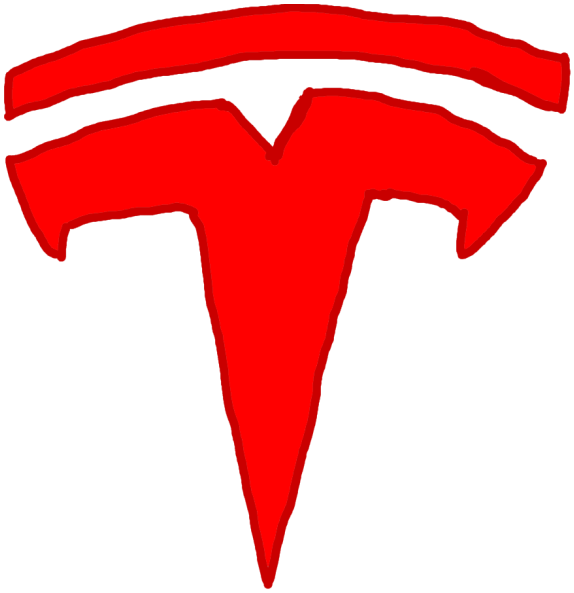
ECPE 196

Final Project Report

Team Budget Tesla



Michael Kmak, Hemad Mumtaz, Beau Pasquier, Oscar Wong, and Davis Young

Acknowledgments

For this project, we would like to thank the following people for their help and guidance through the design and implementation processes of this project:

- Dr. Basha
- Dr. Khoie
- Mark Foreman
- Dr. Pallipuram
- Dr. Mathews
- Dr. Mueller
- Jason To-Tran

Abstract/Problem Overview

Pacific's campus is just the right size to walk from one end to the other in about 15 minutes. However, for students with injuries or disabilities this 15-minute trip is magnified, and it can make a trip from one class to another impossible if the time in-between is too small. The goal of this project was to create an autonomous system that would enable the transfer of people around campus, specifically targeted towards those for whom travel is more difficult and time-consuming. To fill this niche, we refurbished an abandoned electric golf cart and rigged it with the processing and perceptive hardware necessary for automation.

Designing and implementing the system took place in distinct blocks, assigned among the team members. To analyze camera data and detect obstacles, a convolutional neural network is paired with a short-range FMCW radar sensor. Based on the sensor input, a mini computer processes the data and computes what the motor control should do in order to get the golf cart from point A to B in a safe manner. The path between those points is calculated and tracked via an onboard GPS unit. These electronics are powered by a marine battery coupled with buck converters. A microcontroller-operated winch is used to activate the brake pedal and the throttle is controlled with a digital potentiometer. The cart's power is supplied by a refurbished array of batteries, sufficient for several hours of operation.

Project Objectives

This transportation system is not limited to doing tours for visitors or pickup for disabled students but should be a general form of transportation accessible by all. There is a variability in the pickup and destination locations implemented by the system, but these locations are limited to the general Stockton campus of University of the Pacific. All locations are connected by a paved road that our vehicle can safely drive on. The accuracy of the cart will drop off and pick up at the paved road closest to the main entrance of the location, within 10 meters from the front entrance of all specified locations. Each named building has at least one such point, with some (such as the University Center) having multiple for convenience. The cart is able to aptly adapt to changes in campus geography, such as detours, removing or adding points, and other occasions.

The system is able to handle all of the terrain around campus. Both moving and stationary objects are detected and safely avoided. The system should be able to run for at least 1 hour, under a full load driving full speed. However, the cart is expected to be idle during classes and will only run during passing periods. Estimated to run about 15 minutes per hour. Nighttime operation is not a requirement for this system. The batteries fully charge after a night of charging. Speed of the cart will be variable but it will have maximum limits of 15 mph and will follow all traffic signs and laws.

Background

For a basic overview of our system, we have taken an existing golf cart and added modifications to it, to enable autonomous functionality. The throttle is controlled by a potentiometer, and so a microcontroller is programmed to control the potentiometer over I2C and apply throttle when acceleration is necessary. The braking is controlled by a winch, which is powered by an H-Bridge circuit to apply positive and negative voltage to wind up or unwind the cable pulling at the brake pedal. This H-Bridge circuit is turned on and off by another microcontroller. The steering wheel is turned using a stepper motor, which is controlled by an integrated circuit and Teensy controller. These three motor control systems have controllers that take in input from a general processing unit that does the decision making for the system. The mini-computer gathers the necessary information to make motor control decisions from a suite of sensors including a camera, radar, and GPS module. These modules send data to the mini-computer where the data is analyzed to determine surrounding conditions and what motor controls are necessary to safely navigate the system to the desired destination. All the peripheral devices, less the winch, are powered by a 12V battery that is converted to standard 5V supply via Buck converter.

Design Implementation

System Structure

As an overview of what the overall system should be doing, the block diagram in Figure 1 depicts the level 0 functional decomposition of our overall system. At the most broad viewpoint of our project, we will have a user send in pick-up location and a destination location, and with supply of power to our system, our golf cart should be able to provide transportation for that person while using sensors to take in data of what the golf cart's surroundings are.

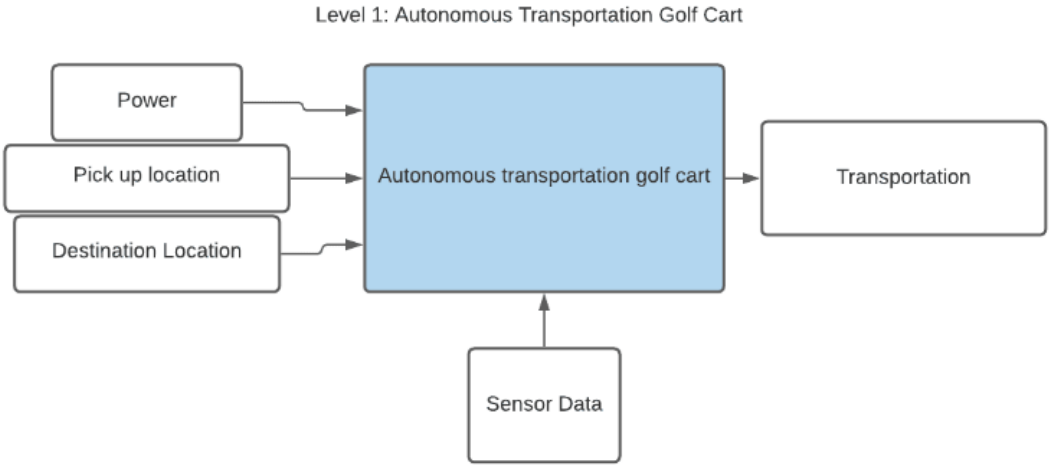


Figure 1: Level 0 Functional Decomposition (Overall System)

The system seen down below in Figure 2, is a breakdown of how the modules inside the autonomous transportation golf cart block from Figure 1, above, are connected. Here we see the same power, pickup location and destination location inputs, as well as sensor inputs along the way, and have the overall output of achieving transportation. This model shows how the project is broken up into separate subsystems for modularity. We take in data from the surroundings, and have sensors connected to microcontrollers or just simply to our mini-computer to process the data and analyze it for the surrounding conditions. With that sensor data, the navigation software system determines how to guide the cart safely to its destination and sends the motor control the necessary information for movement. The microcontrollers on the motor control side, take this data and convert it into any analog signals needed to power the motor control circuitry, finally enabling locomotion.

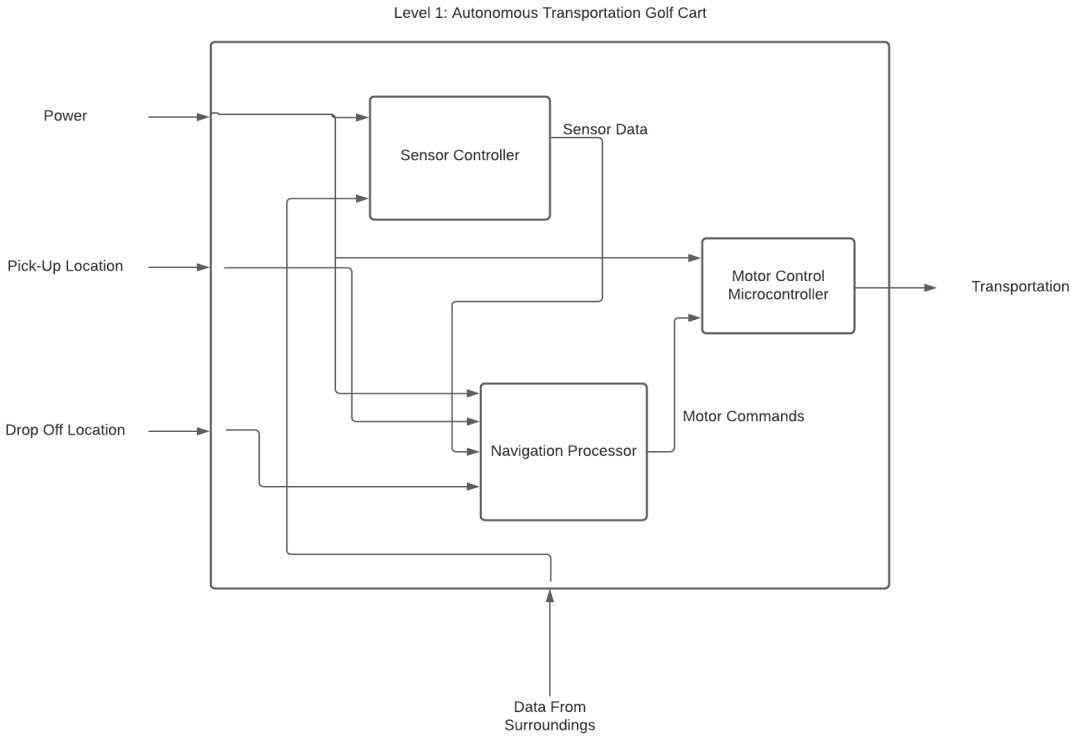


Figure 2: Level 1 Functional Decomposition - Overview

Buck converters step down the voltage of a 12V battery to power the Jetson Nano and a Teensy. The sensors and TM4C microcontrollers get power directly from the Jetson. Four 12V batteries in series power the main motor of the golf cart. Another 12V battery in conjunction with an H bridge circuit controls the winch for the braking system. A breakdown of the hardware layout can be seen in Figure 3.

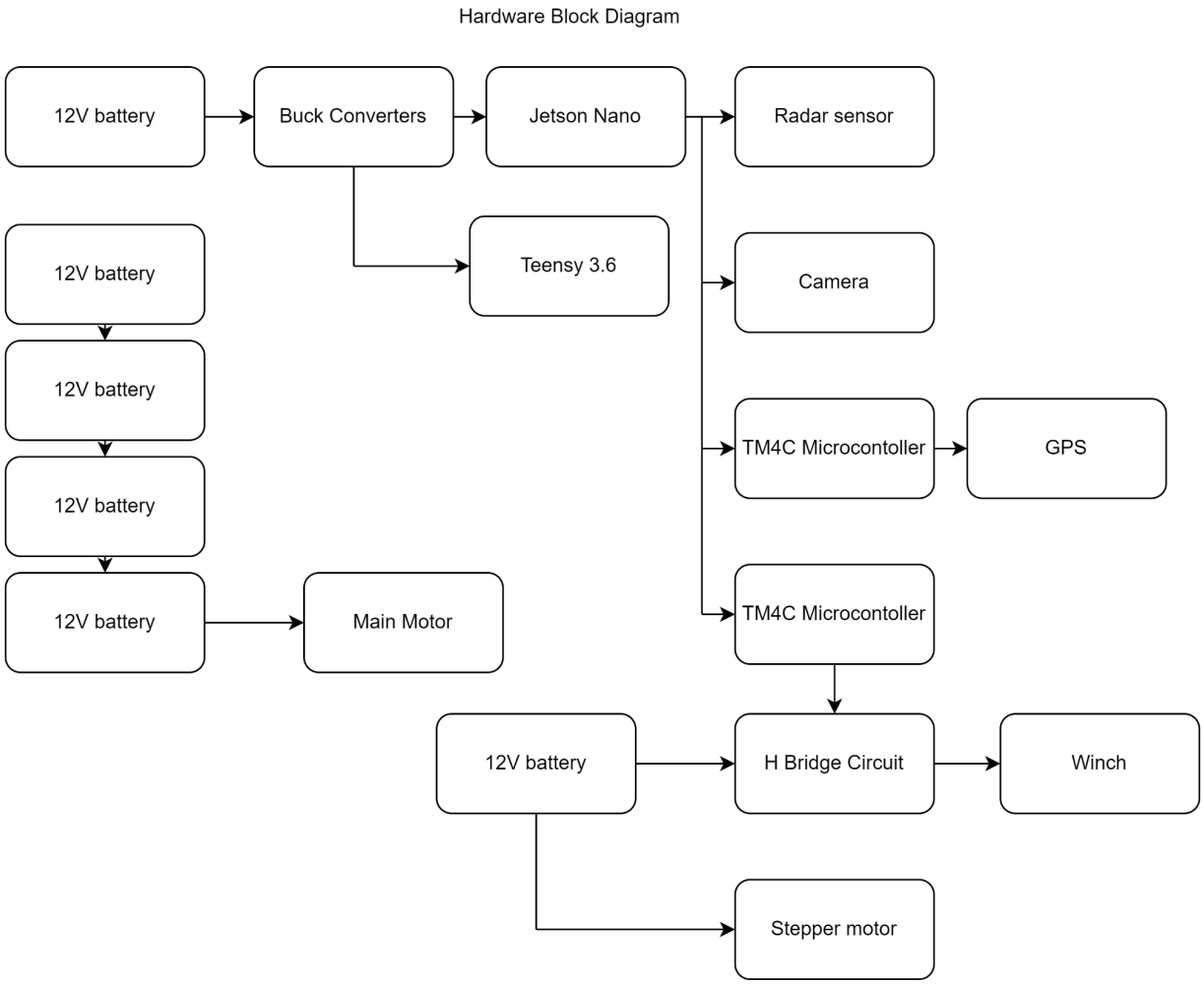


Figure 3: Hardware Block Diagram

An overview of the sensor system's diagram can be found down below, in Figure 4. Here, there are 3 input sensors: camera, radar, and GPS. These three sensor clusters work in tandem to give the navigation system the ability to understand its surroundings and make decisions on how to get to the destination safely. The radar data is processed directly on the chip and sent to the mini-computer through serial communication, where the processing of our camera data occurs. This data is passed to a BT Admin, where a state machine will determine what sort of motor controls are necessary to get the cart from point A to B, without hitting any obstacles, separating out the controls for both acceleration and steering. The acceleration control is split between a microcontroller handling throttle and another controlling braking. The steering control exclusively sends data to a controller that is in control of a stepper motor on the steering wheel itself, to turn the wheel as necessary.

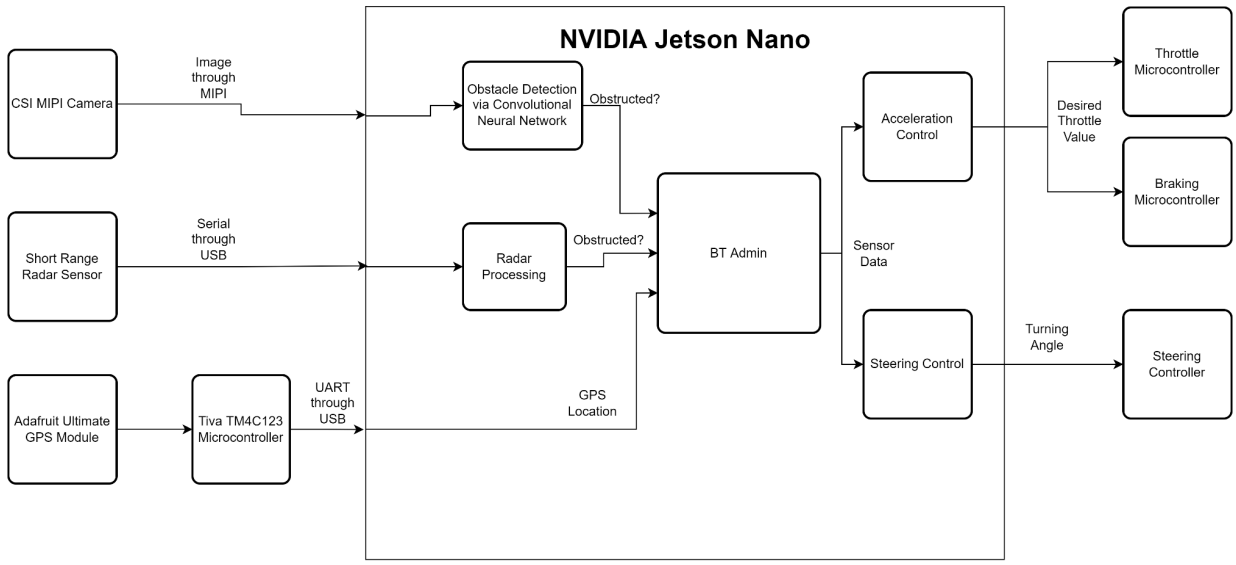


Figure 4: Level 2 Functional Decomposition - Sensor and Data Processing

Hardware Description

In order to determine distances from potential obstacles, we are using an OPS241-B-FM-RP as a short-range radar sensor. Below in Figure 5, is a picture of the radar sensor without any connections or the protective cover. The radar signal processing is done onboard the radar chip and sent using the OmniPreSense AN-10 API for outputting the data for processing. The radar can detect up to 9 objects in its field of view from 1m to 20m away. The data from the API can also be configured to output more specific values for our application.

