MIAMI UNIVERSITY

Self Driving Car

Fall 2023

College of Engineering and Computing

Department of Electrical and Computer Engineering

Professor – James Leonard

Connor Grammens

Zavier Parker

Tyler Worley

Zhuyu Lu

Table of Contents:

Introduction

The project assigned is to be able to develop a way for a remote control car to be able to completely run by itself without any human input. When looking at self-driving cars in the workplace it is seen that these cars represent a groundbreaking technological advancement that has garnered significant attention in recent years. These vehicles are designed to navigate and operate without human intervention, relying on a combination of sophisticated sensors, cameras, GPS, lidar, and precise algorithms. The development of self-driving cars has been a focal point for technology companies and automotive manufacturers, driven by the promise of increased safety, improved traffic efficiency, and enhanced mobility for individuals with limited mobility.

The core technology behind self-driving cars involves the interpretation of vast amounts of data from sensors to make real-time decisions, such as steering, acceleration, and braking. Machine learning and deep neural networks play a crucial role in enabling these vehicles to recognize and respond to complex and dynamic environments, including other vehicles, pedestrians, and various road conditions. The goal of this project is to implement all of these factors into a toy car that is able to roam around Miami's campus on its own.

Project Background

The project has now been in development for 3 years at Miami University with the collection of work for 3 teams. The previous teams have worked on the cars function and crosswalk detection algorithm using Yolo V8. This is where the first group designed and created the cars ability to drive based on programming in the Arduino IDE. There is also an implemented controller that is connected to the car itself that is able to control the speed and direction the car drives in. In the car there are currently three modes present, mode 1 is the general mode that does nothing to the car, mode 2 is the base driving for the car that uses the controller to control the car. Finally mode 3 is the current mode that is being used to have the car drive by itself rather than by using a controller.

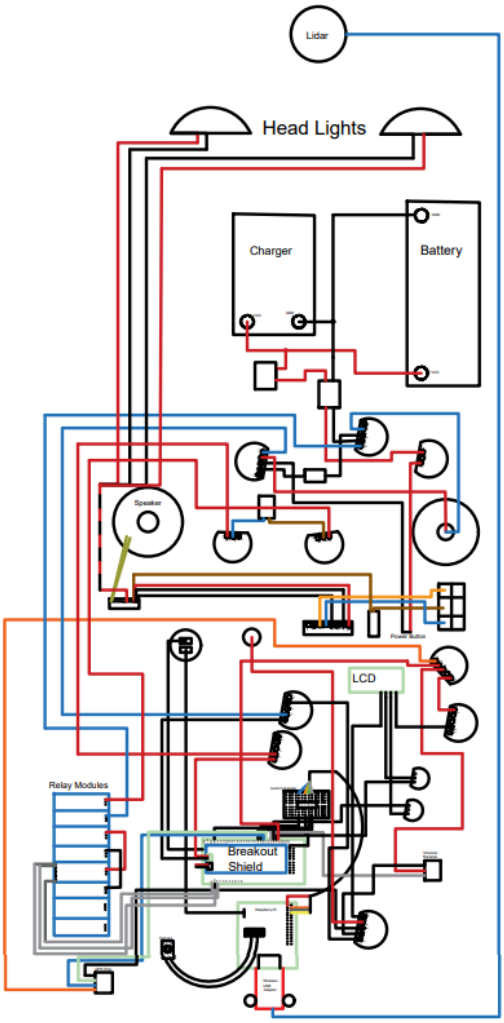

Fig. 1: Most recent circuit diagram of car

Mode 3 is going to utilize all the aspects of a self-driving car that all groups have worked on including the previous group's cross walk algorithm. This algorithm is able to detect the white lines of the crosswalk in order to get an accurate determination and allow the car to be able to recognize it. The circuit diagram has also been done by this group to help give a representation of the connections represented in this project in helping future groups as well. Now last year the work on the car focused on 4 objectives that were carried out. These objectives were path finding using gps, serial connection. Object detection, and simulation. The first part of this work focused on research into each of these topics and how to introduce them in forms to be used in the car. The

second part of this work last year was data collection and building blocks for these objectives in order to use for this semester. This leads into the processes used this semester to achieve the step forward in the project.

