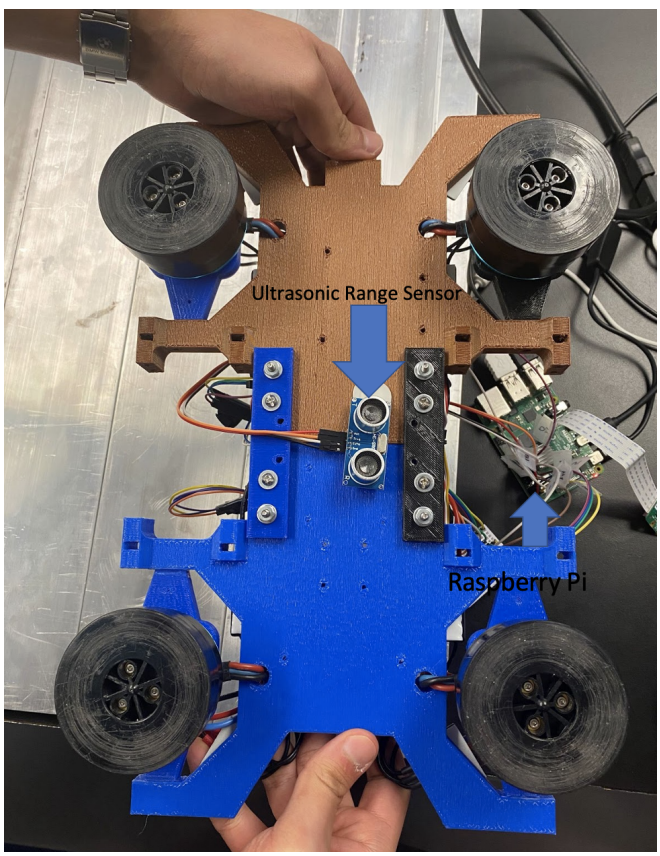
Cost & List Analysis:

Type of Sensor	Price	Description	What it is used for	Link
BMP388 Barometric Pressure Sensor 	$16.99	-measures **pressure** and **temperature** -Altitude tracing -supports I2C and SPI interface	-Pressure Sensors to be implemented for the environment and the tube -Temperature Sensor to sense temperatures for every surface, the pod, battery, engine, wheels, etc.	Amazon
Infrared Distance Sensor 	$29.99	-a near-infrared sensor for capturing measurements between moving and stationary objects -Non-contact distance measurement with a 2-200 cm range -accuracy 0.1cm resolution	-For measuring the distance between the pod and the floor when it is hovering. -Also used for measuring the distance between the pod and the wall -Will measure if the distance is low	Amazon
IMU 	$24.95	-includes Accelerometer, Gyroscope, Magnetometer -measures velocity, acceleration, and heading -motion-sensing system-in-a-chip	-Measure the speed of the pod, calculate its current position. -Measure rotational motion -Measure magnetic force	Amazon
Camera Module 	$23.90	-communicates with the raspberry pi using the camera interface -capable of 3280 x 2464 pixel static images -supports 1080p, 720p, and 480p	-capture videos	Amazon
Raspberry Pi 4 Model B 	$42.49	-Capacity: 2GB of RAM -bluetooth, Wi-Fi -64-bit quad-core processor	-Implementing all the sensors	Amazon

Num. of parts: **5** Total cost: **$138.32**

Results:

In this project, we focused on improving the previous design of the hyperloop specifically on the sensors and the control of the pod. We were using the parts used previously discussed to analyze and see what needed to be improved. We were able to implement all the sensors in the pod. We were able to write codes for the sensors to retrieve data from the sensors for analysis. We were also able to place the temperature and IMU sensors on the top of the chassis of the pod and the range sensor at the bottom. However, the range sensor was replaced with an ultrasonic range sensor due to a hardware problem, which will be discussed further in the discussion section. In addition, we managed to control all the four hover motors by using the raspberry pi. Furthermore, we were able to replace the receiver, and transmitter from the previous design and use the raspberry pi as a transmitter to control the RPMs of the motors and the height levitation of the pod.



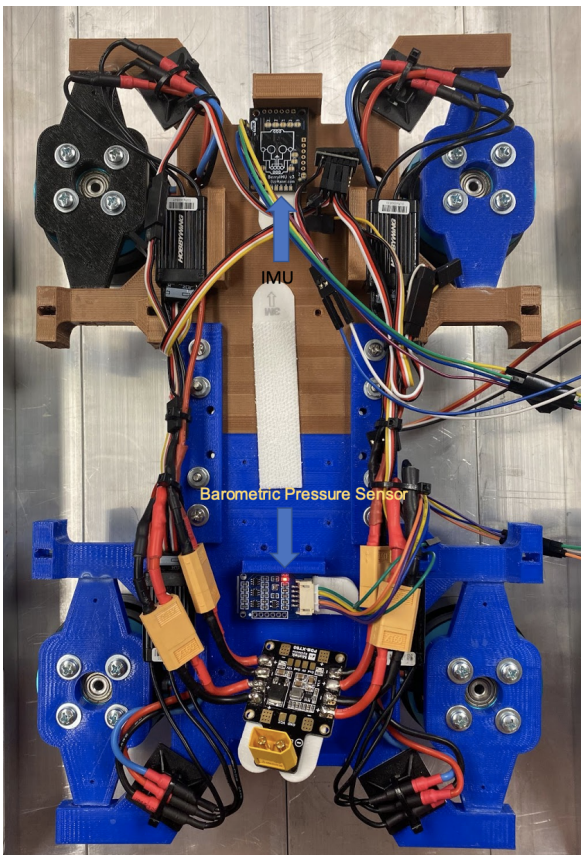


Figure 4.1 & 4.2 - Top and Bottom of the pod with the sensors installed

The programming language that we used for this project was python and you find the code in appendix A. We managed to retrieve the reference codes for IMU and modified it to read and display accelerations and angles. We also wrote the code for the temperature sensor and it worked perfectly. We created a GUI to output the sensors' values in a new window. Also, we created an algorithm to control the RPMs of the hover motors depending on the distance to the

ground recorded by the range ultrasonic sensor to draw the least possible amount of power from the battery and maintain a desired height. We were not able to finish the whole design that we wanted because we were not able to implement the servo motors designs to control the direction of the pod. The reason for this is that the previous structural design did not work with the servo motors and we were lacking parts (two servo motors) that the previous team had, but it was not possible to recover them. However, we started working towards a solution to be able to fix the previous design without having to change the servo motors that were already bought. We designed a four to one ratio gear system that could be implemented in the previous design, but we would have to alter the chassis and reprint it completely. We were not able to do this due to lacking parts, budget, and time.