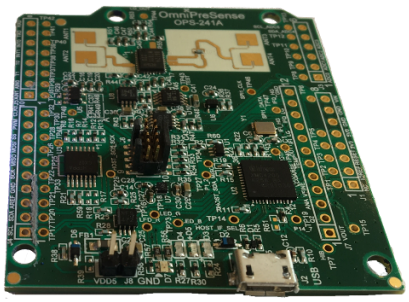


Figure 5: OPS241-B-FM-RP Short Range Radar

Using 4 antennas, the FMCW radar sends out a 24 GHz mm wave and uses the reverberations from objects in the field of view, a 78° horizontal and vertical azimuth, to determine the distance from each object. This short range radar allows our cart to detect objects in all weather conditions, and determine how far we are from obstacles that are identified by the camera vision.

The GPS module utilizes the Adafruit Ultimate GPS Breakout (Figure 6), which is centered around an MTK3339 chipset.

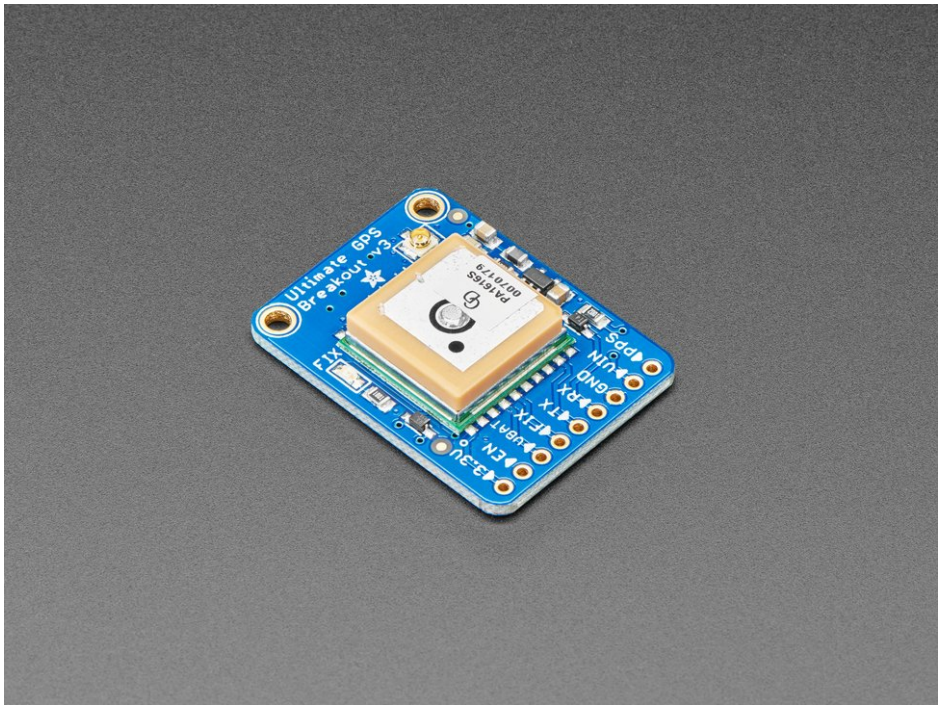


Figure 6: Adafruit Ultimate GPS Breakout

The module queries satellites once per second, though it is capable of updating as fast as 10Hz. It has an accuracy of about 3 meters - standard for GPS technology - and can use up to 22 satellites for tracking. The board draws 20-25mA of power. Data is outputted as NMEA sentences over a 9600-baud UART interface. Being a breakout board, it can be plugged into a microcontroller without any worry about circuit design. For this project, a TM4C123GXL launchpad is used to stand between the GPS chip's UART interface and the Jetson's USB plug.

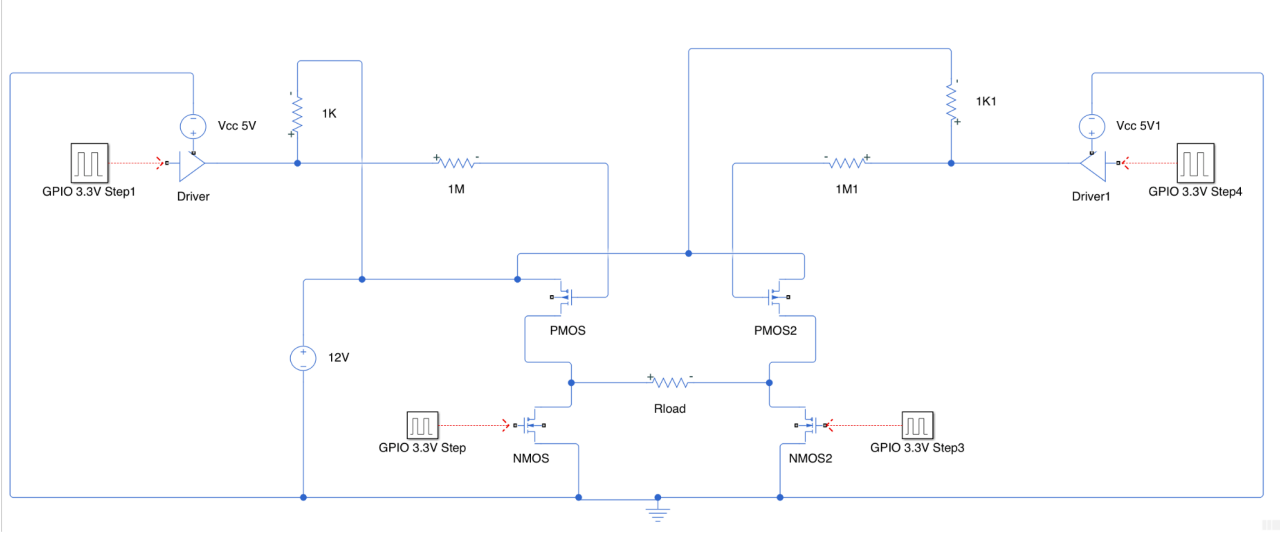


Figure 7: H-bridge circuit Schematic

In Figure 7, we have our schematic of our circuit which controls the direction of which way we would like our winch motor to travel. Applying +12V will have the winch operate clockwise while applying -12V will have it operate counterclockwise. To achieve this, 4 high current rated mosfets were used. Turning on the gates of any P and N channel FETS will cause a complete circuit. These gates were powered by GPIO pins on a TM4C Microcontroller. While testing the mosfets early on it turns out the P-channel fets operated in depletion mode rather than enhancement as specified in the datasheet. Because of this the turn on voltage was actually close to 11 volts rather than 3.3 volts. To overcome this issue, two DM7404N buffers were used. This scaled up the voltage to 12 volts due to it being powered by the same battery used as our power supply. To have a voltage drop between the source and gate of our mosfet we incorporated 1M ohm resistance at the end of the buffer's output to drop the voltage from 12 to 11 volts. Thus being able to activate the gate switching on our GPIO pin of 3.3V. Figure 8 displays the timing diagram for the output voltage when specific FETS are set high and low.

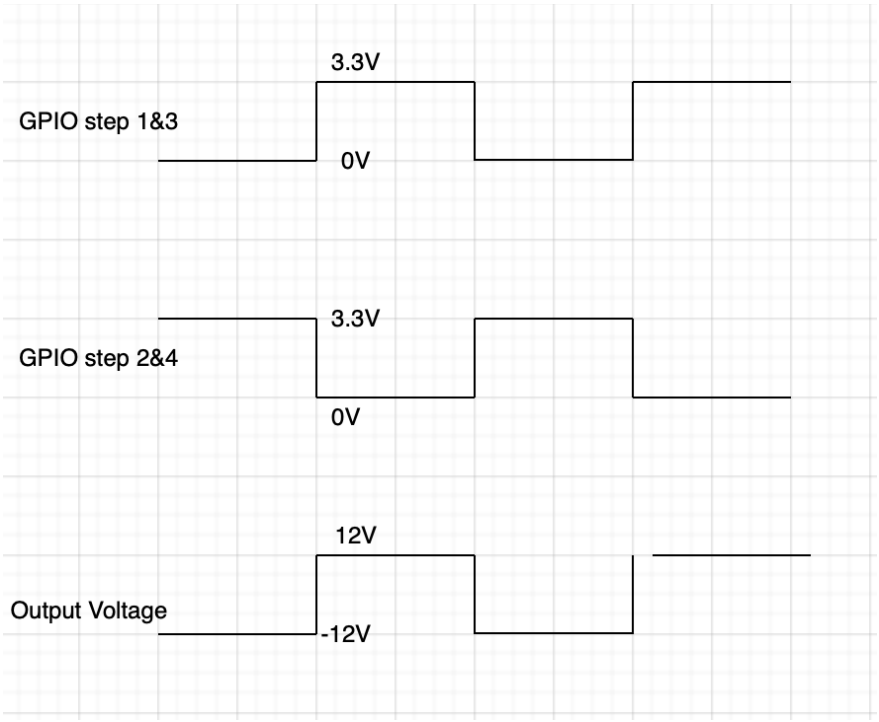


Figure 8: H-Bridge Timing Diagram

For throttle we mimicked the circuit schematic in Figure 9. The output of the throttle pedal was confirmed by using a digital multimeter to read the output of the wiring. This would eventually be replaced with a digital potentiometer as it would act as our resistor.

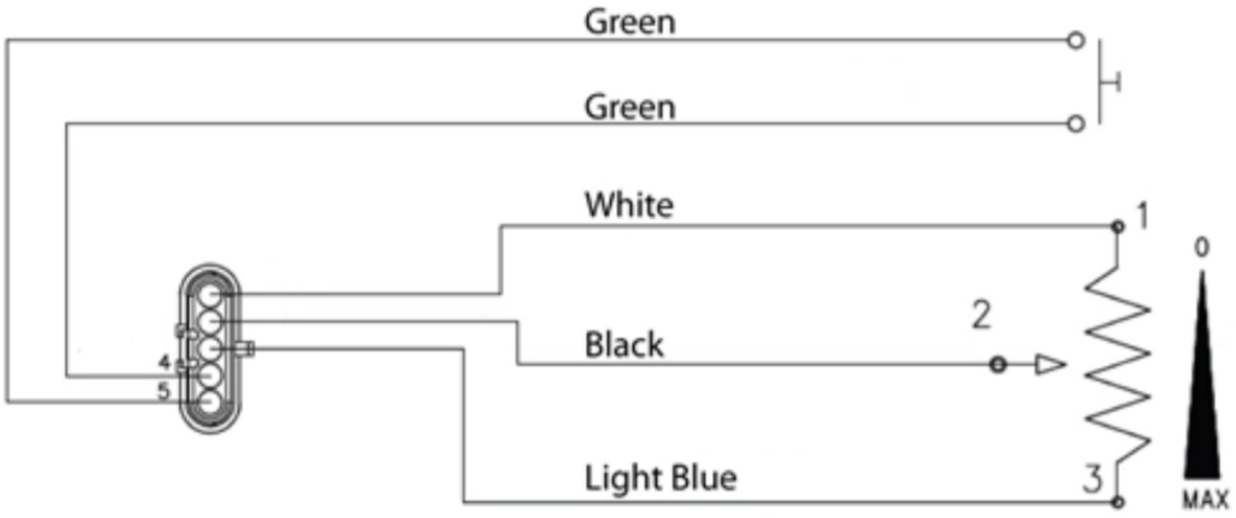


Figure 9: Throttle Circuit Schematic

Running the golf cart requires four of the five batteries we have available. The motor requires 48V. To reach the required voltage, two 12V 55AH batteries are used in series with two 12V 80AH car batteries. Since the cart will only run as long as the lowest rated amp hour battery,

the cart will run for around 40 minutes. The more current drawn, the more the amp hour rating decreases. With up to 55A drawn at full speed, the rating will be close to 35-40AH.

$$2600W/48V = 54.167A \text{ drawn}$$

$$40\ minutes\ of\ operation = 37AH/54.167$$

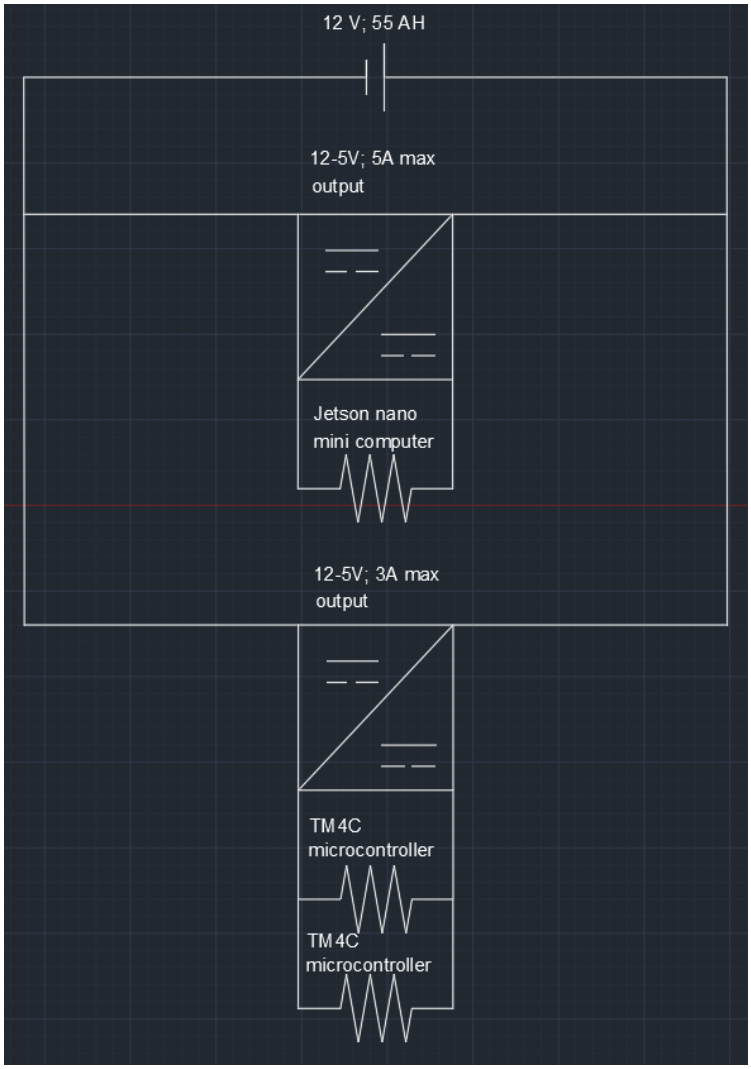


Figure 10: Diagram of Power system for hardware

Figure 10 shows the circuit diagram for the power supply to the hardware. A 12V battery powers the jetson nano and microcontrollers. Two buck converters are used to step down the voltage to 5V. The larger buck converter has a max output of 5A to support the jetson nano which requires up to 4A. The smaller buck converter has a maximum output of 3A which is more than enough for the 150mA of current each microcontroller requires.

With all the devices being powered by the buck converters, the amount of current drawn is 4A maximum. The battery acquired is rated for 20AH. The original plan was to use the marine battery rated for 55AH but that is needed to run the golf cart. With the new battery, the hardware can be powered for about 5 hours.

$$5\ hr\ of\ operation\ =\ 20AH/4A$$

Software Description

The diagram below, in Figure 11, shows the flow used for loading, training and saving a convolutional neural network for image classification. A neural network was used, because of its compatibility with images, and strengths in image classification. A Google Colab environment was used, with the latest tensorflow 2.8, tensorflow_hub 0.8, GPUs, ample memory and Python 3.7 available by default. From there it is a choice in what the base model is, Tensorflow and the built in Keras library has a plethora of models to choose from, but it was found that a smaller model was best as overall application is mobile on such a small mini-computer. Data was collected via GoPro attached to a golf cart, as it was driven around all of campus in multiple trips. The data included a variation of surroundings and situations, as footage was taken during passing periods and during the middle of class to ensure enough data for both obstructed and unobstructed paths was available. Once the footage was gathered, it was processed, as still images were captured from the video, and classified into folders of obstructed and unobstructed. This data was loaded into the virtual environment and used to train the selected model, tweaking parameters as necessary to fine-tune and improve accuracy. Once the model is trained, it is validated with a separate set of data, and also tested for basic classification accuracy with a chosen series of pictures. The model is saved and exported, for further use on the NVIDIA Jetson Nano.