Project Research

Object Detection Using YOLOv8 and Person Tracking:

Incorporating object detection and person tracking into our project aimed to enhance our existing system's capabilities. My initial task involved implementing object detection specifically for recognizing people. I began by researching the use of YOLOv8, following the precedent set by the previous team. My investigation led me to their GitHub repository, which provided a foundational understanding of the model and its application.

I explored the YOLOv8 documentation and various technical resources, including StackOverflow discussions, to detect a person and track their position within video frames. This research was crucial in learning how to extract specific coordinates (x1, x2, y1, y2) representing the detected person’s location in the frame.

Further experimentation resulted in the development of a formula with my colleague Zach, enabling us to calculate the distance between the person and the car’s camera. This formula, m * (1/dataY) + par, where dataY is derived from the difference between the x-coordinates, allows us to get the distance in feet.

Our testing process involved gathering data on a person's position relative to the camera at various distances—5, 10, 15, and 20 feet. This data helped us refine our detection and tracking algorithms. To enhance safety, I introduced a conditional operation in the system: if a person is detected within an 8-foot radius of the car, the system triggers a safety protocol to prevent potential collisions.

This research not only improved the system's efficiency but also its reliability in different environments, demonstrating the practical benefits of integrating advanced object detection technologies like YOLOv8.

Path finding using GPS

The majority of the research on path finding using GPS occurred during last semester on how GPS works and how to use it. It was found that GPS works by having a receiver determine its own location by measuring the time it takes for a signal to arrive at its location from at least four satellites which can be seen in figure 2. The primary goal of the research of GPS for last semester and this one involved improving the precision of location data through advancements in satellite technology, signal processing, and algorithms.

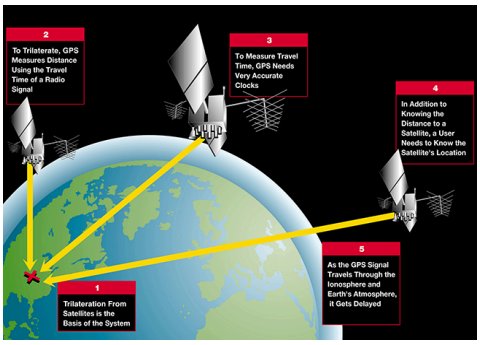

Fig. 2: Representation of GPS using satellites

Now last year's research focused mainly on using NMEA sentences to portray GPS on the car itself. This was done by the gps chip taking the raw satellite data and converting it into NMEA sentences similar to the ones seen in figure 3. Then by traversing through these sentences the values of latitude, longitude, time and data can be extrapolated. This latitude and longitude was then used to create a general map of the planned route that the car would drive resulting in figure 4. This though had an inaccuracy of about 10 meters proving to need improvements to the type of GPS used to get the location of the car. This is what led to the research done this semester to help improve this accuracy and precision.

```
$GNRMC,181722.00,A,4000.1256100,N,08301.5461206,W,0.000,,270718,,,A,V*0F
$GNVTG,,T,,M,0.000,N,0.000,K,A*3D
$GNGNS,181722.00,4000.1256100,N,08301.5461206,W,AANN,17,99.99,221.231,-33.698,,,V*1F
$GNGGA,181722.00,4000.1256100,N,08301.5461206,W,1,12,99.99,221.231,M,-33.698,M,,*4A
$GNGSA,A,3,03,06,12,17,19,24,28,02,,,,,99.99,99.99,99.99,1*37
$GNGSA,A,3,74,84,66,82,73,83,68,67,80,,,,99.99,99.99,99.99,2*3F
$GNGLL,4000.1256100,N,08301.5461206,W,181722.00,A,A*67
$GNGRS,181722.00,1,1.7,0.2,-1.3,-1.3,-1.5,1.8,2.5,1.2,,,,,1,0*7E
$GNGRS,181722.00,1,-4.2,0.8,4.2,1.6,-1.7,2.7,-4.8,-1.4,4.6,,,,2,0*74
$GNGST,181722.00,9.6,,,,0.71,0.71,0.71*51
$GNZDA,181722.00,27,07,2018,00,00*7E
$GNGBS,181722.00,0.7,0.7,0.7,,,,,,*57
```