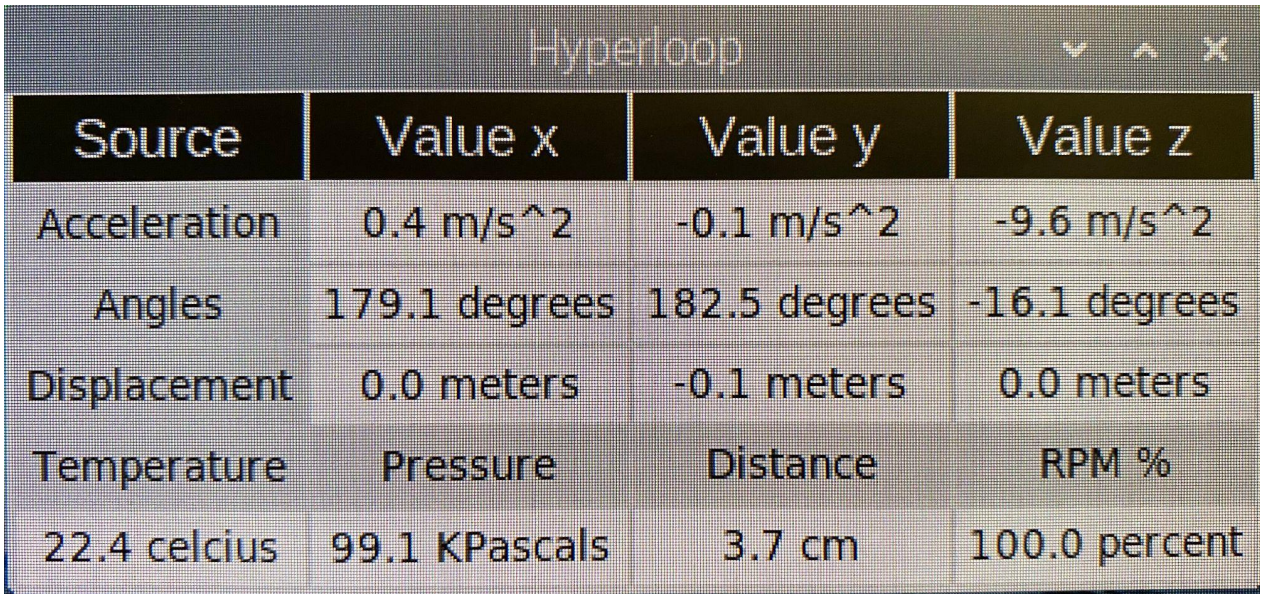


Figure 5 - GUI with sensor data

We did further testing to find out how much weight the pod could resist. So first we found out the weight of the prototype without the battery and then the weight of the battery. Then we started the pod and started adding weights on top until we found out the maximum weight that the pod can support while maintaining levitation. Results are shown in the table below.

Item	Weight (Kg)
Prototype (Pod)	2.05
Battery	0.368
Maximum Added Weight	1.57
Total Maximum Weight	3.988

Table 1 - Weight

Then we found out the maximum current consumption of the pod by connecting a multimeter in series with one of the motors. We found the amount of current it needs to function at the start to get the initial boost to start spinning and at a steady state at 100% rpms after it already started spinning. Then we just multiply that result times the 4 hoover engines to get the total current drawn from the battery. Results are shown in the table below.

Item	Current (mA)
1 Hoover Engine (Boost)	15
1 Hoover Engine (Steady)	9.6
Total Current Consumption (Boost)	60
Total Current Consumption (Steady)	38.4

Table 2 - Current

Discussion:

At the start of our project, we ordered all the parts we needed, the IMU, barometric pressure and temperature sensor, Laser PING distance sensor, camera module, and the Raspberry Pi. When the Laser PING Rangefinder sensor arrived, we noticed it arrived in poor handling, but we were not sure if it was damaged. The pin holders appeared melted; however, the chip did not look damaged, so we hoped for the best and continued on. When working with the Laser PING sensor, we first decided to write it in python. Eventually, after researching, we could not find any reference code of python for the Laser PING sensor, so we changed to writing the code in C because we thought we would be able to find C code for the Laser PING sensor. In the Laser Ping sensor datasheet, it said it had a C code library, but we were not able to find it. We decided to focus on another sensor because we were behind schedule. We started to work with the IMU; we found reference code in language C. We modified it to create our own code and we were able to get it working. It gave us what we needed which was acceleration and angles. To get the distance from the IMU, we needed to write more code. The distance is not very accurate when just using the IMU. We wrote more code to fix the error, but it is not reliable. We then wrote the code for the pressure sensor; it worked well, we had no problem, and we were receiving data with no error. Then we started working with the Laser PING sensor again, and after long research we were able to find a C code library that could work with the sensor. The code that we wrote did not work when implemented and we knew that it should have been working. We inspected the chip with a magnifying glass and we realized a defect in the chip in one of the soldering joints. Due to little time left, we decided to use another readily available distance sensor, the ultrasonic range sensor. After changing the sensor, we were able to write python code for it. The C code did not work with the sensor, so we had to translate all the codes we already wrote from C to python. It was quite easy doing that, it just took some time.

After the three sensors were working individually, we started to implement all the sensors together. We combined the code together to have all three sensors working and getting data at the same time. We started with the servos, but we only had two out of the four available for us to use. While we were trying to get in contact with the previous team, we started working on code for the servos and we were able to learn how to control them. After working with the two servos, we realized the servos were not suitable for the project because the servos were not able to turn in a five-degree angle that was needed; the minimum the servos would turn was at a twenty-degree angle. This is four times too much of what was needed. As well, the pod design needed to be reprinted and fixed because the hoover motor mount was not placed correctly. To fix that, we would need to reprint the chassis and the motor mounts again with some modifications. But first, we would need to fix the problem of the servo motors' turn angle. One of our ideas was to implement a gear system, a four-to-one ratio to decrease the angle from twenty-degree to a five-degree angle. We were able to create a design of a gear system to implement it with the servo motors, but we needed to altere the chassis due to the lack of space to implement this gear system. The problem was that one member of the previous team had the other two servos in his possession and we were unable to get into contact with him to get the remaining servos back.