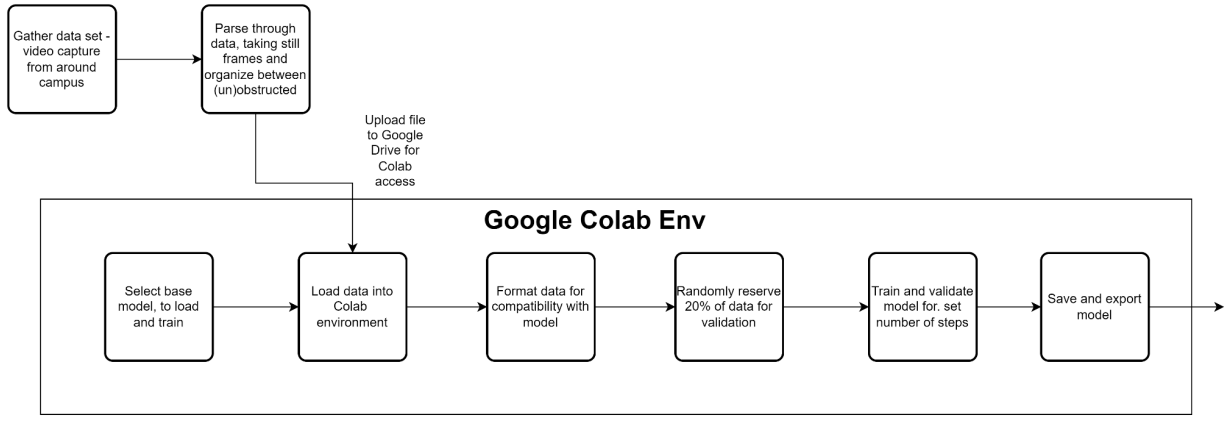


Figure 11: Obstacle Detection Model Training with Google Colab

Once the model for obstacle detection is trained, and the model has been loaded onto the Jetson, the overall system is able to classify obstructed and unobstructed paths, if the correct hardware requirements are met. In order to load the classification model onto a mini-computer, it is required to have available memory greater than 1.5GB, and to run the MIPI camera in parallel with the model loaded, the power supply should be capable of delivering >2A. With these requirements fulfilled, the software structure of the image classification module can be seen down below in Figure 12, where the system boots up and loads the necessary library modules, the camera is loaded as well and begins feeding images through the model for predictions. These predictions are then passed through a network port and delivered to the overall system administrator, which determines what to do with this obstructed/unobstructed data. Necessary library installations for this module include: cv2, tensorflow, tensorflow_hub, and time.

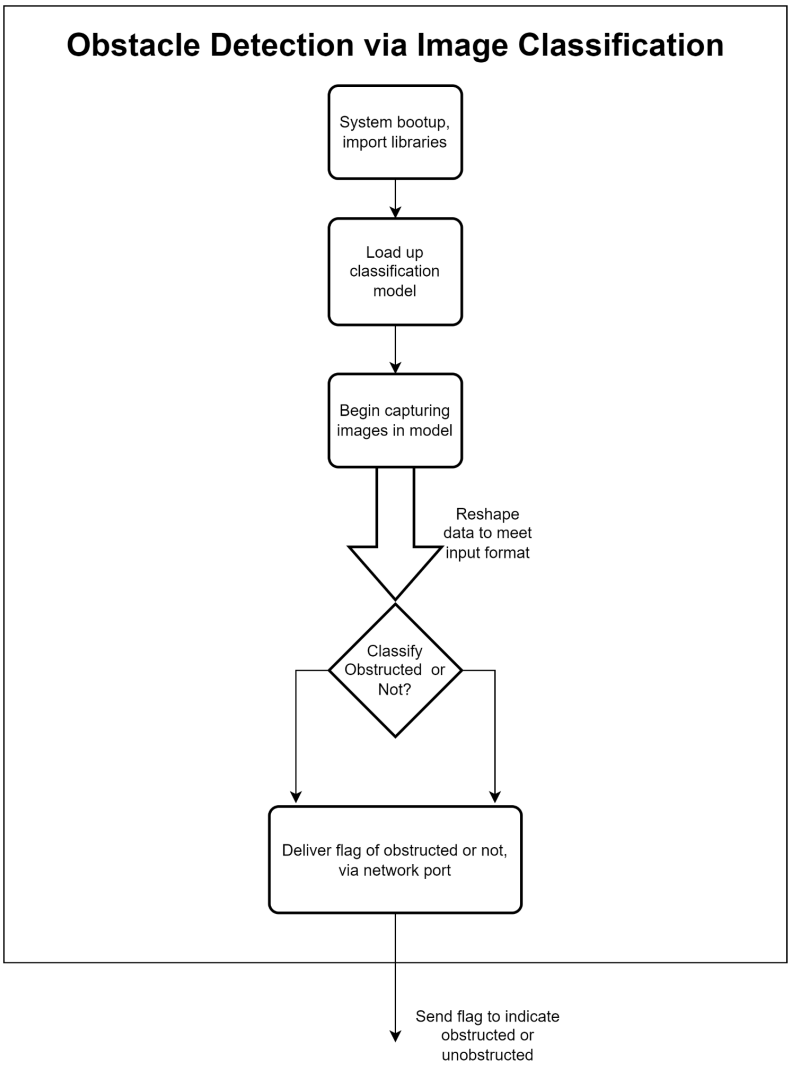


Figure 12: Level 3 Functional Decomposition - Image Classification Breakdown

Without a device with memory >2GB, the model will fail to load onto the Jetson. In order to bypass this issue, the model needs to be processed further and converted into a TFLite model, where it saves all the nodes and information in a linear and more compact structure. The TFLite model comes packed into TensorFlow's standard library, but requires unique functions to load and execute the tensors in this new form. The model is loaded into an interpreter, and images are fed into that with outputs being probabilities of the image classified as obstructed or unobstructed, with the one with higher probability being the prediction it decides on.

For throttle control, code was also developed for I2C communication between a microcontroller and a digital potentiometer to allow for variable resistance based on the output from the throttle algorithms on the jetson. This code would take an input from the UART/USB communication on the microcontroller and the microcontroller would change the resistance of the throttle to increase or decrease acceleration. Though, this was not fully implemented in our final product.

Development was also done into what a future lane detection algorithm would be. Using openCV, a canny edge filter is applied with a general region of interest and Hough Lines Principle to create an idea of the lines in the golf carts path. Once lines are detected that represent what the bounds of the pathway look like, calculations in terms of the angle between the two lines can be made to determine if you are going straight down the path, or headed towards one of the sides.

The radar software takes in data from the radar sensor's API into a python program for extra post-processing. It configures the serial port and gives commands to the radar chip to output more specific values better for our application. The program sets the radar to output distance from 3 objects in the FOV every 100ms. The minimum distance detected is 1.2m from the radar and any data closer is filtered out to get rid of feedback from the cart. The radar must fill an array buffer of 3 values reported within 3m before the status is changed from obstructed to unobstructed or vice versa. This ensures there aren't false positives or negatives, and over adjustment of the throttle or brake.

The cart's navigation is based on GPS coordinates relative to a nodemap of campus. The UOP campus is represented internally as a collection of nodes and edges placed over footpaths, where each node has a real-world coordinate.

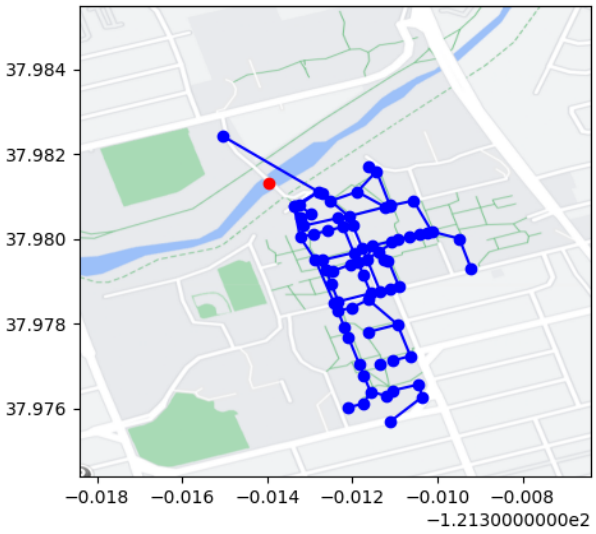


Figure 13: Campus map

An algorithm determines the most efficient path between two nodes and follows the cart's travel along that path. As it approaches an intermediary node, the cart will determine its distance to that node. When it gets close enough it'll begin to search for the next node. NMEA sentences from the GPS are processed by the microcontroller, which sends coordinate data to the Jetson. The GPS readings are typically more than a meter off, so the accuracy is not enough to assure that the golf cart stays on a path.

The various submodules communicate with a program known as the BT Admin. The admin uses data from the input modules to determine the cart's actions, and sends the necessary data to the output modules (Figure 14).

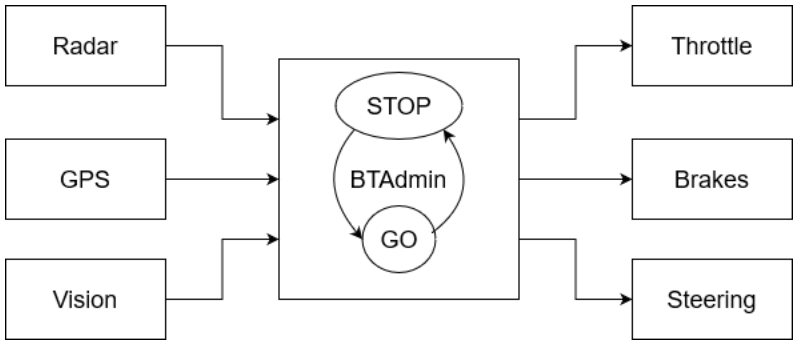


Figure 14: BTAdmin data flow and state machine

To communicate with each submodule, the admin utilizes Unix Domain Sockets, acting as a sort of web server for each submodule to connect to. These update a shared dictionary of booleans which is analyzed by a simple state machine. This machine can either STOP or GO,

and will send values to the brakes accordingly. If all the input modules are giving positive signals, then the state will be GO. In all other cases, it will be STOP. The admin was designed this way to ensure that unexpected behavior is more likely to make the cart stop than to send it out of control. The limited decision-making complexity of the cart, however, was due to time constraints.

To enable the GPIO's needed to turn on certain FETS, a program was written in code composer studio which would enable the pinouts needed. Number 5 in the appendix shows the code flow for this task.

Additionally software has been developed to create smooth/gradual throttle and steering, although this was not actually implemented onto the Jetson, as the cart was not running to fine-tune this more complex motor control. To test basic functionality, we used a more simple stop and go implementation to cut out any complications that could have corrupted the basic functionality tests we were trying to achieve. The gradual brake and throttle algorithm developed was a formula given both sensor inputs and time into the equation and then incrementally changes the throttle level. There are limits on the equation of -50 to 50, where -50 represents maximum brake applied, 0 means no brake or throttle applied (coast), and 50 means to apply full throttle. Every number between -50 to 0 and 0 to 50 represents a partial amount of either brake or throttle applied, respectively. The equation is as follows:

$$Throttle(n) = Throttle(n-1) - 10R - 3O - 2S - 2D + U$$

n: time, R: radar, O: image processing obstructed, S: stop sign approach, D: destination approach, U: image processing unobstructed

The equation for steering is as follows:

$$Steering(n) = Rsin(n-N) - Lsin(n-N)$$

n: current time, N: time at start of turn, R: right turn bool, L: left turn bool

Table 1 describes the software written for this project. Programs on the Jetson can be found in the *budgetTesla/* folder, found in the *bt* user's home directory. Others are programmed to the microcontroller.

Program	**Purpose**
geolocation_py	GPS module.
macronav_playground	GPS testing scripts and prototypes.
btadmin	Administrative module, testing utilities, interprocess communications helper script.

compVision	Computer vision module.
radar	Radar processing module.
brake_control	Translate commands from Jetson into voltages for braking H-bridge.
gps_processing	Process NMEA sentences and forward coordinates to Jetson.

Table 1: Software listing

Verification Procedure and Evaluation

In order to verify the parts received, tests were performed to ensure that they work as intended. For the mosfets a simple voltage difference test was done between the drain and source of the FET to see if there was a voltage difference. This shows that the gate is open and not always conducting. Table 2 shows the results of this. Next, we connected each FET to our source of 12 volts and measured the resistance between the source and drain. For our N channels we noticed that as we ramped our gate voltage up to close to 3 volts, our resistance was approximately 0 ohms, meaning our gate was conducting. However, when the same test was done to the P channels we had to ramp up the voltage to approximately 10 volts to finally see the resistance dropping. Due to this, additional circuitry needed to be added. Initial ideas were op-amps or a boost converter to increase voltage. In the end a buffer driver was used. Link 1 shows the result of this buffer boosting our 3.3v output to around 11 volts.

Link 1: https://youtube.com/shorts/UZYnDXrqOes?feature=share

Once we knew individual parts worked in our circuit, we connected them together as well as our microcontroller code. Link 2 shows the results of our braking system. In this test run the braking was on for a short period of time to show proof of concept. The timing can easily be increased by adjusting our delay function in our code shown in number 5 in the appendix.

Link 2: https://youtube.com/shorts/GTcQ7MSOgOQ?feature=share

Mosfet type	N-channel(V)	N-channel(V)	N-channel(V)
+ source - drain	.474	.48	.48
- source + drain	0	0	0

Mosfet type	P-channel(V)	P-channel(V)	P-channel(V)
+ source - drain	0	0	0
- source + drain	.487	.496	.481

Table 2: MOSFET Voltage difference between Drain and Source

In order to test the output of the throttle we connected a DMM to the wiring inside the throttle sensor. Link 3 shows a video of the output. After we confirmed resistance was the output we split open the wire(orange) that sends out the resistance and replaced it with a variable

resistor. Unfortunately the variable resistor connected to the input of the ECU produced no throttle.

Link 3: https://youtube.com/shorts/GoSoN-Cnn0Y?feature=share

For radar testing and verification, we used the serial program PuTTY to read the radar data into a laptop to check for biases or inaccuracies in our radar. After confirming functionality indoors on a laptop, testing was continued on several different surfaces around campus to ensure it would read all kinds of objects. Cars and metal objects had the best reflection, therefore also the most accurate readings from the radar. Trees and people were detected but could not consistently be read at a distance more than 10m away. To ensure our radar would work when moving, the radar sensor was tested while being driven around on a golf cart. Figure 15 below show what a GoPro was seeing next to what the output in our serial terminal on a laptop.

Figure 15: Radar Testing on Golf Cart

In Figure 15 above, the serial output is on the left and the camera is on the right. In the orange box, the time since the radar was turned on is displayed in seconds. The blue boxes exhibit the group of three walking in front of the cart as they are getting closer each fraction of a second, while the yellow boxes are discrepancies from what was there and what the radar was reading. This is most likely caused by the radar being unmounted when driving around and getting feedback from the cart or the ground and shows extra processing needs to be done. Figure 16 below shows additional radar data collected and making sure we are getting additional good data. The colors correlate to what the radar is seeing and what objects are there in the camera's view.

Figure 16: Additional Radar Testing

Due to feedback and undetermined readings on the radar sensor, additional processing has to be done, but we know that the radar is seeing and reading people and objects while in motion. To solve this problem, we filter the outputs from the radar to better fit our application. The output from the radar code just needs to show if the cart is obstructed or not. The program reads the input from the radar sensor, splits it into individual numbers, and determines if the cart is obstructed by checking if someone is within 3m. A demo of the functional radar is posted on youtube using the link below. In the video, you can see the radar reads obstructed when there is a person in front. Then, once the subject walks out of range it clears the obstruction value. When off to the side, the radar does not read until the subject is in front of the sensor and clears it again after. Finally, the subject, from out of range, walks into obstruction range and the radar properly detects an obstacle.

Link 4: Short Range Radar Demo - UOP 2022

In order to verify the functionality of the image classification network, a basic test run of inputting a known image where the path is known to be obstructed is input, followed by the input of an image of the same path being unobstructed. If the model is able to differentiate the presence of obstacles in the two samples, then we can verify that the model is making predictions accurately. Because the overall image is the same for both, the model will have to isolate out the obstacle itself to make a prediction on the obstruction rather than have some sort of background distraction cause a false-positive result for an obstructed image. This was tested in the Google Colab environment first, and then the model was transferred over to our system hardware (the Jetson), and the test was rerun to verify that the model transfers over accurately as well. As can be seen down below in Figure 17, the obstructed image is accurately classified and so is the unobstructed. Below that, in Figure 18, you can see the actual output of the obstructed and unobstructed image. As mentioned earlier, both images are of the same background, but one

clearly has people in the view, blocking the path. In the terminal output, you can see the same predicted labels seen in the Colab environment, verifying that when the model is transferred over to the Jetson, the data structure remains valid and intact. A video of the system working live can be seen down below, as Link 5, where the object detection is working to classify live and output its predictions into the command terminal. It should be noted that the video capture is indoors whereas the dataset the model was trained on was from footage exclusively outside, and so there is some inaccuracy expected. This note aside, the accuracy of the classification still seems to be fairly accurate at seeing the human in the frame as an obstacle.