Fig. 3: NMEA sentences

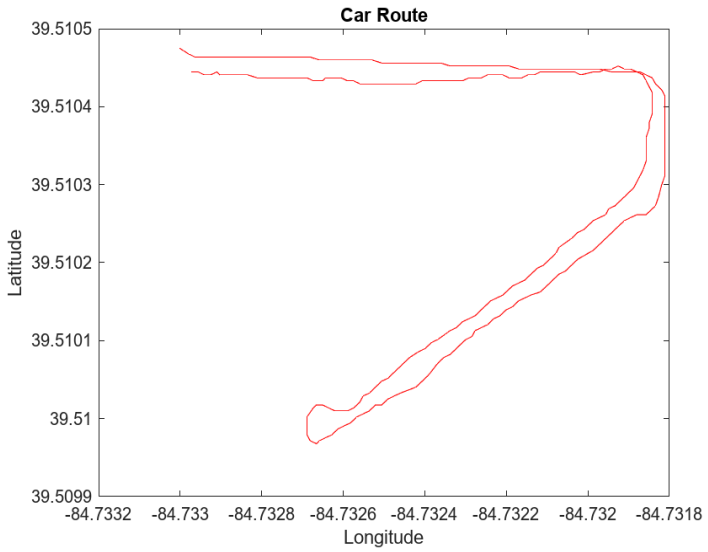

Fig. 4: GPS route with NMEA

When looking at a way to improve the accuracy and precision of the GPS from the previous semester it was found the UBX was the best outcome. When researching UBX for GPS it was found that UBX uses a software called U-center for programming and setting up the values needing to be received from the GPS. Now in order for the use of the U-center to work on a gps chip is to connect the chip directly to the computer without an arduino. Now for the setup on the car the gps chip needs to be connected to the arduino mega in order for the car to use gps data. When looking at a way to solve this two videos were found that provided a solution to this and more detail into how UBX works and how to use it. These youtube videos by iforce2d were insightful for how to use UBX to get a more precise and accurate latitude and longitude.

Solution Process

Path finding using GPS

Now the first part of the solution process for Path finding with GPS is looking into the different message types that are used in the U-center software, this being primarily NMEA and UBX. Now according to iforce2d's video the message type for UBX that is needed for GPS is the NAV messages where NAV stands for navigation. The NAV message has multiple data types at its disposal for the different values that are available from the GPS chip. The first data type that was looked at was the NAV-POSLLH or the Geodetic Position. This type of GPS targeting is done through geodetic coordinates, which are a type of curvilinear orthogonal coordinate system used in geodesy based on a reference ellipsoid, as seen in Figure 5. Now the gps first needs to be set up to receive this data type which is normally done in the U-center software but in this case is done manually in Arduino. This is done by sending byte messages patterns seen in figure 6 that turn off NMEA messages and any others and turn on the NAV-POSLLH messages. In this way the gps chip will only look for the NAV-POSLLH and not the others. After these messages are sent to the gps chip a structure is created to form the variables including latitude, longitude, and height.