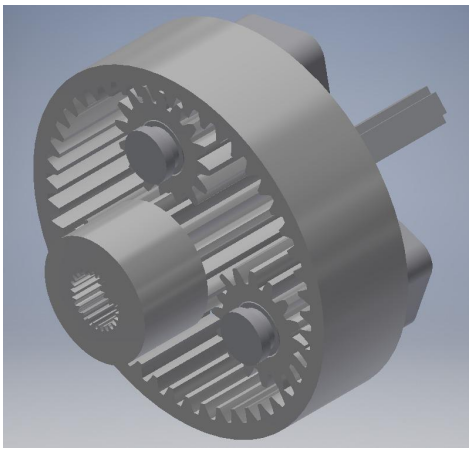


Figure 6 - Gear system connected to the axis of rotation of the mount

We decided to set that aside to start working with the electric speed controller of the pod. We began to work with the Raspberry Pi and tried to control the electric speed controllers. We encountered an error from the speed controllers in the form of sounds and they were not running at all. After careful thinking and consideration, we decided to use a logic analyzer to retrieve the signal that was working with the receiver and transmitter. In the standby position, we wanted to see when the signal was not outputting an error to the electric speed controller. We discovered the signal was a 55 Hz pulse with a 5% duty cycle to get the motor on standby. The error we had before was due to giving it a zero percent duty cycle. With the logic analyzer, we found that the max rpm with the receiver signal was a 55 Hz pulse with a 10% duty cycle. After doing this, we were able to replicate the signal with the Raspberry Pi and were able to control the motors with no issue. We also wrote code to control it manually with the Raspberry Pi to send a sequence to control the rpm of the motor to control the pod. Then we started testing and experimenting and

found out that the max rpms signal possible was a 55 Hz pulse with a 12% duty cycle. We created the GUI to output the sensor's values in a new window to be easily readable. We added a condition of when the temperature detects the hyperloop overheating, then it would print a warning to let the user know that it will shut down after 10 seconds. And finally, we created an algorithm to decrease or increase the rpms of the hoover motors to keep a certain desired distance so the motors can draw less energy from the battery and the pod can last longer. We tested our hypothesis of the motors running in full battery with or without code. The average time was 9 minutes without code. With code the average time was 9 minutes and 45 seconds. We saw an improvement of battery charge durability by about 10%.

Conclusion

The Hyperloop development is at its early stages and scientists and developers are moving first to offer the world the best prototype for real development and implementation of a working Hyperloop. Elon Musk's plan is the premiere model but has received skepticism from scientists who argue that it misses addressing some important issues. Our group's main objective was to improve the kinks noted in the works of the previous group. We included several sensors that helped improve our control system. We have sensors for temperature, pressure, and ultrasonic distance sensors that are designed to perform specific functions. Our control system has an inertial measurement unit(IMU) which includes an accelerometer, gyroscope, magnetometer, and a barometer each with a specific role to perform in the entire matrix of the control system. We were able to add the sensor system to the pod and with the sensors' feedback, we were able to control the RPMs of the motors and reduce the amount of power drawn in the battery. Also, we used the feedback to add some safeguards to protect the integrity of the pod. We could say that we were able to improve the hyperloop pod, and leave the first steps for another team to keep working on this project to make it completely autonomous. If we had more time, we would have been able to use the servo motors to control the movement of the pod. We hope that another group takes interest in this project and decides to keep improving it to accomplish our, and the previous team, goals. In the future, if a team were to continue with this project, it would be good to implement a custom board where all the sensors can be implemented into it.

Works Cited

[1] MIT Hyperloop Team. Final Report: An overview of the design, build, and testing process for MIT's entry in the SpaceX Hyperloop Competition, 2017. Accessed from http://web.mit.edu/mopg/www/papers/MITHyperloop_FinalReport_2017_public.pdf

[2] Analog sensor vs Digital sensor | Difference between Analog sensor and Digital sensor. Retrieved from https://www.rfwireless-world.com/Terminology/Analog-sensor-vs-Digital-sensor.html

[3] "Laser Sensors for The Hyperloop". *Althensensors.Com*, https://www.althensensors.com/application-stories/laser-sensors-for-the-hyperloop/.

[4] Catterall, Connor et al. *Hyperlynx Final Design Report*. 2015, https://www.pdf-archive.com/2015/12/05/finalreport-hyperlynx/finalreport-hyperlynx.pdf. Accessed 24 Nov 2020.

[5] Arizona's SpaceX Hyperloop Competition (AZ Loop). Final Design Package. Accessed from https://www.freedomsphoenix.com/Media/Media-Files/AZLoop-Final-Design-Package.pdf