Link 5: https://youtu.be/ENJZwubd0_o

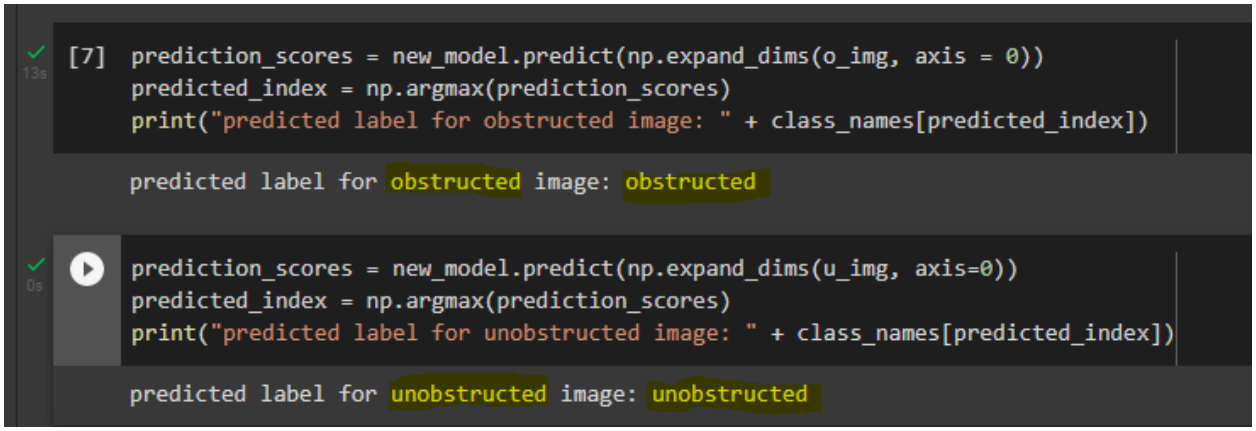


Figure 17: Image Classification Verification in Google Colab

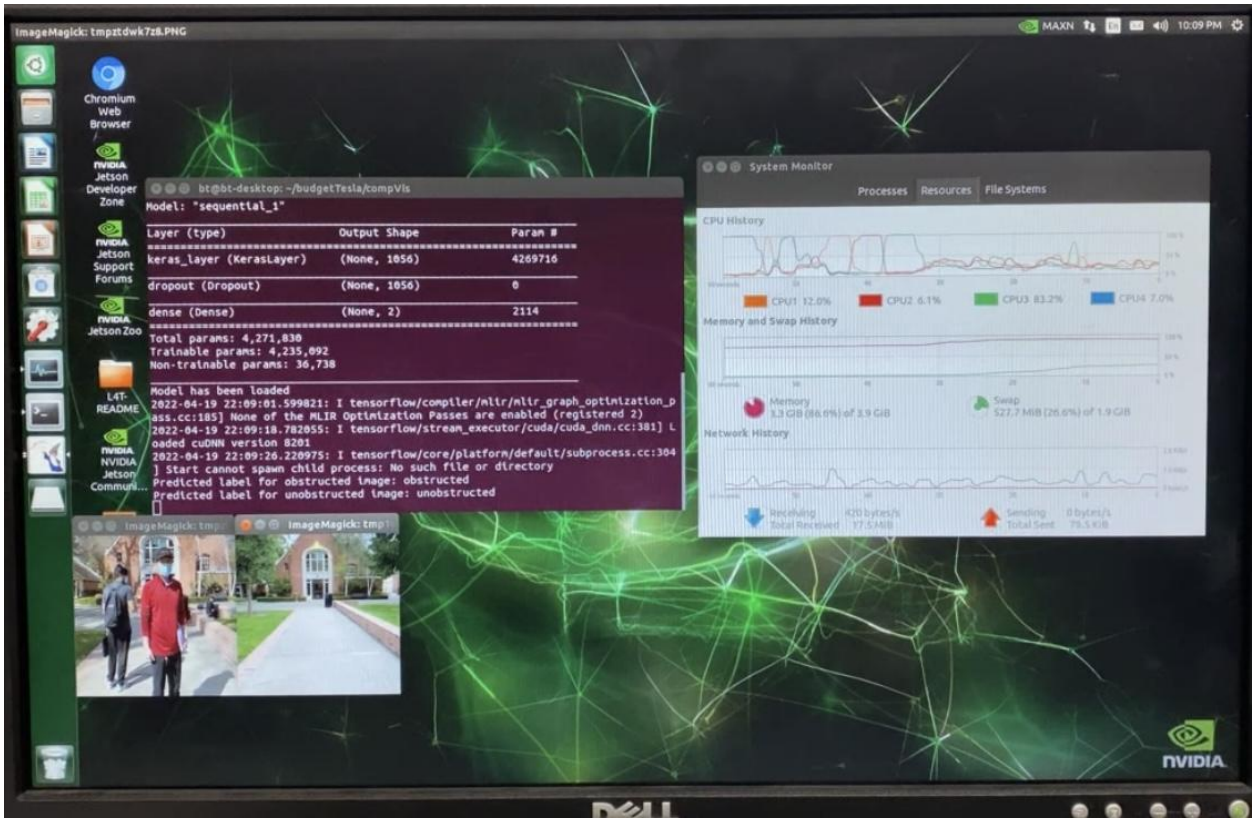


Figure 18: Image Classification Verification on NVIDIA Jetson Nano 4GB

The overall flow of the image processing was verified for functionality as well. As described earlier in Figure 12, for the Level 3 Functional Decomposition, the Jetson must boot up the libraries, load the image classification model, camera and begin the input stream of video capture. Once the input model is loaded and the camera is capturing, the model immediately begins outputting classifications. The full system running in a headless mode, which is the mode it runs in when implemented with the actual system can be seen down below in Link 6. Again, this video should be watched with the disclaimer that the output is subject to inaccuracies because the test is conducted indoors and in a small, cluttered room.

Link 6: https://youtu.be/Wl0484pz3nI

When these two tests are placed together, we have verified both the accuracy of the model in classifying obstacles and the transferability of the system to run on the Jetson Nano without any other dependencies.

Additionally, after the Jetson Nano 4GB was burned, a proof of concept implementation to get the system running purely on laptop and not on mobile mini-computer was developed. To do so, the same Colab environment was used, but because Google Colab environment is a cloud based computing service, it did not have access to local peripherals like the host webcam. In order to get the system running on a laptop, JavaScript was used to create a live video stream,

using a laptop's webcam as the input. Link 7 provides a video of the system working in live time, to classify the pathway outside of Khoury. As seen, the path is initially unobstructed as there are no obstacles in the way, at this startup there is some time needed for the model to settle itself, as the laptop was being balanced to keep the path in frame while having the image as level as possible. Then the model locks on, and displays unobstructed. From then, a person walks into the middle of the pathway, and the model instantly begins to detect the path is obstructed. Once the person walks off the path and out of the frame the model reverts back to its classification of unobstructed.

Link 7: https://youtu.be/ojc0MZv4JUw

Since the 4GB Jetson burned, different workarounds were explored to try to figure out a way to still salvage the image processing module of the project. After a successful conversion of the TensorFlow model to a TensorFlowLite model, and a new script written to run a TFLite interpreter rather than standard model, allowed us to achieve similar results as seen above, but on the more memory limited Jetson. It is important to note, that when the conversion occurred to a TFLite model, there were some drawbacks from the same accuracy we had seen earlier on the standard model, but still we had basic functionality on the smaller package. A test of this system can be seen in Link 8, down below, where the whole system is running headless on the Jetson 2GB itself and in a portable real-life simulation. The camera is hooked up to the Jetson which is being powered by a portable battery that is stepped down through a buck converter. Technically the system wouldn't need a laptop hooked up to it, but for demonstration purposes of showing what the output of the image processing actually is, there is a terminal open to show that output. At first, the video shows the terminal printed with "Predicted Label: Unobstructed" but as a person steps into frame it switches to obstructed, and then as they leave it returns to unobstructed. This is repeated multiple times to prove the test was not some sort of coincidence.

Link 8: https://youtu.be/bIDPd2QDNHE

To test the GPS module, routes were calculated across campus and the GPS chip was walked along these routes, using a laptop to power and process the data. Figure 19 shows the results of one such trip.

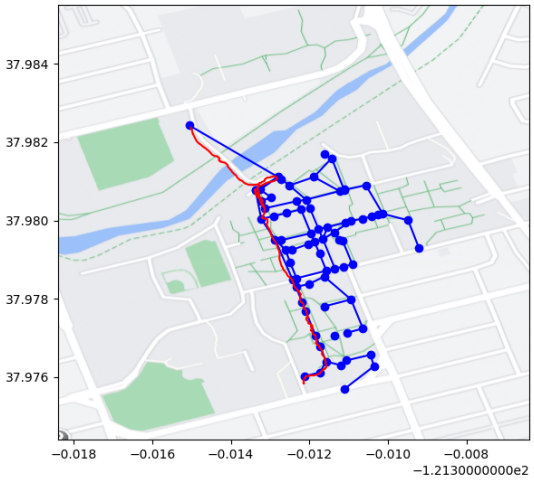


Figure 19: In red, path taken from Olson to the mailroom

Further testing was performed to roughly judge the accuracy of the GPS chip. The chip was walked in clearly-shaped paths along campus and the data output was compared to the shape. A circle, square, and 90-degree turn were tested, and the GPS data for all of these tests was significantly different than the expected result. Figure 20 is one such test, for a 4-meter square walked on the UC lawn during a clear day.

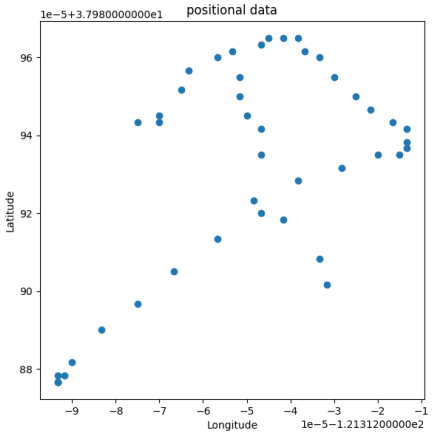


Figure 20: GPS trace of a square path 4 meters wide.

The data hardly resembles a square and does not end where it begins. From this, it was determined that the GPS's inaccuracies are significant enough that it cannot be used for pathkeeping, and that more accurate testing is not necessary to support this conclusion.

A basic functionality simulation of the throttle equation is shown below in Table 3, as input sequences for radar detection, image processing detection, stop sign approaching or destination approaching were programmed. A test was created where simulated inputs for each of these signals is created to mimic what the golf cart would see in real time, with most of the time the path being unobstructed so the throttle beginning to accelerate the cart, then occasionally getting signals from radar, image processing or GPS regarding an obstruction or key destination approaching.

Clock	Output	Radar	Unobstructed	Stop Sign	Destination	Unobstructed
Clock count: 0	Throttle Val: 1	0	0	0	0	1
Clock count: 1	Throttle Val: 2	0	0	0	0	1
Clock count: 2	Throttle Val: 3	0	0	0	0	1
Clock count: 3	Throttle Val: 4	0	0	0	0	1
Clock count: 4	Throttle Val: 5	0	0	0	0	1
Clock count: 5	Throttle Val: 6	0	0	0	0	1
Clock count: 6	Throttle Val: 7	0	0	0	0	1
Clock count: 7	Throttle Val: 8	0	0	0	0	1
Clock count: 8	Throttle Val: 9	0	0	0	0	1
Clock count: 9	Throttle Val: 10	0	0	0	0	1
Clock count: 10	Throttle Val: 11	0	0	0	0	1
Clock count: 11	Throttle Val: 12	0	0	0	0	1
Clock count: 12	Throttle Val: 13	0	0	0	0	1
Clock count: 13	Throttle Val: 14	0	0	0	0	1
Clock count: 14	Throttle Val: 15	0	0	0	0	1
Clock count: 15	Throttle Val: 16	0	0	0	0	1
Clock count: 16	Throttle Val: 17	0	0	0	0	1
Clock count: 17	Throttle Val: 18	0	0	0	0	1

Clock count: 18	Throttle Val: 19	0	0	0	0	1
Clock count: 19	Throttle Val: 20	0	0	0	0	1
Clock count: 20	Throttle Val: 17	0	1	0	0	0
Clock count: 21	Throttle Val: 14	0	1	0	0	0
Clock count: 22	Throttle Val: 11	0	1	0	0	0
Clock count: 23	Throttle Val: 8	0	1	0	0	0
Clock count: 24	Throttle Val: 5	0	1	0	0	0
Clock count: 25	Throttle Val: 2	0	1	0	0	0
Clock count: 26	Throttle Val: -1	0	1	0	0	0
Clock count: 27	Throttle Val: -4	0	1	0	0	0
Clock count: 28	Throttle Val: -7	0	1	0	0	0
Clock count: 29	Throttle Val: -10	0	1	0	0	0
Clock count: 30	Throttle Val: -9	0	0	0	0	1
Clock count: 31	Throttle Val: -8	0	0	0	0	1
Clock count: 32	Throttle Val: -7	0	0	0	0	1
Clock count: 33	Throttle Val: -6	0	0	0	0	1
Clock count: 34	Throttle Val: -5	0	0	0	0	1
Clock count: 35	Throttle Val: -4	0	0	0	0	1
Clock count: 36	Throttle Val: -3	0	0	0	0	1
Clock count: 37	Throttle Val: -2	0	0	0	0	1
Clock count: 38	Throttle Val: -1	0	0	0	0	1
Clock count: 39	Throttle Val: 0	0	0	0	0	1
Clock count: 40	Throttle Val: 1	0	0	0	0	1
Clock count: 41	Throttle Val: 2	0	0	0	0	1
Clock count: 42	Throttle Val: 3	0	0	0	0	1

Clock count: 43	Throttle Val: 4	0	0	0	0	1
Clock count: 44	Throttle Val: 5	0	0	0	0	1
Clock count: 45	Throttle Val: 6	0	0	0	0	1
Clock count: 46	Throttle Val: 7	0	0	0	0	1
Clock count: 47	Throttle Val: 8	0	0	0	0	1
Clock count: 48	Throttle Val: 9	0	0	0	0	1
Clock count: 49	Throttle Val: 10	0	0	0	0	1
Clock count: 50	Throttle Val: 1	1	0	0	0	1
Clock count: 51	Throttle Val: -12	1	1	0	0	0
Clock count: 52	Throttle Val: -25	1	1	0	0	0
Clock count: 53	Throttle Val: -24	0	0	0	0	1
Clock count: 54	Throttle Val: -23	0	0	0	0	1
Clock count: 55	Throttle Val: -22	0	0	0	0	1
Clock count: 56	Throttle Val: -21	0	0	0	0	1
Clock count: 57	Throttle Val: -20	0	0	0	0	1
Clock count: 58	Throttle Val: -19	0	0	0	0	1
Clock count: 59	Throttle Val: -18	0	0	0	0	1
Clock count: 60	Throttle Val: -19	0	0	0	0	1
Clock count: 61	Throttle Val: -20	0	0	0	0	1
Clock count: 62	Throttle Val: -21	0	0	1	0	1
Clock count: 63	Throttle Val: -20	0	0	1	0	1
Clock count: 64	Throttle Val: -19	0	0	1	0	1
Clock count: 65	Throttle Val: -18	0	0	0	0	1
Clock count: 66	Throttle Val: -17	0	0	0	0	1
Clock count: 67	Throttle Val: -16	0	0	0	0	1

Clock count: 68	Throttle Val: -15	0	0	0	0	1
Clock count: 69	Throttle Val: -14	0	0	0	0	1
Clock count: 70	Throttle Val: -13	0	0	0	0	1
Clock count: 71	Throttle Val: -12	0	0	0	0	1
Clock count: 72	Throttle Val: -11	0	0	0	0	1
Clock count: 73	Throttle Val: -10	0	0	0	0	1
Clock count: 74	Throttle Val: -9	0	0	0	0	1
Clock count: 75	Throttle Val: -8	0	0	0	0	1
Clock count: 76	Throttle Val: -7	0	0	0	0	1
Clock count: 77	Throttle Val: -6	0	0	0	0	1
Clock count: 78	Throttle Val: -5	0	0	0	0	1
Clock count: 79	Throttle Val: -4	0	0	0	0	1
Clock count: 80	Throttle Val: -5	0	0	0	1	1
Clock count: 81	Throttle Val: -6	0	0	0	1	1
Clock count: 82	Throttle Val: -7	0	0	0	1	1
Clock count: 83	Throttle Val: -6	0	0	0	0	1
Clock count: 84	Throttle Val: -5	0	0	0	0	1
Clock count: 85	Throttle Val: -4	0	0	0	0	1
Clock count: 86	Throttle Val: -3	0	0	0	0	1
Clock count: 87	Throttle Val: -2	0	0	0	0	1
Clock count: 88	Throttle Val: -1	0	0	0	0	1
Clock count: 89	Throttle Val: 0	0	0	0	0	1
Clock count: 90	Throttle Val: 1	0	0	0	0	1
Clock count: 91	Throttle Val: 2	0	0	0	0	1
Clock count: 92	Throttle Val: 3	0	0	0	0	1

Clock count: 93	Throttle Val: 4	0	0	0	0	1
Clock count: 94	Throttle Val: 5	0	0	0	0	1
Clock count: 95	Throttle Val: 6	0	0	0	0	1
Clock count: 96	Throttle Val: -3	1	0	0	0	1
Clock count: 97	Throttle Val: -18	1	1	1	0	0
Clock count: 98	Throttle Val: -35	1	1	1	1	0
Clock count: 99	Throttle Val: -50	1	1	0	1	0
Clock count: 100	Throttle Val: -50	1	1	1	1	0

Table 3: Throttle Control Simulation

Results for the lane detection algorithm development can be seen down below in Figure 21, where we have the original image on the left, the canny edge filter applied in the middle and the region of interest detected on the right. The next steps would be to apply the Hough Lines Principle and determine the angle between the two paths, then make micro-adjustments to correct the course and maintain the path.