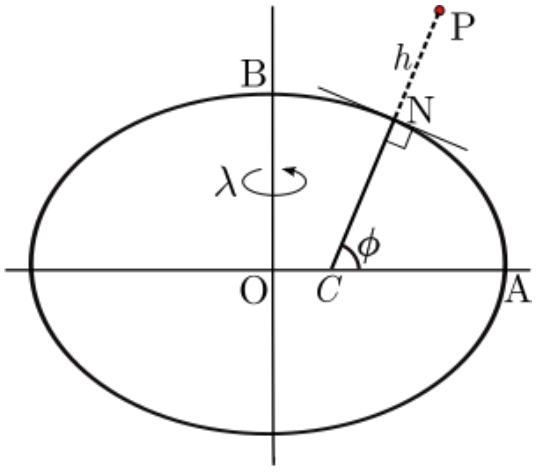

Fig. 5: Geodetic coordinates P(ϕ,λ,h)

```
const char UBLOX_INIT[] PROGMEM = {
  // Disable NMEA
  0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x24, // GxGGA off
  0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x01,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x2B, // GxGLL off
  0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x02,0x00,0x00,0x00,0x00,0x00,0x01,0x02,0x32, // GxGSA off
  0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x03,0x00,0x00,0x00,0x00,0x00,0x01,0x03,0x39, // GxGSV off
  0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x04,0x00,0x00,0x00,0x00,0x00,0x01,0x04,0x40, // GxRMC off
  0xB5,0x62,0x06,0x01,0x08,0x00,0xF0,0x05,0x00,0x00,0x00,0x00,0x00,0x01,0x05,0x47, // GxVTG off

  // Disable UBX
  0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x17,0xDC, //NAV-PVT off
  0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x12,0xB9, //NAV-POSLLH off
  0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x13,0xC0, //NAV-STATUS off

  // Enable UBX
  //0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x07,0x00,0x01,0x00,0x00,0x00,0x00,0x18,0xE1, //NAV-PVT on
  0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x02,0x00,0x01,0x00,0x00,0x00,0x00,0x13,0xBE, //NAV-POSLLH on
  //0xB5,0x62,0x06,0x01,0x08,0x00,0x01,0x03,0x00,0x01,0x00,0x00,0x00,0x00,0x14,0xC5, //NAV-STATUS on

  // Rate
  //0xB5,0x62,0x06,0x08,0x06,0x00,0x64,0x00,0x01,0x00,0x01,0x00,0x7A,0x12, //(10Hz)
  //0xB5,0x62,0x06,0x08,0x06,0x00,0xC8,0x00,0x01,0x00,0x01,0x00,0xDE,0x6A, //(5Hz)
  0xB5,0x62,0x06,0x08,0x06,0x00,0xE8,0x03,0x01,0x00,0x01,0x00,0x01,0x39, //(1Hz)
};
```

Fig. 6: UBX byte messages

Before the structure variables are used at all it goes through a GPS data reception and validation routine. It continuously reads incoming GPS data, checks for a specific header sequence, collects a payload, calculates a checksum for the payload, and validates the checksum against the received bytes. If the received data and checksum are valid, it returns true, indicating successful data reception and validation. Then the data can simply be collected by going through the variables of the structure where latitude and longitude can be represented by long values. With the process of the GPS data reception and validation routine written and proven to work it was determined to use NAV-PVT, also known as Navigation PVT Solution, as it contains accessible variables seen in figure 7. PVT works by which a GNSS receiver measures the transmitting time of GNSS signals emitted from four or more GNSS satellites and these measurements are used to obtain its position and reception time.

```
struct NAV_PVT {
  unsigned char cls;
  unsigned char id;
  unsigned short len;
  unsigned long iTOW;          // GPS time of week of the navigation epoch (ms)
  unsigned short year;         // Year (UTC)
  unsigned char month;         // Month, range 1..12 (UTC)
  unsigned char day;           // Day of month, range 1..31 (UTC)
  unsigned char hour;          // Hour of day, range 0..23 (UTC)
  unsigned char minute;        // Minute of hour, range 0..59 (UTC)
  unsigned char second;        // Seconds of minute, range 0..60 (UTC)
  unsigned char valid;                 // Validity Flags (see graphic below)
  unsigned long tAcc;          // Time accuracy estimate (UTC) (ns)
  long nano;                   // Fraction of second, range -1e9 .. 1e9 (UTC) (ns)
  unsigned char fixType;       // GNSSfix Type, range 0..5
  unsigned char flags;                 // Fix Status Flags
  unsigned char flags2;     // reserved
  unsigned char numSV;         // Number of satellites used in Nav Solution
  long lon;                    // Longitude (deg)
  long lat;                    // Latitude (deg)
  long height;                 // Height above Ellipsoid (mm)
  long hMSL;                   // Height above mean sea level (mm)
  unsigned long hAcc;          // Horizontal Accuracy Estimate (mm)
  unsigned long vAcc;          // Vertical Accuracy Estimate (mm)
  long velN;                   // NED north velocity (mm/s)
  long velE;                   // NED east velocity (mm/s)
  long velD;                   // NED down velocity (mm/s)
  long gSpeed;                 // Ground Speed (2-D) (mm/s)
  long headMot;                // Heading of motion 2-D (deg)
```

Fig. 7: NAV_PVT structure

Object Detection Using YOLOv8

The primary objective for the Raspberry Pi was to capture video frames in real time and transmit them to a server designated for object detection. To achieve this, the Raspberry Pi utilizes the cv2 library to interface with the camera, configuring settings to capture frames at 640x480 resolution and 5 fps. This balance between quality and performance was crucial, given the computational limitations of the Raspberry Pi. Once captured, each frame is converted into PIL Image format, then compressed into JPEG via a bytes buffer to minimize the frame size, facilitating more efficient transmission over the network. A TCP socket connection is established between the Raspberry Pi and the server, allowing for the serialized JPEG data and its size to be sent and ensuring the server can correctly get the incoming data.