Appendix A(code):

```
#Libraries

#Pressure Libraries
import time
import smbus
import math

#IMU Libraries
import sys
import IMU
import datetime
import os

#Distance Libraries
import RPi.GPIO as GPIO

#ESC Liraries
import signal
import atexit
import random

#GUI
from Tkinter import Scale, Tk, Frame, Label, Button
from ttk import Notebook,Entry

#******************** END LIBRARIES********************

#Variables

#Pressure
# define BMP388 Device I2C address
I2C_ADD_BMP388_AD0_LOW   =   0x76
I2C_ADD_BMP388_AD0_HIGH   =   0x77
I2C_ADD_BMP388   =      I2C_ADD_BMP388_AD0_LOW

BMP388_REG_ADD_WIA   =   0x00
BMP388_REG_VAL_WIA      =   0x50

BMP388_REG_ADD_ERR   =   0x02
```

```
BMP388_REG_VAL_FATAL_ERR   =   0x01
BMP388_REG_VAL_CMD_ERR     =   0x02
BMP388_REG_VAL_CONF_ERR    =   0x04

BMP388_REG_ADD_STATUS   =   0x03
BMP388_REG_VAL_CMD_RDY     =   0x10
BMP388_REG_VAL_DRDY_PRESS   =   0x20
BMP388_REG_VAL_DRDY_TEMP   =   0x40

BMP388_REG_ADD_CMD   =   0x7E
BMP388_REG_VAL_EXTMODE_EN   =   0x34
BMP388_REG_VAL_FIFI_FLUSH   =   0xB0
BMP388_REG_VAL_SOFT_RESET   =   0xB6

BMP388_REG_ADD_PWR_CTRL   =   0x1B
BMP388_REG_VAL_PRESS_EN    =   0x01
BMP388_REG_VAL_TEMP_EN     =   0x02
BMP388_REG_VAL_NORMAL_MODE   =   0x30

BMP388_REG_ADD_PRESS_XLSB   =   0x04
BMP388_REG_ADD_PRESS_LSB   =   0x05
BMP388_REG_ADD_PRESS_MSB   =   0x06
BMP388_REG_ADD_TEMP_XLSB   =   0x07
BMP388_REG_ADD_TEMP_LSB    =   0x08
BMP388_REG_ADD_TEMP_MSB    =   0x09

BMP388_REG_ADD_T1_LSB   =   0x31
BMP388_REG_ADD_T1_MSB   =   0x32
BMP388_REG_ADD_T2_LSB   =   0x33
BMP388_REG_ADD_T2_MSB   =   0x34
BMP388_REG_ADD_T3      =   0x35
BMP388_REG_ADD_P1_LSB   =   0x36
BMP388_REG_ADD_P1_MSB   =   0x37
BMP388_REG_ADD_P2_LSB   =   0x38
BMP388_REG_ADD_P2_MSB   =   0x39
BMP388_REG_ADD_P3      =   0x3A
BMP388_REG_ADD_P4      =   0x3B
BMP388_REG_ADD_P5_LSB   =   0x3C
BMP388_REG_ADD_P5_MSB   =   0x3D
BMP388_REG_ADD_P6_LSB   =   0x3E
```

```
BMP388_REG_ADD_P6_MSB    =    0x3F
BMP388_REG_ADD_P7        =    0x40
BMP388_REG_ADD_P8        =    0x41
BMP388_REG_ADD_P9_LSB    =    0x42
BMP388_REG_ADD_P9_MSB    =    0x43
BMP388_REG_ADD_P10       =    0x44
BMP388_REG_ADD_P11       =    0x45

#IMU
RAD_TO_DEG = 57.29578
M_PI = 3.14159265358979323846
G_GAIN = 0.070          # [deg/s/LSB]  If you change the dps for gyro, you need to update this value accordingly
AA =  0.40              # Complementary filter constant
MAG_LPF_FACTOR = 0.4    # Low pass filter constant magnetometer
ACC_LPF_FACTOR = 0.4    # Low pass filter constant for accelerometer
ACC_MEDIANTABLESIZE = 9         # Median filter table size for accelerometer. Higher = smoother but a longer delay
MAG_MEDIANTABLESIZE = 9         # Median filter table size for magnetometer. Higher = smoother but a longer delay

################## Compass Calibration values ############
# Use calibrateBerryIMU.py to get calibration values
# Calibrating the compass isnt mandatory, however a calibrated
# compass will result in a more accurate heading value.

magXmin =  -2743
magYmin =  -1896
magZmin =  -522
magXmax =  234
magYmax =  738
magZmax =  1279

################ END Calibration offsets #################

gyroXangle = 0.0
gyroYangle = 0.0
gyroZangle = 0.0
CFangleX = 0.0
CFangleY = 0.0
```

```
CFangleXFiltered = 0.0
CFangleYFiltered = 0.0
kalmanX = 0.0
kalmanY = 0.0
oldXMagRawValue = 0
oldYMagRawValue = 0
oldZMagRawValue = 0
oldXAccRawValue = 0
oldYAccRawValue = 0
oldZAccRawValue = 0


#Kalman filter variables
Q_angle = 0.02
Q_gyro = 0.0015
R_angle = 0.005
y_bias = 0.0
x_bias = 0.0
XP_00 = 0.0
XP_01 = 0.0
XP_10 = 0.0
XP_11 = 0.0
YP_00 = 0.0
YP_01 = 0.0
YP_10 = 0.0
YP_11 = 0.0
KFangleX = 0.0
KFangleY = 0.0

#Distance - no variables


#Algorithm
u = 0
rateofchange = 0.02
DisplacementX = 0.00
DisplacementY = 0.00
DisplacementZ = 0.00
r = 0
t = 0
```

```
j = 0
l = 1
#******************** END VARIABLES**************************

#Classes and methods

#Pressure
class BMP388(object):
   """docstring for BMP388"""
   def __init__(self, address=I2C_ADD_BMP388):
      self._address = address
      self._bus = smbus.SMBus(1)
      # Load calibration values.
      if self._read_byte(BMP388_REG_ADD_WIA) == BMP388_REG_VAL_WIA:
         print("Pressure sersor is BMP388!\r\n")
         u8RegData = self._read_byte(BMP388_REG_ADD_STATUS)
         if ( u8RegData & BMP388_REG_VAL_CMD_RDY ):
            self._write_byte(BMP388_REG_ADD_CMD, BMP388_REG_VAL_SOFT_RESET)
            time.sleep(0.01)
      else:
         print("Pressure sersor NULL!\r\n")
      self._write_byte( BMP388_REG_ADD_PWR_CTRL,BMP388_REG_VAL_PRESS_EN |
BMP388_REG_VAL_TEMP_EN | BMP388_REG_VAL_NORMAL_MODE)
      self._load_calibration()

   def _read_byte(self,cmd):
      return self._bus.read_byte_data(self._address,cmd)

   def _read_s8(self,cmd):
      result = self._read_byte(cmd)
      if result > 128:result -= 256
      return result

   def _read_u16(self,cmd):
      LSB = self._bus.read_byte_data(self._address,cmd)
      MSB = self._bus.read_byte_data(self._address,cmd+1)
      return (MSB    << 8) + LSB

   def _read_s16(self,cmd):
      result = self._read_u16(cmd)
```

```
    if result > 32767:result -= 65536
    return result

def _write_byte(self,cmd,val):
    self._bus.write_byte_data(self._address,cmd,val)

def _load_calibration(self):
    print("_load_calibration\r\n")
    "load calibration"
    """ read the temperature calibration parameters """
    self.T1 =self._read_u16(BMP388_REG_ADD_T1_LSB)
    self.T2 =self._read_u16(BMP388_REG_ADD_T2_LSB)
    self.T3 =self._read_s8(BMP388_REG_ADD_T3)