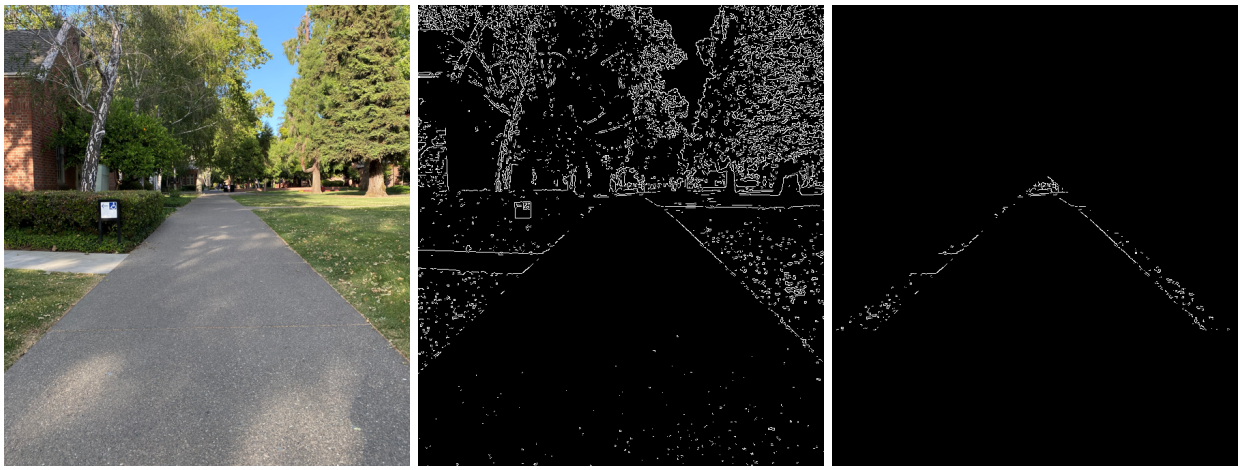


Figure 21: Lane Detection OpenCV

To verify the buck converters work properly. The buck converters were connected to a power supply and a resistor. The potentiometer was adjusted until the output voltage reached the desired 5V output from a 12V input. The circuit was designed where both buck converters are in parallel to the battery. Then the loads are in parallel to the buck converter. 1k resistors were used

to represent the load since the test was to verify that each load in parallel would have 5V across. The test was successful and I recorded the results seen in Link 9.

Link 9: https://youtube.com/shorts/m8zKVcM03Kc?feature=share

To verify the output current, a variable resistor was needed. Using a fixed resistor would burn out the load because too much current would be drawn. The variable resistor can take a lot of current and can be set to the proper resistance to test for the current needed. After confirming that the large buck converter can supply 4.5A and the small one can supply 2.5A, the hardware was tested. The indicator light shows that enough current is flowing at the correct voltage. The jetson nano was run on the desktop and it did not crash. The power test can be seen in Link 10.

Link 10: https://youtube.com/shorts/la5Pda1aNJQ

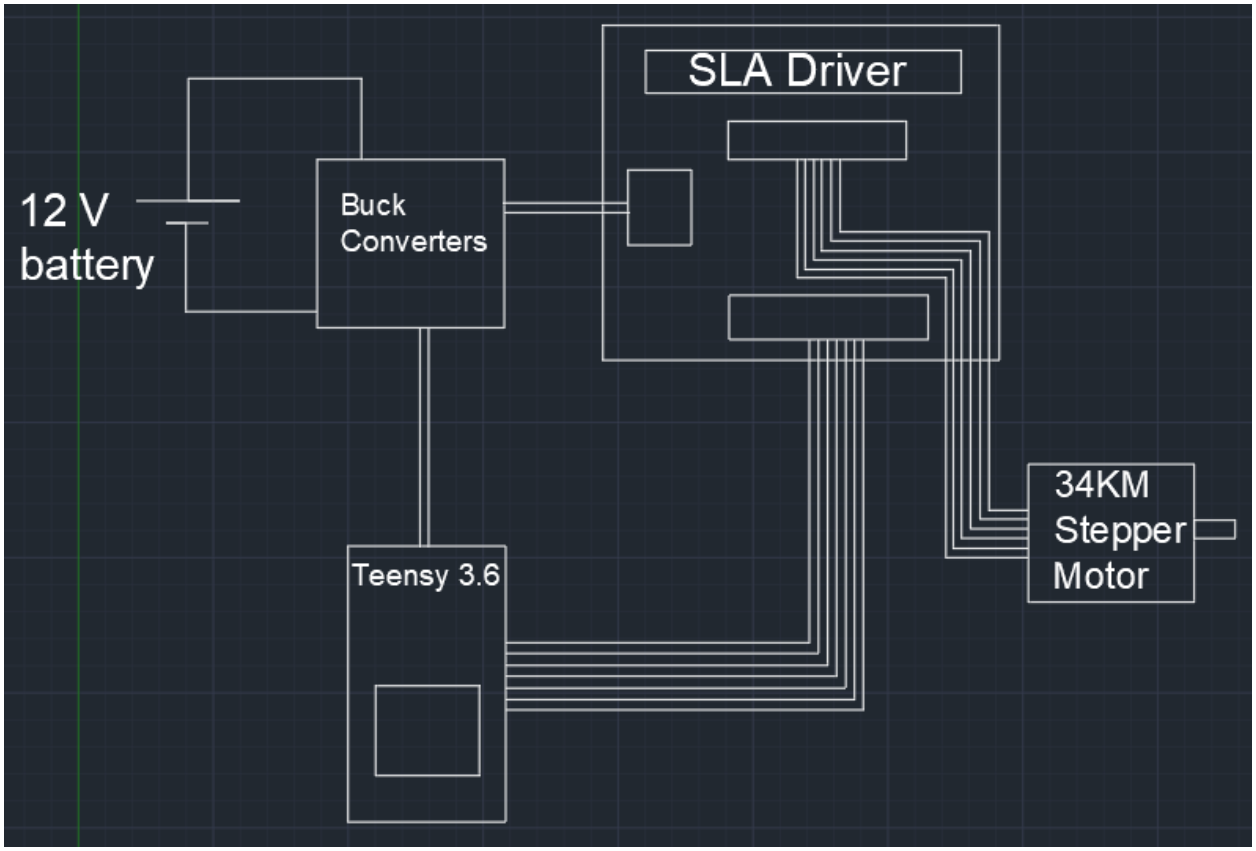


Figure 22: Steering System Schematic

Figure 22 is a schematic diagram of the steering system. The steering system works through a teensy 3.6 running Arduino code. The teensy communicates with a SLA driver that is connected to the stepper motor. The stepper motor turns a gear that turns a larger gear. The larger gear has a bolt going through it that connects to the shaft of the steering wheel. However, the

increased load of connecting the steering wheel causes problems with the system. The stepper motor does not have enough torque to turn the steering wheel because it is limited by current. The driver has a maximum output current of 2.5-3A. The stepper motor needs more current than that to turn the wheel. Even with the golf cart propped up on a jack to remove friction between the wheel and the floor. For demonstration purposes, the steering wheel is disconnected to show that the stepper motor can be controlled through the Teensy. In the video of link 11, the gear changes directions. The Arduino code can also be programmed to change the speed.

Link 11: https://www.youtube.com/watch?v=VL8IOvUzgec

Analysis

The results show that we have sufficient braking to stop the golf cart from its top speed of 15 MPH. The specification we did not meet was that our initial design included using a PWM output from our microcontroller to adjust the speed of the motor. Because we are using a fixed GPIO value our speed is fixed and thus will be more jerky and less tuned than a speed controlled system.

The results from the radar data show that the specifications for obstacle detection have been met. Our specification for operation in all weather conditions is also satisfied with the radar sensor. The radar module correctly determines the distance from several objects in the field of view and outputs the correct value to our minicomputer whether our cart is obstructed or not.

The GPS performed as expected. During A-to-B testing, the module was able to follow the path through the trip without missing any nodes. The accuracy was such that it was never necessary to stray off the path to trigger a node's proximity check, and the navigation consistently ran from start to finish. The precise testing supported the listed specifications claiming that accuracy was <3m. As expected, the accuracy was not enough to support use in lane-keeping.

The administration script was able to effectively communicate with submodules and manage the state of the cart. The performance of the module thus meets specifications. However, its implementation and completion is less than was planned. The original plans for decision-making were cut and replaced with the simple "stop or go" system. This works for very basic testing, but is nowhere near good enough for a properly usable product, both for safety reasons and because there is no ability to control speed, which would be necessary for turning or speed limits. The throttle output module was not implemented at all, and the braking is implemented but nonfunctional.

The specification for image processing has been met. Because we lost the use of the NVIDIA Jetson Nano 4GB in the 13th week of the semester, our platform that would have been capable of running that original design and classification model was compromised. But with the smaller 2GB Jetson Nano, a TensorFlowLite conversion was required to compress the original model into being more memory friendly and working in tandem with the rest of the system. However, with this compressed model there are some sacrifices in the accuracy that have to be accounted for as the model complexity decreases. The image classification currently runs with the use of a laptop or with the full system running on the Jetson, as proven in the results section of this report.

The specifications regarding battery life were not met. In the middle of the second semester, we discovered that all of the 8 V batteries do not hold a charge. We filled them up with distilled water and waited a couple days but they drew no current and had a voltage of 3V. Our hardware and braking is powered by two marine batteries that Mark Foreman had in storage. The

marine batteries are rated for 55Ah and 12V. Using two used car batteries, all four batteries in series add up to 48V that can power the golf cart. Unfortunately, the Ah rating of 55Ah is significantly lower than the 135Ah rating of the 8 V batteries. So the cart will be operational under a full load for less than an hour. The other issue is that we cannot perform a full test due to the fact that two batteries are used for the motor and there is not enough to power the hardware. Future implementation requires either a new set of 8V batteries with higher amp hour ratings. Or two more 12 V batteries but the time the golf cart can run on a full charge will be less than an hour.

The buck converters met the power requirements but did not meet the design requirements. Using a 12V battery, the buck converters can step down the voltage and supply power to the microcontrollers and Jetson Nano. However, the circuit is designed using a breadboard and premade buck converters rather than on a PCB. Any circuit on a breadboard will have some element of exposed wires which can lead to accidents. A finished product is a PCB board with safe housing that cannot be accessed easily.

The steering specifications have not been met. The stepper motor works, and the direction can be controlled with Arduino code. However, the driver cannot supply enough current without burning out. So the steering wheel cannot be connected and rotated. A new driver with a higher amp rating will need to replace the current SLA driver. It is possible other elements of the system may need to be changed when introducing a new driver.

Throttle specifications were not met. The proper outputs are being sent from our software programs, but the digital potentiometer would not correctly manipulate the throttle how it was hoped. When set up manually to test the throttle, changing the resistance alone did not make the cart move, this led us to the conclusion that the throttle needed to be adjusted in a different way to control it or the cart was not functioning properly.

Future Work

Before even thinking about deployment, the golf cart needs safety features. Our plans included an emergency stop button that the passenger could push, but it was not implemented. If the cart were to run autonomously with passengers aboard, a safety mechanism that allows the cart to stop running immediately would be essential.

Extra radar and ultrasonic sensors could be used to increase the field of vision for our computer system and allow detection of more objects around the cart. Currently, we only have computer vision on the front of the cart to detect obstructions in the forward direction. Additional sensors would allow the cart more autonomous ability to avoid obstacles and better safety features.

New batteries with higher amp-hour ratings for a longer operation time. As of now, the golf cart can only run for about forty minutes on a full charge. Getting batteries rated for 140 AH would be ideal. Then the marine batteries would be free to power the hardware and the breaking mechanism as originally planned.

Reducing size of the circuit. Building the buck converter on a PCB so the circuit can be hidden out of view. Prevents theft and is less likely to be damaged. PCB boards also eliminate exposed metal that could short circuit.

For future work in terms of braking we can implement a PWM signal instead of a GPIO. Lastly, designing a PCB board instead of having it be on a breadboard. With this we can use thicker wires that can handle more current and thus lead to longer braking times.

With the lack of a larger memory computing device, the project could not be fully implemented into one connected system. In the future, with time and more budget (as the Jetson Nano also has a very long backorder and waitlist), we would purchase a computing device with more memory that could run our system more seamlessly.

Additionally, it would always be better to train with larger and larger datasets with more diverse conditions so that the image processing model has more background information to be making its decisions on. Grabbing more data with different lighting conditions could help, although it was proven that even though the training data was all captured around midday, the model still makes accurate predictions at near dusk as our final image processing demonstration was performed right before the sunset.

Work was done to begin more complex throttle and steering control, as well as some lane detection algorithms, but without a working golf cart these areas of development went largely unused, as they would need to be verified and fine-tuned with the cart actually running. Instead only basic functionality was given to the golf cart motor controls to prove their functionality (limited to stop and go). Given a fully functioning, more work could be done to implement more

complex features like dynamic throttle and steering to ensure a smooth ride, as well as lane detection with micro-steering adjustments to ensure the golf cart stays on path.

GPS tracking could also be improved, mainly in two ways. The first would be to improve the map's accuracy. The current map is handmade, and nodes are placed sparsely and not very precisely. More points and more accurate points would allow for better tracking along a route. These could be pulled from services such as OpenStreetMaps. The second major improvement would be to take physical distance into account. The current algorithm finds the path with the fewest nodes. It would be simple to weight each edge according to physical distance, and that simple change would result in much quicker paths.

Decision-making also needs expansion. The current two-state system is not nearly robust enough to be used in a full product. A cart needs to be able to slow down, turn, obey traffic laws, etc, and two states just aren't enough for that. Especially in regards to throttle: a proper vehicle should be able to adjust its speed according to the situation, but the current cart can only choose between one speed and staying still. Whether it needs to turn, follow behind a slower cart, or obey traffic laws, speed must be variable.

Budget

Borrowed: Materials that were borrowed includes; 3 TM4C microcontrollers, 2 DM7404N buffers, 2 12 volt batteries, NVIDIA Jetson Nano 4GB.

Item	Quantity	Cost	Borrowed?
NVIDIA Jetson Nano 2GB	1	$343.32	
Arducam Wide Angle Camera	1	$31.98	
NVIDIA Jetson Nano 4GB	1	$0.00	Yes
MicroSD 32GB Disk	2	$0.00	Yes
Power Adapter USB-C 5V 3A	1	$10.99	
USBC power cable	1	$3.95	
TM4C123G Microcontrollers	4	$0.00	Yes
Steering Controller	4	$30.76	
5A Buck Converter	1	$7.39	
Adafruit Ultimate GPS Breakout v3	1	$29.95	Yes
N-channel Mosfets	3	$4.10	
P-channel Mosfets	3	$3.80	
DM7404N Buffer	2	$0.00	yes
Digital Potentiometer	1	$4.95	
Socket adapters	4	$0.95	
12V Batteries	2	$0.00	Yes
OPS241-FM-B-RP	1	$169.00	
Radar cover	1	$29.00	
Total		$670.14	

Global, Social, and Environmental Impact

Inter-relationship

A fleet of self driving vehicles around campus would interconnect the disabled faculty and staff. They would have the ability to get around campus with ease, faster than a regular student walking. This would save significant amounts of time depending on where they need to go around campus.

Potential Impacts on Health and Safety

Safety is the top priority for our senior project because the cart can pose a threat to students on campus. Working with a vehicle that can drive 15 mph and weighs more than 700 pounds puts a large responsibility on our team. Especially since we are implementing an autonomous driving system meaning we will not be in front of the wheel. Not only are pedestrians at risk, but other drivers on the road. Our golf cart must therefore be able to follow traffic laws to ensure the safety of other vehicles on the road. The safety of pedestrians and passengers is the top priority and we weighted that concept the highest in our weight table. After all, the purpose of our project is to serve the students and faculty at the university. Our senior project will make University of the Pacific a more friendly campus for handicapped students and faculty. The purpose of our project is to provide safe and efficient travel for students with disabilities. Our cart can also act as a safer form of transportation around campus as opposed to walking alone. The safety would slightly increase as a perfect self driving vehicle can become better than a human driver. Faster reaction times and continuous learning may one day develop a self-driving golf cart that gets into less accidents than a human driver.

Cultural and Environmental

A working autonomous vehicle could pave the way for a fleet of similar vehicles. The transportation ecosystem within UOP could be more efficient and safer. Students would spend less time walking to classes and use that extra time to be more productive. The potential for nighttime vehicles that could patrol the campus or transport students that feel unsafe to walk at night. A patrolling golf cart, even unmanned, would act as a deterrent to any thieves looking for bicycle wheels. Access to more autonomous vehicles would make living in residential buildings such as Calaveras, townhouses, and grace hall more appealing. The number of skateboard users would decrease, in turn, the number of skateboard accidents and thefts would decrease.

Costs

The immediate cost of the cart is outlined in the Dollar Budget section, summing to $670.14. However, running the vehicle on campus would come with future and recurring costs. A location for the cart to charge would have to be built. The charging process could be

human-operated or automatic, depending on future designs. The main battery of the golf cart could be automated, but the hardware uses another external battery that is separate which would require manual charging. Building a charging station entails an initial flat cost and ongoing operating costs. The electricity bill would increase for charging the cart everyday. The motor alone would consume 2500 Watts. The vehicle would require regular maintenance, both visual and functional. Cleaning the cart surfaces would probably be a daily job, either before deployment or after-hours. Mechanical maintenance would be performed similar to any other cart in Pacific's fleet. In addition, the cart would have to receive regular system checks, to make sure that no part of the automation is failing.