    """ read the pressure calibration parameters """
    self.P1 =self._read_s16(BMP388_REG_ADD_P1_LSB)
    self.P2 =self._read_s16(BMP388_REG_ADD_P2_LSB)
    self.P3 =self._read_s8(BMP388_REG_ADD_P3)
    self.P4 =self._read_s8(BMP388_REG_ADD_P4)
    self.P5 =self._read_u16(BMP388_REG_ADD_P5_LSB)
    self.P6 =self._read_u16(BMP388_REG_ADD_P6_LSB)
    self.P7 =self._read_s8(BMP388_REG_ADD_P7)
    self.P8 =self._read_s8(BMP388_REG_ADD_P8)
    self.P9 =self._read_s16(BMP388_REG_ADD_P9_LSB)
    self.P10 =self._read_s8(BMP388_REG_ADD_P10)
    self.P11=self._read_s8(BMP388_REG_ADD_P11)
    #print(self.T1)
    #print(self.T2)
    #print(self.T3)
    #print(self.P1)
    #print(self.P2)
    #print(self.P3)
    #print(self.P4)
    #print(self.P5)
    #print(self.P6)
    #print(self.P7)
    #print(self.P8)
    #print(self.P9)
    #print(self.P10)
    #print(self.P11)
def compensate_temperature(self,adc_T):
```

```
        partial_data1 =  (adc_T - (256 *  (self.T1)))
        partial_data2 =  (self.T2 * partial_data1)
        partial_data3 =  (partial_data1 * partial_data1)
        partial_data4 =  (( partial_data3) * ( self.T3))
        partial_data5 = ( (( partial_data2) * 262144) +  partial_data4)
        partial_data6 =  (( partial_data5) / 4294967296)
        self.T_fine = partial_data6
        comp_temp =  ((partial_data6 * 25)  / 16384)
        return comp_temp;

    def compensate_pressure(self,adc_P):
        partial_data1 = self.T_fine * self.T_fine
        partial_data2 = partial_data1 / 64
        partial_data3 = (partial_data2 * self.T_fine) / 256
        partial_data4 = (self.P8 * partial_data3) / 32
        partial_data5 = (self.P7 * partial_data1) * 16
        partial_data6 = (self.P6 * self.T_fine) * 4194304;
        offset =  ( (self.P5) *  140737488355328) + partial_data4 + partial_data5 + partial_data6

        partial_data2 = (( self.P4) * partial_data3) / 32
        partial_data4 = (self.P3 * partial_data1) * 4
        partial_data5 = ( (self.P2) - 16384) * ( self.T_fine) * 2097152
        sensitivity = (( (self.P1) - 16384) *  70368744177664) + partial_data2 + partial_data4 +
partial_data5

        partial_data1 = (sensitivity / 16777216) * adc_P
        partial_data2 =  (self.P10) *  (self.T_fine)
        partial_data3 = partial_data2 + (65536 *  (self.P9))
        partial_data4 = (partial_data3 * adc_P) / 8192
        partial_data5 = (partial_data4 * adc_P) / 512
        partial_data6 =  ( adc_P *  adc_P)
        partial_data2 = ( (self.P11) *  (partial_data6)) / 65536
        partial_data3 = (partial_data2 * adc_P) / 128
        partial_data4 = (offset / 4) + partial_data1 + partial_data5 + partial_data3
        comp_press = (( partial_data4 * 25) /  1099511627776)
        return comp_press;

    def get_temperature_and_pressure_and_altitude(self):
        """Returns pressure in Pa as double. Output value of "6386.2"equals 96386.2 Pa = 963.862
hPa."""
```

```
        xlsb = self._read_byte(BMP388_REG_ADD_TEMP_XLSB)
        lsb =  self._read_byte(BMP388_REG_ADD_TEMP_LSB)
        msb =  self._read_byte(BMP388_REG_ADD_TEMP_MSB)
        adc_T = (msb << 16) + (lsb << 8) + (xlsb)
        temperature = self.compensate_temperature(adc_T)
        xlsb = self._read_byte(BMP388_REG_ADD_PRESS_XLSB)
        lsb =  self._read_byte(BMP388_REG_ADD_PRESS_LSB)
        msb =  self._read_byte(BMP388_REG_ADD_PRESS_MSB)

        adc_P = (msb << 16) + (lsb << 8) + (xlsb)
        pressure = self.compensate_pressure(adc_P)
        altitude  = 4433000 * (1 - pow(((pressure/100.0) / 101325.0), 0.1903))

        return temperature,pressure,altitude
def pressureMethod():
    try:
        #time.sleep(0.5)
        temperature,pressure,altitude = bmp388.get_temperature_and_pressure_and_altitude()
        finalTemperature = temperature/100.0
        finalPressure = pressure/100000.0
        finalAltitude = altitude/100.0
        #print(' Temperature = %.1f Pressure = %.2f  Altitude =%.2f
'%(finalTemperature,finalPressure,finalAltitude))
        return finalTemperature, finalPressure, finalAltitude
    except IOError as e:
        print("IO error detected")
        print(e)


#IMU - no methods

def kalmanFilterY ( accAngle, gyroRate, DT):
    y=0.0
    S=0.0

    global KFangleY
    global Q_angle
    global Q_gyro
    global y_bias
    global YP_00
```

```
    global YP_01
    global YP_10
    global YP_11

    KFangleY = KFangleY + DT * (gyroRate - y_bias)

    YP_00 = YP_00 + ( - DT * (YP_10 + YP_01) + Q_angle * DT )
    YP_01 = YP_01 + ( - DT * YP_11 )
    YP_10 = YP_10 + ( - DT * YP_11 )
    YP_11 = YP_11 + ( + Q_gyro * DT )

    y = accAngle - KFangleY
    S = YP_00 + R_angle
    K_0 = YP_00 / S
    K_1 = YP_10 / S

    KFangleY = KFangleY + ( K_0 * y )
    y_bias = y_bias + ( K_1 * y )

    YP_00 = YP_00 - ( K_0 * YP_00 )
    YP_01 = YP_01 - ( K_0 * YP_01 )
    YP_10 = YP_10 - ( K_1 * YP_00 )
    YP_11 = YP_11 - ( K_1 * YP_01 )

    return KFangleY

def kalmanFilterX ( accAngle, gyroRate, DT):
    x=0.0
    S=0.0

    global KFangleX
    global Q_angle
    global Q_gyro
    global x_bias
    global XP_00
    global XP_01
    global XP_10
    global XP_11
```

```
    KFangleX = KFangleX + DT * (gyroRate - x_bias)

    XP_00 = XP_00 + ( - DT * (XP_10 + XP_01) + Q_angle * DT )
    XP_01 = XP_01 + ( - DT * XP_11 )
    XP_10 = XP_10 + ( - DT * XP_11 )
    XP_11 = XP_11 + ( + Q_gyro * DT )

    x = accAngle - KFangleX
    S = XP_00 + R_angle
    K_0 = XP_00 / S
    K_1 = XP_10 / S

    KFangleX = KFangleX + ( K_0 * x )
    x_bias = x_bias + ( K_1 * x )

    XP_00 = XP_00 - ( K_0 * XP_00 )
    XP_01 = XP_01 - ( K_0 * XP_01 )
    XP_10 = XP_10 - ( K_1 * XP_00 )
    XP_11 = XP_11 - ( K_1 * XP_01 )

    return KFangleX

#Distance

def checkdist():
        GPIO.output(16, GPIO.HIGH)
        time.sleep(0.000015)
        GPIO.output(16, GPIO.LOW)
        while not GPIO.input(18):
                pass
        t1 = time.time()
        while GPIO.input(18):
                pass
        t2 = time.time()
        return(t2-t1)*340/2

def distanceMethod():
    distancevalue = checkdist()
    #print 'Distance: %0.3f m' %distancevalue
    time.sleep(0.5)
```

```
    return distancevalue

#ESC

#GUI
def displaytable_constants():
  label = Label(window, text="Source", bg="black", fg="white", padx=3, pady=3)
  label.config(font=('Arial', 14))
  label.grid(row=0, column=0, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(0, weight=1)

  label = Label(window, text="Value x", bg="black", fg="white", padx=3, pady=3)
  label.config(font=('Arial', 14))
  label.grid(row=0, column=1, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(1, weight=1)

  label = Label(window, text="Value y", bg="black", fg="white", padx=3, pady=3)
  label.config(font=('Arial', 14))
  label.grid(row=0, column=2, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(2, weight=1)

  label = Label(window, text="Value z", bg="black", fg="white", padx=3, pady=3)
  label.config(font=('Arial', 14))
  label.grid(row=0, column=3, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(3, weight=1)

  label = Label(window, text="Temperature", bg="light grey",fg="black",padx=3,pady=3)
  label.grid(row=4, column=0, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(0, weight=1)

  label = Label(window, text="Pressure", bg="light grey",fg="black",padx=3,pady=3)
  label.grid(row=4, column=1, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(1, weight=1)

  label = Label(window, text="Acceleration", bg="light grey",fg="black",padx=3,pady=3)
  label.grid(row=1, column=0, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(0, weight=1)

  label = Label(window, text="Displacement", bg="light grey",fg="black",padx=3,pady=3)
```

```
  label.grid(row=3, column=0, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(0, weight=1)

  label = Label(window, text="Distance", bg="light grey",fg="black",padx=3,pady=3)
  label.grid(row=4, column=2, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(2, weight=1)

  label = Label(window, text="Angles", bg="light grey",fg="black",padx=3,pady=3)
  label.grid(row=2, column=0, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(0, weight=1)

  label = Label(window, text="RPM %", bg="light grey",fg="black",padx=3,pady=3)
  label.grid(row=4, column=3, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(3, weight=1)
#  window.after(1000,displaytable)

def displaytable_content(temp,press,accx,accy,accz,dx,dy,dz,dis,angx,angy,angz,rpm):
  label = Label(window, text="%3.1f celcius" % temp, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=5, column=0, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(0, weight=1)

  label = Label(window, text="%3.1f KPascals" % press, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=5, column=1, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(1, weight=1)

  label = Label(window, text="%3.1f m/s^2" % accx, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=1, column=1, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(1, weight=1)

  label = Label(window, text="%3.1f m/s^2" % accy, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=1, column=2, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(2, weight=1)

  label = Label(window, text="%3.1f m/s^2" % accz, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=1, column=3, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(3, weight=1)

  label = Label(window, text="%3.1f meters" % dx, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=3, column=1, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(1, weight=1)
```

```
  label = Label(window, text="%3.1f meters" % dy, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=3, column=2, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(2, weight=1)

  label = Label(window, text="%3.1f meters" % dz, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=3, column=3, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(3, weight=1)

  label = Label(window, text="%3.1f cm" % dis, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=5, column=2, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(2, weight=1)

  label = Label(window, text="%3.1f degrees" % angx, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=2, column=1, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(1, weight=1)

  label = Label(window, text="%3.1f degrees" % angy, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=2, column=2, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(2, weight=1)

  label = Label(window, text="%3.1f degrees" % angz, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=2, column=3, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(3, weight=1)

  label = Label(window, text="%3.1f percent" % rpm, bg="white",fg="black",padx=3,pady=3)
  label.grid(row=5, column=3, sticky="nsew", padx=1, pady=1)
  window.grid_columnconfigure(3, weight=1)

#***************** END METHODS/Classes**************************

#Code before loop

#Pressure
bmp388 = BMP388()
#IMU
a = datetime.datetime.now()

#Setup the tables for the mdeian filter. Fill them all with '1' so we dont get devide by zero error