Conclusion

In conclusion, autonomous driving is an aspirational and far reaching goal that we can see on the horizon. But the actual implementation of such a system requires far more than a single semester of design and another semester of implementation, given a 5-person team. If this were the team's only responsibility (say no other classes were taken), it could have been an achievable goal to have a fully implemented system, but because that sort of time isn't available, this report provides a basic proof of concept of a system that would work, but needs more development to account for specific complexities and conditions that can arise in a daily transportation system. Overall, we have solid design, implementation and testing to prove capabilities of peripheral power, throttle, obstacle detection, and navigational guidance, all of which provides a framework which could be fully fleshed out at a further date. With unexpected complications arriving during the actual implementation phase of the project, the system itself is not quite what was originally envisioned and specified, but given those adversities the current project is a testament to hours of engineering and design work put in.

This progress would not have been possible without the guidance and support of all the acknowledgements mentioned earlier, and countless others.

References

[1] / Yunitsblog, “Hacking an electric golf cart gas pedal,” *yunitsblog*, 09-Aug-2016. [Online]. Available: https://yunitsblog.wordpress.com/2016/08/09/hacking-an-electric-golf-cart-gas-pedal/. [Accessed: Oct-2021].

[2] A. 15 and K. Burke, “How does a self-driving car see?,” *NVIDIA Blog*, 16-Apr-2019. [Online]. Available: https://blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see/. [Accessed: Sep-2021].

[3] A. Kumar, “How to use GPIO pins on Jetson Nano Developer Kit: Nvidia jetson,” *Maker Pro*, 24-Apr-2022. [Online]. Available: https://maker.pro/nvidia-jetson/tutorial/how-to-use-gpio-pins-on-jetson-nano-developer-kit. [Accessed: 23-Apr-2022].

[4] “Alpha prime,” *Velodyne Lidar*, 15-Oct-2021. [Online]. Available: https://velodynelidar.com/products/alpha-prime/. [Accessed: Nov-2022].

[5] “Automotive short range and medium range radar integrated circuits and reference designs,” *Automotive short range and medium range radar integrated circuits and reference designs | TI.com*. [Online]. Available: https://www.ti.com/solution/automotive-medium-short-range-radar. [Accessed: Sep-2021].

[6] “AWR1642 evaluation module single-chip mmwave sensing ...” [Online]. Available: https://www.ti.com/lit/pdf/swru508. [Accessed: Nov-2021].

[7] C. Hoffman, “Tesla Model 3, model Y dropping radar sensors,” *Car and Driver*, 29-Nov-2021. [Online]. Available: https://www.caranddriver.com/news/a36542541/tesla-model-3-model-y-pure-vision/. [Accessed: Dec-2021].

[8] D. Pandey, “Lane detection for a self-driving car using opencv,” *Medium*, 21-Apr-2021. [Online]. Available: https://medium.com/analytics-vidhya/lane-detection-for-a-self-driving-car-using-opencv-e2aa95105b89. [Accessed: 23-Apr-2022].

[9] “Data sheet for fullbridge IC Driver (H).” [Online]. Available: https://www.ti.com/lit/ds/symlink/drv8305.pdf. [Accessed: Nov-2021].

[10] “Driving dataset,” *a2d2.audi*, 13-Apr-2022. [Online]. Available: https://www.a2d2.audi/a2d2/en.html. [Accessed: Nov-2021].

[11] E. Editors, “Radar sensing for driverless vehicles,” *Digi*, 02-Nov-2016. [Online]. Available: https://www.digikey.com/es/articles/radar-sensing-for-driverless-vehicles. [Accessed: Dec-2021].

[12] K. Team, “Keras Documentation: Transfer Learning & Fine-tuning,” *Keras*. [Online]. Available: https://keras.io/guides/transfer_learning/. [Accessed: Nov-2021].

[13] “Levels of autonomous driving, explained,” *J.D. Power*. [Online]. Available: https://www.jdpower.com/cars/shopping-guides/levels-of-autonomous-driving-explained. [Accessed: Aug-2021].

[14] “Lidar vs radar: Detection, tracking, and imaging,” *Wevolver*. [Online]. Available: https://www.wevolver.com/article/lidar-vs-radar-detection-tracking-and-imaging#:~:text=LiDAR%20uses%20lasers%20with%20a%20much%20lower%20wavelength%20than%20the,high%2Dresolution%20image%20it%20creates. [Accessed: Nov-2021].

[15] “Load and preprocess images : Tensorflow Core,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/load_data/images. [Accessed: Nov-2021].

[16] “Object detection from a vehicle using deep learning ... - core.” [Online]. Available: https://core.ac.uk/download/pdf/146989032.pdf. [Accessed: Nov-2021].

[17] “Radar based object detection and tracking for autonomous driving,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/8443497. [Accessed: Nov-2021].

[18] “Radar contribution to highly automated driving,” *IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6986787. [Accessed: Nov-2021].

[19] “Sensors for Adas: 3. radars in Adas Technology,” *RSIP Vision*, 13-Jan-2022. [Online]. Available: https://rsipvision.com/sensors-in-adas-technology-radars/. [Accessed: Feb-2022].

[20] “SIMPLECV tutorial¶,” *Tutorial*. [Online]. Available: http://tutorial.simplecv.org/en/latest/. [Accessed: Nov-2021].

[22] “Tida-010045 Dev Kit,” *TIDA-010045 dev kit - Simulation, hardware & system design tools forum - Simulation, hardware & system design tools - TI E2E support forums*. [Online]. Available: https://e2e.ti.com/support/tools/simulation-hardware-system-design-tools-group/sim-hw-system-design/f/simulation-hardware-system-design-tools-forum/1071973/tida-010045-dev-kit. [Accessed: Nov-2021].

Appendixes

A. Component price and source list

Item	Quantity	Cost	Borrowed?	Link
NVIDIA Jetson Nano 2GB	1	$343.32		https://www.newegg.com/nvidia-945-13541-0000-000/p/N82E16813190013
Arducam Wide Angle Camera	1	$31.98		https://www.uctronics.com/arducam-8-mp-sony-imx219-camera-module-with-m12-lens-fisheye-for-invidia-jetson-nano.html
NVIDIA Jetson Nano 4GB	1	$0.00	Yes	
MicroSD 32GB Disk	2	$0.00	Yes	
Power Adapter USB-C 5V 3A	1	$10.99		https://www.amazon.com/SHNITPWR-Raspberry-Pi-Converter-Transformer-Compatible/dp/B0811H9GQ3/ref=asc_df_B0811H9GQ3/?tag=hyprod-20&linkCode=df0&hvadid=459618435588&hvpos=&hvnetw=g&hvrand=10193141423323129463&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9032200&hvtargid=pla-942094763736&psc=1
USBC power cable	1	$3.95		https://www.alie

				xpress.com/item/100500299315 7436.html?_ran dl_currency=US D&_randl_shipto =US&src=googl e&src=google&a lbch=shopping& acnt=631-313-3 945&slnk=&plac =&mtctp=&albbt =Google_7_sho pping&albagn=8 88888&isSmbAc tive=false&isSm bAutoCall=false &needSmbHouy i=false&albcp=1 5229744697&al bag=126912416 582&trgt=14805 51745650&crea =en1005002993 157436&netw=u &device=c&albp g=14805517456 50&albpd=en10 0500299315743 6&gclid=Cj0KC Qjw3IqSBhCoA RIsAMBkTb0ge qk-w2a44Q7ShI xHq7M9KzW33 Yg0Ro2fawZ2u wePnDtXPMbH JrAaAjMXEALw _wcB&gclsrc=a w.ds&aff_fcid=2 3c3e1e0cd9544 1c9cea2396b9c 8abe3-1648613 555688-04592- UneMJZVf&aff_f sk=UneMJZVf& aff_platform=aaf &sk=UneMJZVf &aff_trace_key= 23c3e1e0cd954

				41c9cea2396b9c8abe3-1648613555688-04592-UneMJZVf&terminal_id=9129afb67f49420bbde0bf89e386b77e&afSmartRedirect=y
TM4C123G Microcontrollers	4	$0.00	Yes	
Steering Controller	4	$30.76		https://www.ebay.com/itm/401213672320?chn=ps&norover=1&mkevt=1&mkrid=711-117182-37290-0&mkcid=2&itemid=401213672320&targetid=1262749492702&device=c&mktype=&googleloc=9032200&poi=&campaignid=15428034462&mkgroupid=1339471544481&rlsatarget=pla-1262749492702&abcId=9300763&merchantid=101639532&gclid=CjwKCAjwxOCRBhA8EiwA0X8hizNMbYroC6kap4-3s8zX5qpIMiiEUvdjKH5-7OJ393X9W-Uf0ZVH3hoCEqMQAvD_BwE
5A Buck Converter	1	$7.39		https://www.amazon.com/Solu-Adjustable-Step-down-Converter-Voltmeter/dp/B00VWL41TM/ref=

				asc_df_B00VWL41TM/?tag=hyprod-20&linkCode=df0&hvadid=216538221087&hvpos=&hvnetw=g&hvrand=3674925847936404409&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9032200&hvtargid=pla-352597795389&psc=1
Adafruit Ultimate GPS Breakout v3	1	$29.95	Yes	
N-channel Mosfets	3	$4.10		https://www.mouser.com/ProductDetail/Infineon-IR/IRLB3036PBF?qs=9%252BKlkBgLFf2IRFFZoHru3w%3D%3D&gclid=Cj0KCQiAu62QBhC7ARIsALXijXQofC_2BYgQRO4U4UABLrejjCJcEPUzMDVEgNwyiTsX0lQaShWryyoaAh8WEALw_wcB
P-channel Mosfets	3	$3.80		https://www.mouser.com/ProductDetail/Vishay-Semiconductors/SQP90P06-07L_GE3?qs=Vcr9%2FL0R50iEtlo1k0%2FsMQ%3D%3D
DM7404N Buffer	2	$0.00	yes	
Digital Potentiometer	1	$4.95		https://www.digikey.com/en/prod

				ucts/detail/adafruit-industries-llc/4286/10385110?utm_adgroup=Evaluation%20Boards%20-%20Expansion%20Boards%2C%20Daughter%20Cards&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Development%20Boards%2C%20Kits%2C%20Programmers_NEW&utm_term=&utm_content=Evaluation%20Boards%20-%20Expansion%20Boards%2C%20Daughter%20Cards
Socket adapters	4	$0.95		https://www.adafruit.com/product/4397
12V Batteries	2	$0.00	Yes	
OPS241-FM-B-RP	1	$169.00		https://omnipresense.com/product/ops241-b-short-range-fmcw-radar-sensor/
Radar cover	1	$29.00		https://omnipresense.com/product/67/

B: Operational instructions

To start the golf cart, power on the Jetson. Log in as the *bt* user with password *jetson* and navigate to *~/budgetTesla*. Execute *startup.sh* to boot up the administrative program and all the submodules. Programs can be started individually and found within their respective subdirectories.

C. Software

1. **Image Processing: Transfer Learning and Training of nasnet_mobile_v5**

```
import itertools
import os

import matplotlib.pylab as plt
import numpy as np
from PIL import Image
import glob

import tensorflow as tf
import tensorflow_hub as hub
#!pip install pyyaml h5py  # Required to save models in HDF5 format


print("TF version:", tf.__version__)
print("Hub version:", hub.__version__)
print("GPU is", "available" if tf.config.list_physical_devices('GPU') else "NOT AVAILABLE")

from google.colab import drive
drive.mount('/content/gdrive')
#drive.flush_and_unmount()

#!7z x /content/gdrive/MyDrive/data/train.zip
!7z x /content/gdrive/MyDrive/data/trainingDatav2.zip

model_name =  "nasnet_mobile" #"efficientnetv2-xl-21k" # @param ['efficientnetv2-s',
'efficientnetv2-m', 'efficientnetv2-l', 'efficientnetv2-s-21k', 'efficientnetv2-m-21k',
'efficientnetv2-l-21k', 'efficientnetv2-xl-21k', 'efficientnetv2-b0-21k',
'efficientnetv2-b1-21k', 'efficientnetv2-b2-21k', 'efficientnetv2-b3-21k',
'efficientnetv2-s-21k-ft1k', 'efficientnetv2-m-21k-ft1k', 'efficientnetv2-l-21k-ft1k',
'efficientnetv2-xl-21k-ft1k', 'efficientnetv2-b0-21k-ft1k', 'efficientnetv2-b1-21k-ft1k',
'efficientnetv2-b2-21k-ft1k', 'efficientnetv2-b3-21k-ft1k', 'efficientnetv2-b0',
'efficientnetv2-b1', 'efficientnetv2-b2', 'efficientnetv2-b3', 'efficientnet_b0',
'efficientnet_b1', 'efficientnet_b2', 'efficientnet_b3', 'efficientnet_b4', 'efficientnet_b5',
'efficientnet_b6', 'efficientnet_b7', 'bit_s-r50x1', 'inception_v3', 'inception_resnet_v2',
'resnet_v1_50', 'resnet_v1_101', 'resnet_v1_152', 'resnet_v2_50', 'resnet_v2_101',
'resnet_v2_152', 'nasnet_large', 'nasnet_mobile', 'pnasnet_large', 'mobilenet_v2_100_224',
'mobilenet_v2_130_224', 'mobilenet_v2_140_224', 'mobilenet_v3_small_100_224',
'mobilenet_v3_small_075_224', 'mobilenet_v3_large_100_224', 'mobilenet_v3_large_075_224']

model_handle_map = {
  "efficientnetv2-s":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_s/feature_vector/2",
  "efficientnetv2-m":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_m/feature_vector/2",
  "efficientnetv2-l":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_l/feature_vector/2",
  "efficientnetv2-s-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_s/feature_vector/2",
  "efficientnetv2-m-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_m/feature_vector/2",
  "efficientnetv2-l-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_l/feature_vector/2",
  "efficientnetv2-xl-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_xl/feature_vector/2",
  "efficientnetv2-b0-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_b0/feature_vector/2",
```

```
  "efficientnetv2-b1-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_b1/feature_vector/2",
  "efficientnetv2-b2-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_b2/feature_vector/2",
  "efficientnetv2-b3-21k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_b3/feature_vector/2",
  "efficientnetv2-s-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_s/feature_vector/2",
  "efficientnetv2-m-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_m/feature_vector/2",
  "efficientnetv2-l-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_l/feature_vector/2",
  "efficientnetv2-xl-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_xl/feature_vector/2",
  "efficientnetv2-b0-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_b0/feature_vector/2",
  "efficientnetv2-b1-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_b1/feature_vector/2",
  "efficientnetv2-b2-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_b2/feature_vector/2",
  "efficientnetv2-b3-21k-ft1k":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_b3/feature_vector/2",
  "efficientnetv2-b0":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_b0/feature_vector/2",
  "efficientnetv2-b1":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_b1/feature_vector/2",
  "efficientnetv2-b2":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_b2/feature_vector/2",
  "efficientnetv2-b3":
"https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet1k_b3/feature_vector/2",
  "efficientnet_b0": "https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1",
  "efficientnet_b1": "https://tfhub.dev/tensorflow/efficientnet/b1/feature-vector/1",
  "efficientnet_b2": "https://tfhub.dev/tensorflow/efficientnet/b2/feature-vector/1",
  "efficientnet_b3": "https://tfhub.dev/tensorflow/efficientnet/b3/feature-vector/1",
  "efficientnet_b4": "https://tfhub.dev/tensorflow/efficientnet/b4/feature-vector/1",
  "efficientnet_b5": "https://tfhub.dev/tensorflow/efficientnet/b5/feature-vector/1",
  "efficientnet_b6": "https://tfhub.dev/tensorflow/efficientnet/b6/feature-vector/1",
  "efficientnet_b7": "https://tfhub.dev/tensorflow/efficientnet/b7/feature-vector/1",
  "bit_s-r50x1": "https://tfhub.dev/google/bit/s-r50x1/1",
  "inception_v3": "https://tfhub.dev/google/imagenet/inception_v3/feature-vector/4",
  "inception_resnet_v2":
"https://tfhub.dev/google/imagenet/inception_resnet_v2/feature-vector/4",
  "resnet_v1_50": "https://tfhub.dev/google/imagenet/resnet_v1_50/feature-vector/4",
  "resnet_v1_101": "https://tfhub.dev/google/imagenet/resnet_v1_101/feature-vector/4",
  "resnet_v1_152": "https://tfhub.dev/google/imagenet/resnet_v1_152/feature-vector/4",
  "resnet_v2_50": "https://tfhub.dev/google/imagenet/resnet_v2_50/feature-vector/4",
  "resnet_v2_101": "https://tfhub.dev/google/imagenet/resnet_v2_101/feature-vector/4",
  "resnet_v2_152": "https://tfhub.dev/google/imagenet/resnet_v2_152/feature-vector/4",
  "nasnet_large": "https://tfhub.dev/google/imagenet/nasnet_large/feature_vector/4",
  "nasnet_mobile": "https://tfhub.dev/google/imagenet/nasnet_mobile/feature_vector/4",
  "pnasnet_large": "https://tfhub.dev/google/imagenet/pnasnet_large/feature_vector/4",
  "mobilenet_v2_100_224":
"https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/feature_vector/4",
  "mobilenet_v2_130_224":
"https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/feature_vector/4",
  "mobilenet_v2_140_224":
"https://tfhub.dev/google/imagenet/mobilenet_v2_140_224/feature_vector/4",
  "mobilenet_v3_small_100_224":
"https://tfhub.dev/google/imagenet/mobilenet_v3_small_100_224/feature_vector/5",
  "mobilenet_v3_small_075_224":
"https://tfhub.dev/google/imagenet/mobilenet_v3_small_075_224/feature_vector/5",
```

```
  "mobilenet_v3_large_100_224":
"https://tfhub.dev/google/imagenet/mobilenet_v3_large_100_224/feature_vector/5",
  "mobilenet_v3_large_075_224":
"https://tfhub.dev/google/imagenet/mobilenet_v3_large_075_224/feature_vector/5",
}

model_image_size_map = {
  "efficientnetv2-s": 384,
  "efficientnetv2-m": 480,
  "efficientnetv2-l": 480,
  "efficientnetv2-b0": 224,
  "efficientnetv2-b1": 240,
  "efficientnetv2-b2": 260,
  "efficientnetv2-b3": 300,
  "efficientnetv2-s-21k": 384,
  "efficientnetv2-m-21k": 480,
  "efficientnetv2-l-21k": 480,
  "efficientnetv2-xl-21k": 512,
  "efficientnetv2-b0-21k": 224,
  "efficientnetv2-b1-21k": 240,
  "efficientnetv2-b2-21k": 260,
  "efficientnetv2-b3-21k": 300,
  "efficientnetv2-s-21k-ft1k": 384,
  "efficientnetv2-m-21k-ft1k": 480,
  "efficientnetv2-l-21k-ft1k": 480,
  "efficientnetv2-xl-21k-ft1k": 512,
  "efficientnetv2-b0-21k-ft1k": 224,
  "efficientnetv2-b1-21k-ft1k": 240,
  "efficientnetv2-b2-21k-ft1k": 260,
  "efficientnetv2-b3-21k-ft1k": 300,
  "efficientnet_b0": 224,
  "efficientnet_b1": 240,
  "efficientnet_b2": 260,
  "efficientnet_b3": 300,
  "efficientnet_b4": 380,
  "efficientnet_b5": 456,
  "efficientnet_b6": 528,
  "efficientnet_b7": 600,
  "inception_v3": 299,
  "inception_resnet_v2": 299,
  "nasnet_large": 331,
  "pnasnet_large": 331,
}

model_handle = model_handle_map.get(model_name)
pixels = model_image_size_map.get(model_name, 224)

print(f"Selected model: {model_name} : {model_handle}")

IMAGE_SIZE = (pixels, pixels)
print(f"Input size {IMAGE_SIZE}")

BATCH_SIZE = 16 #@param {type:"integer"}

#from tensorflow.python.ops.gen_string_ops import substr
#data_dir = tf.keras.utils.get_file(
#    'flower_photos',
#    'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
#    untar=True)
#data_dir = "/content/train"
data_dir = "/content/trainingData"
```

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    label_mode="categorical",
    #labels=data_labels,
    labels="inferred", #can be infered or can be set in data_labels
    #class_names=["trafficlight", "stop", "speedlimit", "crosswalk"], #only necessary if data
labels are inferred
    #color_mode='rgb',
    batch_size=1,
    image_size=IMAGE_SIZE,
    shuffle = True,
    seed=111,
    subset="training",
    validation_split=.20,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    label_mode="categorical",
    #labels=data_labels,
    labels="inferred",
    #class_names=["trafficlight", "stop", "speedlimit", "crosswalk"],
    #color_mode='rgb',
    batch_size=1,
    image_size=IMAGE_SIZE,
    shuffle = True,
    seed=111,
    subset="validation",
    validation_split=.20,
)
print(train_ds.class_names)

def build_dataset(subset):
  return tf.keras.preprocessing.image_dataset_from_directory(
      data_dir,
      validation_split=.20,
      subset=subset,
      label_mode="categorical",
      # Seed needs to provided when using validation_split and shuffle = True.
      # A fixed seed is used so that the validation set is stable across runs.
      seed=123,
      image_size=IMAGE_SIZE,
      batch_size=1)

#train_ds = build_dataset("training")
class_names = tuple(train_ds.class_names)
#class_names = tuple(["trafficlight", "stop", "speedlimit", "crosswalk"])
#class_names = tuple(["0", "1"])
#train_size = train_ds.cardinality().numpy()
#rain_size = 964   #This number changes based on how many numbers are in the training set
#train_size = 1399
train_size = 1545
train_ds = train_ds.unbatch().batch(BATCH_SIZE)
train_ds = train_ds.repeat()

normalization_layer = tf.keras.layers.Rescaling(1. / 255)
preprocessing_model = tf.keras.Sequential([normalization_layer])
do_data_augmentation = False #@param {type:"boolean"}
if do_data_augmentation:
  preprocessing_model.add(
      tf.keras.layers.RandomRotation(40))
```

```
  preprocessing_model.add(
      tf.keras.layers.RandomTranslation(0, 0.2))
  preprocessing_model.add(
      tf.keras.layers.RandomTranslation(0.2, 0))
  # Like the old tf.keras.preprocessing.image.ImageDataGenerator(),
  # image sizes are fixed when reading, and then a random zoom is applied.
  # If all training inputs are larger than image_size, one could also use
  # RandomCrop with a batch size of 1 and rebatch later.
  preprocessing_model.add(
      tf.keras.layers.RandomZoom(0.2, 0.2))
  preprocessing_model.add(
      tf.keras.layers.RandomFlip(mode="horizontal"))
train_ds = train_ds.map(lambda images, labels:
                        (preprocessing_model(images), labels))

#val_ds = build_dataset("validation")
#valid_size = val_ds.cardinality().numpy()
#valid_size = 240 #This value depends on how many images are in the valdiation set
#valid_size = 349
valid_size = 386
val_ds = val_ds.unbatch().batch(BATCH_SIZE)
val_ds = val_ds.map(lambda images, labels:
                    (normalization_layer(images), labels))

do_fine_tuning = True #False #@param {type:"boolean"}

print("Building model with", model_handle)
model = tf.keras.Sequential([
    # Explicitly define the input shape so the model can be properly
    # loaded by the TFLiteConverter
    tf.keras.layers.InputLayer(input_shape=IMAGE_SIZE + (3,)),
    hub.KerasLayer(model_handle, trainable=do_fine_tuning),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(len(class_names),
                          kernel_regularizer=tf.keras.regularizers.l2(0.0001))
])
model.build((None,)+IMAGE_SIZE+(3,))
model.summary()

model.compile(
  optimizer=tf.keras.optimizers.SGD(learning_rate=0.005, momentum=0.9),
  loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True, label_smoothing=0.1),
  metrics=['accuracy'])

steps_per_epoch = train_size // BATCH_SIZE
validation_steps = valid_size // BATCH_SIZE
tf.config.run_functions_eagerly(False)
run_opts = tf.compat.v1.RunOptions(report_tensor_allocations_upon_oom = True)
hist = model.fit(
    train_ds,
    epochs=4, steps_per_epoch=steps_per_epoch,
    validation_data=val_ds,
    validation_steps=validation_steps).history

plt.figure()
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0,2])
plt.plot(hist["loss"])
plt.plot(hist["val_loss"])
```

```
plt.figure()
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0,1])
plt.plot(hist["accuracy"])
plt.plot(hist["val_accuracy"])

x, y = next(iter(val_ds))
image = x[0, :, :, :]
true_index = np.argmax(y[0])
plt.imshow(image)
plt.axis('off')
plt.show()

'''
current_dir = os.getcwd()
print(current_dir)
#img= Image.open("../content/human_detection_dataset/human_detection_dataset/0/"114.png"")
#image_dir = "../content/human_detection_dataset/human_detection_dataset/0/"114.png""
image_list = []
for filename in glob.glob('human_detection_dataset/human_detection_dataset/0/*.png'):
    im=Image.open(filename)
    image_list.append(im)
img= Image.open(image_list[0])
plt.imshow(img)
plt.axis('off')
plt.show()
'''

# Expand the validation image to (1, 224, 224, 3) before predicting the label
prediction_scores = model.predict(np.expand_dims(image, axis=0))
predicted_index = np.argmax(prediction_scores)
print("True label: " + class_names[true_index])
print("Predicted label: " + class_names[predicted_index])

# Create and train a new model instance.
#model = create_model()
#model.fit(train_images, train_labels, epochs=5)

# Save the entire model to a HDF5 file.
# The '.h5' extension indicates that the model should be saved to HDF5.
model.save('/content/gdrive/MyDrive/Colab Notebooks/nasnet_mobile_v5.h5')
model.summary()

o_img_path = '/content/gdrive/MyDrive/Colab Notebooks/obstructed.PNG'
u_img_path = '/content/gdrive/MyDrive/Colab Notebooks/unobstructed.PNG'
import cv2

IMAGE_CHANNEL = 3 #1 # or 3

def prepare(filepath):
    IMG_SIZE = pixels
    img_array = cv2.imread(filepath)
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
    return new_array.reshape(IMG_SIZE, IMG_SIZE, IMAGE_CHANNEL)
o_img = prepare(o_img_path)
u_img = prepare(u_img_path)

prediction_scores = model.predict(np.expand_dims(o_img, axis = 0))
predicted_index = np.argmax(prediction_scores)
print("predicted label for obstructed image: " + class_names[predicted_index])
```

```
prediction_scores = model.predict(np.expand_dims(u_img, axis=0))
predicted_index = np.argmax(prediction_scores)
print("predicted label for unobstructed image: " + class_names[predicted_index])

saved_model_path = f"/content/gdrive/MyDrive/Colab Notebooks/{model_name}"
tf.saved_model.save(model, saved_model_path)
```

2. Headless Image Processing Object Detection for Jetson Nano

```
import cv2
import tensorflow as tf
import numpy as np
import time
from serial import Serial
from btcomms import BTComms



def gstreamer_pipeline(
    capture_width=1920,
    capture_height=1080,
    display_width=224,#960,
    display_height=224,#540,
    framerate=30,
    flip_method=2,
):
    return (
        "nvarguscamerasrc ! "
        "video/x-raw(memory:NVMM), "
        "width=(int)%d, height=(int)%d, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink drop=True"
        % (
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )

#def main():
serialInst = Serial()

class_names = tuple(["obstructed", "unobstructed"])
pixels = 224
IMAGE_SIZE = (pixels, pixels)



converted_model = "/home/bt/budgetTesla/compVis/converted_model_quant.tflite"
```

```
interpreter = tf.lite.Interpreter(model_path=str(converted_model))
interpreter.allocate_tensors()
#print(interpreter.get_input_details())
input_details = interpreter.get_input_details()[0]
output_details = interpreter.get_output_details()[0]
#window_title = "Obstacle Detection"

video_capture = cv2.VideoCapture(gstreamer_pipeline(), cv2.CAP_GSTREAMER)




if video_capture.isOpened():
    com = BTComms('V')
    print('[vision] sending confirmation')
    com.confirm()   #send bt_admin that vision is loaded and ready
    print('[vision] recieved ACK')
    try:
        #cv2.namedWindow(window_title, cv2.WINDOW_AUTOSIZE)
        #for i in range(0,100):
        prev_state = False
        while True:
            ret, frame = video_capture.read()
            #print(type(frame))
            f_frame = cv2.resize(frame, (224, 224)).astype('float32')

            f_frame = np.expand_dims(f_frame, axis=0)
            #print(type(edit_frame)
            interpreter.set_tensor(input_details["index"], f_frame)
            interpreter.invoke()
            prediction_scores = interpreter.get_tensor(output_details["index"])[0]
            predicted_index = np.argmax(prediction_scores)
            if predicted_index != prev_state:
                prev_state = predicted_index
                com.send(str(predicted_index))  #send value to bt admin
                # print("Predicted label: " + class_names[predicted_index])

            #used if displaying the video
            #time.sleep(.5)
                 # Check to see if the user closed the window
                 # Under GTK+ (Jetson Default), WND_PROP_VISIBLE does not work
correctly. Under Qt it does
                 # GTK - Substitute WND_PROP_AUTOSIZE to detect if window has been
closed by user
            #if cv2.getWindowProperty(window_title, cv2.WND_PROP_AUTOSIZE) >= 0:
            #    cv2.imshow(window_title, frame)
            #else:
```

```
            #    break
            keyCode = cv2.waitKey(10) & 0xFF
                 # Stop the program on the ESC key or 'q'
            if keyCode == 27 or keyCode == ord('q'):
                break
    finally:
        video_capture.release()
        cv2.destroyAllWindows()
else:
    print("Unable to open camera")


    '''
    while true:
           prediction_scores = new_model.predict(np.expand_dims(img, axis=0))
           predicted_index = np.argmax(prediction_scores)
           print("Predicted label: " + class_names[predicted_index])
    '''

#if __name__ == "__main__":
#    main()
```

3. Lane Detection Development

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

def canyEdgeDetector(image):
    edged = cv2.Canny(image, 300, 350)
    return edged

def getROI(image):
    height = image.shape[0]
    width = image.shape[1]
    # Defining Triangular ROI: The values will change as per your camera mounts
    triangle = np.array([[(0, height-200), (width, height-200), (width-280,
int(height/2.7))]])
    # creating black image same as that of input image
    black_image = np.zeros_like(image)
    # Put the Triangular shape on top of our Black image to create a mask
    mask = cv2.fillPoly(black_image, triangle, 255)
    # applying mask on original image
    masked_image = cv2.bitwise_and(image, mask)
    return masked_image

def getLines(image):
    lines = cv2.HoughLinesP(image, 0.3, np.pi/180, 100, np.array([]),
minLineLength=70, maxLineGap=20)
    return lines

def displayLines(image, lines):
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line.reshape(4) #converting to 1d array
            cv2.line(image, (x1, y1), (x2, y2), (255, 0, 0), 10)
    return image

def getLineCoordinatesFromParameters(image, line_parameters):
    slope = line_parameters[0]
    intercept = line_parameters[1]
    y1 = image.shape[0]  # since line will always start from bottom of image
    y2 = int(y1 * (3.4 / 5))  # some random point at 3/5
    x1 = int((y1 - intercept) / slope)
    x2 = int((y2 - intercept) / slope)
    return np.array([x1, y1, x2, y2])

def getSmoothLines(image, lines):
    left_fit = []  # will hold m,c parameters for left side lines
```

```
    right_fit = []  # will hold m,c parameters for right side lines

    for line in lines:
        x1, y1, x2, y2 = line.reshape(4)
        parameters = np.polyfit((x1, x2), (y1, y2), 1)
        slope = parameters[0]
        intercept = parameters[1]

        if slope < 0:
            left_fit.append((slope, intercept))
        else:
            right_fit.append((slope, intercept))

    left_fit_average = np.average(left_fit, axis=0)
    right_fit_average = np.average(right_fit, axis=0)

    # now we have got m,c parameters for left and right line, we need to know x1,y1
x2,y2 parameters
    left_line = getLineCoordinatesFromParameters(image, left_fit_average)
    right_line = getLineCoordinatesFromParameters(image, right_fit_average)
    return np.array([left_line, right_line])

from google.colab import drive
drive.mount('/content/gdrive')

image_path = '/content/gdrive/MyDrive/Colab Notebooks/pathway.PNG'
img = cv2.imread(image_path) #Load Image
cv2_imshow(img)

edged_image = canyEdgeDetector(img)   # Step 1
cv2_imshow(edged_image)
roi_image = getROI(edged_image)         # Step 2
cv2_imshow(roi_image)
lines = getLines(roi_image)             # Step 3

smooth_lines = getSmoothLines(img, lines)    # Step 5
image_with_smooth_lines = displayLines(img, smooth_lines) # Step 4

cv2.imshow("Output", image_with_smooth_lines)
cv2.waitKey(0)
```