acc_medianTable1X = [1] * ACC_MEDIANTABLESIZE
```

```
acc_medianTable1Y = [1] * ACC_MEDIANTABLESIZE
acc_medianTable1Z = [1] * ACC_MEDIANTABLESIZE
acc_medianTable2X = [1] * ACC_MEDIANTABLESIZE
acc_medianTable2Y = [1] * ACC_MEDIANTABLESIZE
acc_medianTable2Z = [1] * ACC_MEDIANTABLESIZE
mag_medianTable1X = [1] * MAG_MEDIANTABLESIZE
mag_medianTable1Y = [1] * MAG_MEDIANTABLESIZE
mag_medianTable1Z = [1] * MAG_MEDIANTABLESIZE
mag_medianTable2X = [1] * MAG_MEDIANTABLESIZE
mag_medianTable2Y = [1] * MAG_MEDIANTABLESIZE
mag_medianTable2Z = [1] * MAG_MEDIANTABLESIZE

IMU.detectIMU()     #Detect if BerryIMU is connected.
if(IMU.BerryIMUversion == 99):
    print(" No BerryIMU found... exiting ")
    sys.exit()
IMU.initIMU()       #Initialise the accelerometer, gyroscope and compass

#Distance
GPIO.setmode(GPIO.BOARD)
GPIO.setup(16,GPIO.OUT,initial=GPIO.LOW)
GPIO.setup(18,GPIO.IN)
time.sleep(2)

#ESCs
atexit.register(GPIO.cleanup)

servopin = 11
maximum = 10
minimum = 5
print("starting program")
GPIO.setmode(GPIO.BOARD)
GPIO.setup(servopin,GPIO.OUT, initial=False)
p = GPIO.PWM(servopin, 55)
p.start(minimum)
time.sleep(5)
speed = 12.0

#GUI
window=Tk()
```

```
window.title("Hyperloop")
displaytable_constants()

#************** END OF CODE BEFORE
LOOP**************************************

#While Loop CODE
while (l==1):


    #Pressure
    temperature,pressure,altitude = pressureMethod()                #VARIABLES TO PRINT IN
GUI
    #print (temperature, pressure, altitude)
    #IMU
    #Read the accelerometer,gyroscope and magnetometer values
    ACCx = IMU.readACCx()
    ACCy = IMU.readACCy()
    ACCz = IMU.readACCz()
    GYRx = IMU.readGYRx()
    GYRy = IMU.readGYRy()
    GYRz = IMU.readGYRz()
    MAGx = IMU.readMAGx()
    MAGy = IMU.readMAGy()
    MAGz = IMU.readMAGz()


    #Apply compass calibration
    MAGx -= (magXmin + magXmax) /2
    MAGy -= (magYmin + magYmax) /2
    MAGz -= (magZmin + magZmax) /2


    ##Calculate loop Period(LP). How long between Gyro Reads
    b = datetime.datetime.now() - a
    a = datetime.datetime.now()
    LP = b.microseconds/(1000000*1.0)
    outputString = "Loop Time %5.2f " % ( LP )
```

```
################################################
#### Apply low pass filter ####
################################################
oldXMagRawValue = MAGx
oldYMagRawValue = MAGy
oldZMagRawValue = MAGz
oldXAccRawValue = ACCx
oldYAccRawValue = ACCy
oldZAccRawValue = ACCz

MAGx =  MAGx  * MAG_LPF_FACTOR + oldXMagRawValue*(1 - MAG_LPF_FACTOR);
MAGy =  MAGy  * MAG_LPF_FACTOR + oldYMagRawValue*(1 - MAG_LPF_FACTOR);
MAGz =  MAGz  * MAG_LPF_FACTOR + oldZMagRawValue*(1 - MAG_LPF_FACTOR);
ACCx =  ACCx  * ACC_LPF_FACTOR + oldXAccRawValue*(1  - ACC_LPF_FACTOR);
ACCy =  ACCy  * ACC_LPF_FACTOR + oldYAccRawValue*(1  - ACC_LPF_FACTOR);
ACCz =  ACCz  * ACC_LPF_FACTOR + oldZAccRawValue*(1  - ACC_LPF_FACTOR);

#############################################
#### Median filter for accelerometer ####
#############################################
# cycle the table
for x in range (ACC_MEDIANTABLESIZE-1,0,-1 ):
   acc_medianTable1X[x] = acc_medianTable1X[x-1]
   acc_medianTable1Y[x] = acc_medianTable1Y[x-1]
   acc_medianTable1Z[x] = acc_medianTable1Z[x-1]

# Insert the lates values
acc_medianTable1X[0] = ACCx
acc_medianTable1Y[0] = ACCy
acc_medianTable1Z[0] = ACCz

# Copy the tables
acc_medianTable2X = acc_medianTable1X[:]
acc_medianTable2Y = acc_medianTable1Y[:]
acc_medianTable2Z = acc_medianTable1Z[:]

# Sort table 2
acc_medianTable2X.sort()
acc_medianTable2Y.sort()
```

```
acc_medianTable2Z.sort()

# The middle value is the value we are interested in
ACCx = acc_medianTable2X[int(ACC_MEDIANTABLESIZE/2)];
ACCy = acc_medianTable2Y[int(ACC_MEDIANTABLESIZE/2)];
ACCz = acc_medianTable2Z[int(ACC_MEDIANTABLESIZE/2)];



##########################################
#### Median filter for magnetometer ####
##########################################
# cycle the table
for x in range (MAG_MEDIANTABLESIZE-1,0,-1 ):
   mag_medianTable1X[x] = mag_medianTable1X[x-1]
   mag_medianTable1Y[x] = mag_medianTable1Y[x-1]
   mag_medianTable1Z[x] = mag_medianTable1Z[x-1]

# Insert the latest values
mag_medianTable1X[0] = MAGx
mag_medianTable1Y[0] = MAGy
mag_medianTable1Z[0] = MAGz

# Copy the tables
mag_medianTable2X = mag_medianTable1X[:]
mag_medianTable2Y = mag_medianTable1Y[:]
mag_medianTable2Z = mag_medianTable1Z[:]

# Sort table 2
mag_medianTable2X.sort()
mag_medianTable2Y.sort()
mag_medianTable2Z.sort()

# The middle value is the value we are interested in
MAGx = mag_medianTable2X[int(MAG_MEDIANTABLESIZE/2)];
MAGy = mag_medianTable2Y[int(MAG_MEDIANTABLESIZE/2)];
MAGz = mag_medianTable2Z[int(MAG_MEDIANTABLESIZE/2)];



#Convert Gyro raw to degrees per second
```

```
rate_gyr_x =  GYRx * G_GAIN
rate_gyr_y =  GYRy * G_GAIN
rate_gyr_z =  GYRz * G_GAIN


#Calculate the angles from the gyro.
gyroXangle+=rate_gyr_x*LP
gyroYangle+=rate_gyr_y*LP              #VARIABLES TO PRINT IN GUI
gyroZangle+=rate_gyr_z*LP


#Convert Accelerometer values to degrees
AccXangle =  (math.atan2(ACCy,ACCz)*RAD_TO_DEG)
AccYangle =  (math.atan2(ACCz,ACCx)+M_PI)*RAD_TO_DEG


#Change the rotation value of the accelerometer to -/+ 180 and
#move the Y axis '0' point to up.  This makes it easier to read.
if AccYangle > 90:
   AccYangle -= 270.0
else:
   AccYangle += 90.0

#Kalman filter used to combine the accelerometer and gyro values.
kalmanY = kalmanFilterY(AccYangle, rate_gyr_y,LP)
kalmanX = kalmanFilterX(AccXangle, rate_gyr_x,LP)

#Calculate heading
heading = 180 * math.atan2(MAGy,MAGx)/M_PI

#Only have our heading between 0 and 360
if heading < 0:
   heading += 360

############################################################################
####################Tilt compensated heading#########################
############################################################################
#Normalize accelerometer raw values.
accXnorm = ACCx/math.sqrt(ACCx * ACCx + ACCy *  ACCy + ACCz * ACCz)
accYnorm = ACCy/math.sqrt(ACCx * ACCx + ACCy *  ACCy + ACCz * ACCz)
```

```
    #Calculate pitch and roll
    pitch = math.asin(accXnorm)
    roll = -math.asin(accYnorm/math.cos(pitch))


    #Calculate the new tilt compensated values
    #The compass and accelerometer are orientated differently on the the BerryIMUv1, v2 and v3.
    #This needs to be taken into consideration when performing the calculations

    #X compensation
    if(IMU.BerryIMUversion == 1 or IMU.BerryIMUversion == 3):            #LSM9DS0 and
(LSM6DSL & LIS2MDL)
        magXcomp = MAGx*math.cos(pitch)+MAGz*math.sin(pitch)
    else:                                                #LSM9DS1
        magXcomp = MAGx*math.cos(pitch)-MAGz*math.sin(pitch)

    #Y compensation
    if(IMU.BerryIMUversion == 1 or IMU.BerryIMUversion == 3):            #LSM9DS0 and
(LSM6DSL & LIS2MDL)
        magYcomp =
MAGx*math.sin(roll)*math.sin(pitch)+MAGy*math.cos(roll)-MAGz*math.sin(roll)*math.cos(p
itch)
    else:                                                #LSM9DS1
        magYcomp =
MAGx*math.sin(roll)*math.sin(pitch)+MAGy*math.cos(roll)+MAGz*math.sin(roll)*math.cos(
pitch)




    #Calculate tilt compensated heading
    tiltCompensatedHeading = 180 * math.atan2(magYcomp,magXcomp)/M_PI

    if tiltCompensatedHeading < 0:
        tiltCompensatedHeading += 360
```

```
    ###################### END Tilt Compensation ########################

    #Read the accelerometer,gyroscope and magnetometer values
    ACCx2 = IMU.readACCx()
    ACCy2 = IMU.readACCy()
    ACCz2 = IMU.readACCz()


    yG2 = (ACCx2 * 0.244)/1000
    xG2 = (ACCy2 * 0.244)/1000
    zG2 = (ACCz2 * 0.244)/1000


    yG = (ACCx * 0.244)/1000
    xG = (ACCy * 0.244)/1000
    zG = (ACCz * 0.244)/1000


    ISAccx = xG * 9.81              #VARIABLES TO PRINT IN GUI
    ISAccy = yG * 9.81
    ISAccz = zG * 9.81


    if 0:
        outputString +=("Acc X = %fG  Acc Y =   %fG   Acc Z =  %fG  " % ( yG2, xG2, zG2))
#Raw acceleration in G

    if 0:
        outputString +=("ACC X = %fG  ACC Y =   %fG   ACC Z =  %fG  " % ( yG, xG, zG))
#Filtered Acceleration in G

    if 0:
        outputString +=("ACC X = %f m/s^2  ACC Y =   %f m/s^2  ACC Z =  %f m/s^2  " % (
ISAccx, ISAccy, ISAccz)) #Filtered Acceleration in m/s^2

    if 0:                       #Change to '0' to stop showing the angles from the accelerometer
        outputString += "#  ACCX Angle %5.2f ACCY  Angle %5.2f  #  " % (AccXangle,
AccYangle)

    if 0:                       #Change to '0' to stop  showing the angles from the gyro
```

```
        outputString +="\t# GRYX Angle %5.2f  GYRY Angle %5.2f  GYRZ Angle %5.2f # " %
(gyroXangle,gyroYangle,gyroZangle)

    if 0:                   #Change to '0' to stop  showing the heading
        outputString +="\t# HEADING %5.2f  tiltCompensatedHeading %5.2f #" %
(heading,tiltCompensatedHeading)


    #print(outputString)

    #slow program down a bit, makes the output more readable
    #time.sleep(0.03)

    #distance
    try:
        distanceLow = distanceMethod() * 100 #in cm
    except KeyboardInterrupt:                       #VARIABLE TO PRINT GUI
        GPIO.cleanup()
    #print 'Distance: %3.1f cm' % distanceLow

    #ESCs

    p.ChangeDutyCycle(speed)


    #Algorithm
    if(distanceLow < 3.0):
        speed += 0.1
    if(distanceLow > 3.0):
        speed -= 0.1
    #safeguards
    if(distanceLow <= 2.7):
        speed += 1
    if(speed > 12.0):
        speed = 12.0
    if(speed < 9.0):
        speed = 9.0
    if (u==1):
        if (PreviousAccx == 0.0 or PreviousAccy == 0.0 or PreviousAccz == 0.0):
            u = 0
```

```
    if (u == 1):
        if ((math.fabs((ISAccx - PreviousAccx) / PreviousAccx)) > 0.20):
            DisplacementX += ISAccx * LP * LP
            if (DisplacementX > 0 and t == 0):
                DisplacementX = 0.0
                t = 1
        if ((math.fabs((ISAccy - PreviousAccy) / PreviousAccy)) > 0.20):
            DisplacementY += ISAccy * LP * LP
            if (DisplacementY < 0 and j == 0):
                DisplacementY = 0.0
                j = 1
        if ((math.fabs((ISAccz - PreviousAccz) / PreviousAccz)) > 0.20):
            DisplacementZ += ISAccz * LP * LP
            if (DisplacementZ < 0 and r == 0):
                DisplacementZ = 0.0
                r = 1

    if (temperature > 27.0):
        print "Warning Temperature too high!!! Starting Shutdown in 10 seconds."
        time.sleep(10)
        l = 0

    if 0:
        outputString +=("Displacement X = %5.3f m  Displacement Y =   %5.3f m  Displacement Z
=  %5.3f m  " % ( DisplacementX, DisplacementY, DisplacementZ))

    #print(outputString)


    PreviousAccx = ISAccx
    PreviousAccy = ISAccy
    PreviousAccz = ISAccz

    u = 1

    rpm = (speed/12.0) * 100

    #GUI
```

```
displaytable_content(temperature,pressure,ISAccx,ISAccy,ISAccz,DisplacementX,Displacement
Y,DisplacementZ,distanceLow,kalmanX,kalmanY,gyroZangle,rpm)
    window.update_idletasks()
    window.update()

#****************** END WHILE LOOP*******************************

#******************** END OF CODE*********************************
```