4. Throttle Control Simulation

```
import time
steeringAngle = 0
GPSFlag = 0
currThrottle = 0
clock = 0
universalSleepTime = .5
def throttleControl(R, O, U, S, D):
   #inputs: clock, R, O, U, S, D, inputThrottle
   #clock represents the 2 Hz, clock cycle that updates throttle control every half
second
   #R: radar detection, radar detects and obstacle
   #O: image processing obstructed, camera and image classifier detects an obstacle
   #U: image processing unobstructed, camera and image classifier do not detect an
obstacle
   #S: Stop sign approaching: GPS locational data indicates that golf cart is
approaching a stop sign
   #D: Destination approaching: GPS locational data indicates that golf cart is
approaching its end destination
   #inputThrottle: The current value of the throttle control, before changes are
applied

   #outputs:
   #throttleVal: value of throttle control sent to microcontroller.
   #This value has a minimum value of -50 and a maximum value of 50.
   # -50 represents maximum braking power
   # 50 represents maximum throttle
   # 0 represents no brake and no throttle application
   # >0 means no brake applied
   # <0 means no throttle applied

   #if (clock == 1):
   global currThrottle
   throttleVal = currThrottle + ((-10 * R )+ (-3 * O) + (-2 * S) + (-2 * D) + (U))

   #else:

   #throttleVal = inputThrottle

   if throttleVal > 50:
       outputThrottle = 50
   elif throttleVal < -50:
       outputThrottle = -50
   else:
       outputThrottle = throttleVal

   currThrottle = outputThrottle
```

```
   #return outputThrottle
   print("Throttle Val: " + str(outputThrottle))
   #return outputThrottle


def steeringControl(pathArray):
   #inputs: clock, pathArray
   # clock represents the 2 Hz, clock cycle that updates throttle control every half
second
   #pathArray: array of elements, from navigation system indicating what path is
required to achieve destination
   #outputs: updates a global variable indicating how much the wheel should turn

   #path codes
   #0: end of path
   #1: straight line path
   #2: right turn path
   #3: left turn path

   throttleThreshold = 0
   degreeTurn = 90
   for pathElement in pathArray:
       if pathElement == 1:
           driveStraight()
       elif pathElement == 2:
           turnRight(throttleThreshold, degreeTurn)
       elif pathElement == 3:
           turnLeft(throttleThreshold, degreeTurn)
       else:
           print("Something went wrong")


def driveStraight():
   global GPSFlag
   turn(90)
   while GPSFlag != 0:
       steeringAngle = 0

def turnRight(numThrottleThreshold, degreeTurn):
   global universalSleepTime
   global clock
   global currThrottle
   throttleCount = 0
   turn(degreeTurn)
   curClock = clock
   while (numThrottleThreshold > throttleCount):
       time.sleep(universalSleepTime)
```

```
        if clock == 1:
            clock = 0
            throttleCount += currThrottle
        elif clock == 0:
            clock = 1

    turn(0)

def turnLeft(numThrottleThreshold, degreeTurn):
    global unviversalSleepTime
    global currThrottle
    global clock
    throttleCount = 0

    turn(-1 * degreeTurn)
    curClock = clock
    while (numThrottleThreshold > throttleCount):
        time.sleep(universalSleepTime)
        if clock == 1:
            clock = 0
            throttleCount += currThrottle
        elif clock == 0:
            clock = 1

    turn(0)

def turn(degrees = 0):
    #send to oscar that the cart should turn the wheel to input degrees
    print("send to oscar that the cart should turn the wheel to " + str(degrees) +"
degrees")

def throttleControlTest():
    global clock
    clockCount = 0
    index        = [0, 1, 2, 3, 4, 5, 6, 7, 8,
9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,
38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66
,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,9
5,96,97,98,99,100]
    Radar        = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
    Obstructed   = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```

```
   StopSign     = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1]
   Destination  = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
   Unobstructed = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
   #inputThrottle = 0
   while(clockCount != 101):
       time.sleep(universalSleepTime)
       if clock == 0:
           clock = 1
           print("Clock count: " + str(clockCount))
           throttleControl(Radar[clockCount], Obstructed[clockCount],
Unobstructed[clockCount], StopSign[clockCount], Destination[clockCount])
           clockCount += 1
       else:
           clock = 0


throttleControlTest()
```

5. Braking Prototype

```
/**
 * main.c
 *

 */

//#include "TM4c123gh6pm.h"
#define SYSCTL_RCGCGPIO_R (*(( volatile unsigned long *)0x400FE608 ) )

#define GPIO_PORTF_DATA_R (*(( volatile unsigned long *)0x40025038 ) )
#define GPIO_PORTF_LOCK_R (*(( volatile unsigned long *)0x40025520 ) )
#define GPIO_PORTF_CR_R   (*(( volatile unsigned long *)0x40025524 ) )
#define GPIO_PORTF_PUR_R  (*(( volatile unsigned long *)0x40025510 ) )
#define GPIO_PORTF_DIR_R  (*(( volatile unsigned long *)0x40025400 ) )
#define GPIO_PORTF_DEN_R  (*(( volatile unsigned long *)0x4002551C ) )

#define GPIO_PORTB_DATA_R (*(( volatile unsigned long *)0x400053FC ) )
#define GPIO_PORTB_LOCK_R (*(( volatile unsigned long *)0x40005520 ) )
#define GPIO_PORTB_CR_R   (*(( volatile unsigned long *)0x40005524 ) )
#define GPIO_PORTB_PUR_R  (*(( volatile unsigned long *)0x40005510 ) )
#define GPIO_PORTB_DIR_R  (*(( volatile unsigned long *)0x40005400 ) )
#define GPIO_PORTB_DEN_R  (*(( volatile unsigned long *)0x4000551C ) )

#define GPIO_PORTE_DATA_R (*(( volatile unsigned long *)0x400243FC ) )
#define GPIO_PORTE_LOCK_R (*(( volatile unsigned long *)0x40024520 ) )
#define GPIO_PORTE_CR_R   (*(( volatile unsigned long *)0x40024524 ) )
#define GPIO_PORTE_PUR_R  (*(( volatile unsigned long *)0x40024510 ) )
#define GPIO_PORTE_DIR_R  (*(( volatile unsigned long *)0x40024400 ) )
#define GPIO_PORTE_DEN_R  (*(( volatile unsigned long *)0x4002451C ) )

#define GPIO_PORTD_DATA_R (*(( volatile unsigned long *)0x400073FC ) )
#define GPIO_PORTD_LOCK_R (*(( volatile unsigned long *)0x40007520 ) )
#define GPIO_PORTD_CR_R   (*(( volatile unsigned long *)0x40007524 ) )
#define GPIO_PORTD_PUR_R  (*(( volatile unsigned long *)0x40007510 ) )
#define GPIO_PORTD_DIR_R  (*(( volatile unsigned long *)0x40007400 ) )
#define GPIO_PORTD_DEN_R  (*(( volatile unsigned long *)0x4000751C ) )


int main(void)
{
    //SYSCTL_RCGCGPIO_R |= 0x20; // Enable clock for PORTF
    SYSCTL_RCGCGPIO_R |= 0x3A;  // Enable clock for PORTA,B,E,F
    GPIO_PORTF_LOCK_R  |= 0x4C4F434;
   // GPIO_PORTF_LOCK_R  |= 0x4C4F434B;
    GPIO_PORTF_CR_R  |= 0x01;
```

```
  GPIO_PORTF_DIR_R  |= 0x0E;  // Configure PORTF Pin1, 2 and 3 digital output pins
  GPIO_PORTF_PUR_R  |= 0x10;
  GPIO_PORTF_DEN_R  |= 0x0E;  // Enable PORTF Pin1, 2 and 3 as a digital pins

 // GPIO_PORTB_LOCK_R  |= 0x4C4F434B;
  GPIO_PORTB_CR_R  |= 0x01;
  GPIO_PORTB_DIR_R  |= 0x0E;  // Configure PORTF Pin1, 2 and 3 digital output pins
  GPIO_PORTB_PUR_R  |= 0x10;
  GPIO_PORTB_DEN_R  |= 0x0E;  // Enable PORTF Pin1, 2 and 3 as a digital pins

//  GPIO_PORTE_LOCK_R  |= 0x4C4F434;
  GPIO_PORTE_CR_R  |= 0x01;
  GPIO_PORTE_DIR_R  |= 0x0E;  // Configure PORTF Pin1, 2 and 3 digital output pins
  GPIO_PORTE_PUR_R  |= 0x10;
  GPIO_PORTE_DEN_R  |= 0x0E;  // Enable PORTF Pin1, 2 and 3 as a digital pins

//  GPIO_PORTA_LOCK_R  |= 0x4C4F434;
  GPIO_PORTD_CR_R  |= 0x01;
  GPIO_PORTD_DIR_R  |= 0x0E;  // Configure PORTF Pin1, 2 and 3 digital output pins
  GPIO_PORTD_PUR_R  |= 0x10;
  GPIO_PORTD_DEN_R  |= 0x0E;  // Enable PORTF Pin1, 2 and 3 as a digital pins

  GPIO_PORTF_DATA_R |= 0x02; // turn off red LED
  GPIO_PORTB_DATA_R |= 0x02; // turn off red LED
  GPIO_PORTE_DATA_R &= 0x00; // turn off red LED
  GPIO_PORTD_DATA_R &= 0x00; // turn off red LED

  int x = 2;

  int y = 0;


  if(x==0)  //letting cable out
  {
      GPIO_PORTF_DATA_R &= 0x00; // turn off PF1
      GPIO_PORTB_DATA_R |= 0x02; // turn on PB1
      GPIO_PORTE_DATA_R |= 0x02; // turn on PE1
      GPIO_PORTD_DATA_R &= 0x00; // turn off PD1
     // delay(1000);    // delay for one second
      GPIO_PORTF_DATA_R &= 0x00; // turn off PF1
      GPIO_PORTB_DATA_R &= 0x00; // turn off PB1
      GPIO_PORTE_DATA_R &= 0x00; // turn off PE1
      GPIO_PORTD_DATA_R &= 0x00; // turn off PD1
  }

  else if (x==1)  //braking
  {
```

```
    GPIO_PORTF_DATA_R &= 0x00; // turn off PF1
    GPIO_PORTF_DATA_R |= 0x02; // turn on PF1
    GPIO_PORTB_DATA_R &= 0x00; // turn off PB1
    GPIO_PORTE_DATA_R &= 0x00; // turn off PE1
    GPIO_PORTD_DATA_R |= 0x02;; // turn on PD1


    GPIO_PORTF_DATA_R &= 0x00; // turn off PF1
    GPIO_PORTB_DATA_R &= 0x00; // turn off PB1
    GPIO_PORTE_DATA_R &= 0x00; // turn off PE1
    GPIO_PORTD_DATA_R &= 0x00; // turn off PD1
}




else if (x==2)   //braking then letting out then braking etc...
{

//    GPIO_PORTF_DATA_R &= 0x00; // turn off PF1
    GPIO_PORTF_DATA_R |= 0x02; // turn on PF1   //brake
    GPIO_PORTE_DATA_R &= 0x00; // turn off PE1
    GPIO_PORTD_DATA_R |= 0x02;; // turn on PD1
    GPIO_PORTB_DATA_R &= 0x00; // turn off PB1


    for(y=0; y < 1*3000000; y=y+1);  //delay about 3 second
        {
        }

    GPIO_PORTB_DATA_R |= 0x02; // turn on PB1   //release
    GPIO_PORTD_DATA_R &= 0x00; // turn off PD1
    GPIO_PORTE_DATA_R |= 0x02; // turn on PE1
    GPIO_PORTF_DATA_R &= 0x00; // turn off PF1


    for(y=0; y < 1*3000000; y=y+1);  //delay about 3 second
        {
        }

    GPIO_PORTF_DATA_R |= 0x02; // turn on PF1   //brake
    GPIO_PORTE_DATA_R &= 0x00; // turn off PE1
    GPIO_PORTD_DATA_R |= 0x02;; // turn on PD1
    GPIO_PORTB_DATA_R &= 0x00; // turn off PB1
```

```
        for(y=0; y < 1*3000000; y=y+1);  //delay about 3 second
            {
            }
        GPIO_PORTB_DATA_R |= 0x02; // turn on PB1   //release
        GPIO_PORTD_DATA_R &= 0x00; // turn off PD1
        GPIO_PORTE_DATA_R |= 0x02; // turn on PE1
        GPIO_PORTF_DATA_R &= 0x00; // turn off PF1
        for(y=0; y < 1*3000000; y=y+1);  //delay about 3 second
            {
            }


        GPIO_PORTF_DATA_R |= 0x02; // turn off red LED
        GPIO_PORTB_DATA_R |= 0x02; // turn off red LED
        GPIO_PORTE_DATA_R &= 0x00; // turn off red LED
        GPIO_PORTD_DATA_R &= 0x00; // turn off red LED
    }

    else

        for(y=0; y < 1*3000000; y=y+1);  //delay about 3 second
            {
            }



     //   for(y=0; y < 3000000; y=y+1);  //delay about 3 seconds
      //          {

      //          }



/*  GPIO_PORTF_DATA_R |= 0x02; // turn on red LED
    GPIO_PORTB_DATA_R |= 0x02; // turn on red LED
    GPIO_PORTE_DATA_R |= 0x02; // turn on red LED
    GPIO_PORTD_DATA_R |= 0x02; // turn on red LED

  //  sleep(5);
    GPIO_PORTF_DATA_R &= 0x00; // turn off red LED
    GPIO_PORTB_DATA_R &= 0x00; // turn off red LED
    GPIO_PORTE_DATA_R &= 0x00; // turn off red LED
    GPIO_PORTD_DATA_R &= 0x00; // turn off red LED
*/
```

```
    //GPIO_PORTF_DATA_R |= 0x04; // turn on blue LED
    //GPIO_PORTF_DATA_R |= 0x08; // turn on green LED

    /*unsigned int state;
    SYSCTL->RCGCGPIO |= 0x20;
    GPIOF->LOCK = 0x4C4F434B;
    GPIOF->CR = 0x01;
    GPIOF->PUR |= 0x10;
    GPIOF->DIR |= 0x02;
    GPIOF->DEN |= 0x12;*/

       return 0;
}
```

6. **Braking Code**

See: https://github.com/kmakmichael/seniorproject_mcu

7. Steering Code

```
// Stepper Pins
uint8_t Reset = 7;
uint8_t Dir = 4;
uint8_t M1 = 0;
uint8_t M2 = 1;
uint8_t M3 = 2;
uint8_t Ref = 6;
uint8_t Sync = 5;
uint8_t Clock = 3;
int SPEED = 15;
// copied
void setup() {
 // put your setup code here, to run once:
 // Stepper pin direction inits

 //pinMode: Configures the specified pin to behave either as an input or an output

  pinMode(Reset, OUTPUT);
  pinMode(Dir, OUTPUT);
  pinMode(M1, OUTPUT);
  pinMode(M2, OUTPUT);
  pinMode(M3, OUTPUT);
  pinMode(Ref, INPUT);
  pinMode(Sync, OUTPUT);
  pinMode(Clock, OUTPUT);

  // Stepper pin init states
  //digitalWrite: If the pin has been configured as an OUTPUT with pinMode(), its
voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH,
0V (ground) for LOW.

  digitalWrite(Reset, LOW);
  digitalWrite(Dir, HIGH);
  digitalWrite(M1, LOW);
  digitalWrite(M2, LOW);
  digitalWrite(M3, LOW);
  //analogWrite(Ref, 150);
  digitalWrite(Sync, LOW);
  digitalWrite(Clock, LOW);
}

void loop() {
  // put your main code here, to run repeatedly:
```

```
  digitalWrite(Dir, HIGH);
  for (int i=0; i<400; i++) {


  digitalWrite(Clock, HIGH);
  delay(SPEED);
  digitalWrite(Clock, LOW);
  delay(SPEED);

  }

  delay(2000);


  digitalWrite(Dir, LOW);
  for (int j=0; j<400; j++) {


  digitalWrite(Clock, HIGH);
  delay(SPEED);
  digitalWrite(Clock, LOW);
  delay(SPEED);

  }

   delay(2000);


}
```

8. Radar Code

```
import serial.tools.list_ports
import time

serialInst = serial.Serial()

def checkclose(list1):
    for x in list1:
        if 2.5 >= x:
            return 1
    return 0

serialInst.baudrate = 9600
serialInst.port = 'COM4'
serialInst.open()


rate = "WD"
objects = "O3"
max = "r<10\r\n"
min = "r>1.5\r\n"

serialInst.write(rate.encode('utf-8'))
serialInst.write(objects.encode('utf-8'))
serialInst.write(max.encode('utf-8'))
serialInst.write(min.encode('utf-8'))

while True:

    if serialInst.in_waiting:
        packet = serialInst.readline()
        decoded = packet.decode('utf')
        decoded_list = decoded.strip()
        decoded_list = decoded_list.split(',')
        try:
            data = list(map(float, decoded_list))
            st = checkclose(data)
            print(f'{"Obstructed" if st else "UnObstructed"}')
        except:
            print("Something went wrong")
```

9. Digital Potentiometer Protoype

```
/**
 * main.c using TivaWare library
 */
#include <stdint.h>
#include <stdbool.h>
#include "driverlib/systick.h"
#include "driverlib/i2c.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "driverlib/uart.h"


void main( void ) {

    int8_t input;


    // Enable the GPIOB peripheral
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    //
    // Wait for the GPIOB module to be ready.
    //
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB))
    {
    }

    // Enable the I2C0 peripheral
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    // Wait for the I2C0 module to be ready.
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_I2C0))
    {
    }

    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3);
    //
    // Enable I2C0 functionality on GPIO Port B pins 2 and 3.
    //
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);
```

```
    // Initialize Master and Slave
    I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);

    // Specify slave address
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x28, false);

  /*  if (throttle == 1)
        input = 0x40;
    else
        input = 0;
*/
    // Place the characters to be sent in the data register
    I2CMasterDataPut(I2C0_BASE, input);

    // Initiate send of character from Master to Slave
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);

    // Delay until transmission completes
    while(I2CMasterBusBusy(I2C0_BASE))
    {
    }
}
```

10. GPS Processing Code

See: https://github.com/kmakmichael/geotracking_py

11. GPS Testing Code

See: https://github.com/kmakmichael/macronav-playground

12. BTAdmin & BTComms, testing code

See: https://github.com/kmakmichael/btadmin