On the server side, the setup involves initializing a socket to listen for connections from the Raspberry Pi. The server employs a function, receive_all, to reliably receive and reconstruct serialized frame data into image format. The core functionality revolves around the YOLOv8 model, which processes the frames to detect objects. The detect_persons function specifically targets person detection, calculating distances and other relevant metrics from the detected objects. The server then sends these detection results back to the Raspberry Pi, which may use this data for various

actions, such as writing to a serial port or adjusting vehicle operations based on the proximity of detected objects.

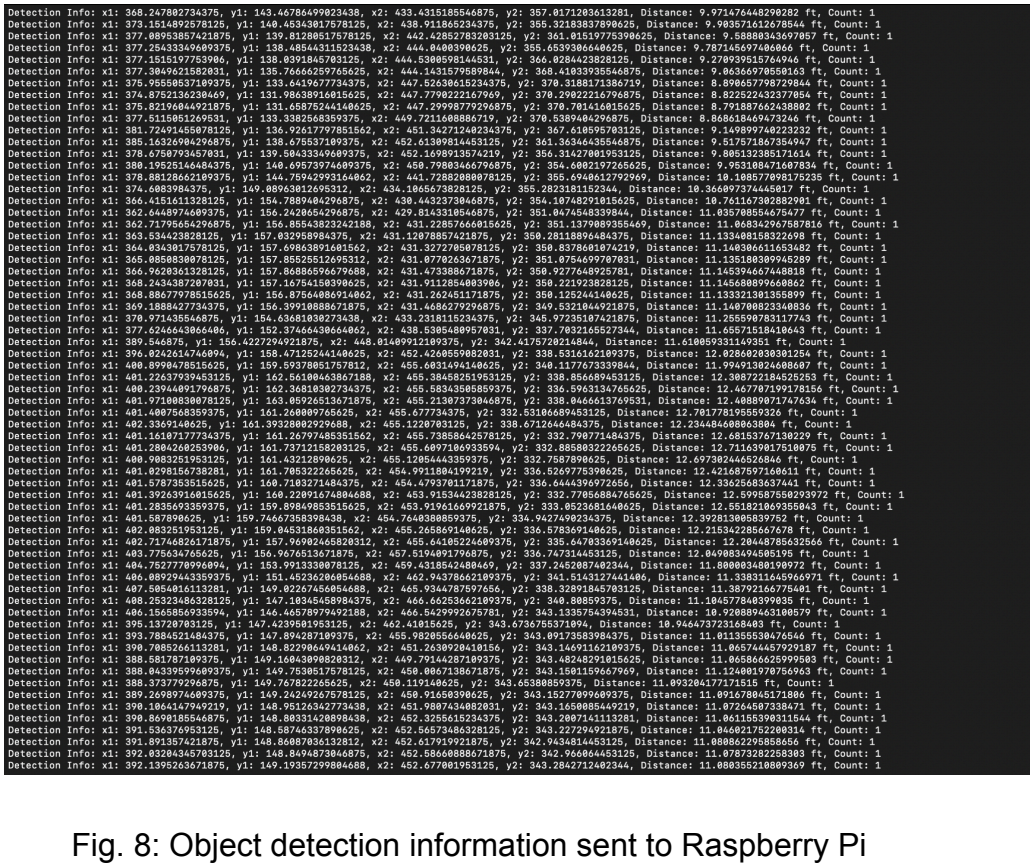

Fig. 8: Object detection information sent to Raspberry Pi

Person Tracking

To first start off with even attempting to have the car follow a person data first needs to be taken from the camera's object detection. By reading the serial monitor and putting every new instance of a line into a string variable called inputString so the camera data can be saved to the arduino. Now the next step is to separate the inputString variable into different variables for x1, x2, y1, y2, and distance. This is done by first finding the instance at which each variable value is at and putting those indexes into different variables. With these indexes the substring function can be used to get each variable in the overall string where the values at the end are separated by a comma. These smaller strings then can be converted to either an integer value or float value as seen in figure 9.

```
int x1Index = values.indexOf("x1:") + 3; // Add 3 to skip "x1:"
int y1Index = values.indexOf("y1:") + 3; // Add 3 to skip "y1:"
int x2Index = values.indexOf("x2:") + 3; // Add 3 to skip "x2:"
int y2Index = values.indexOf("y2:") + 3; // Add 3 to skip "y2:"
int distIndex = values.indexOf("Distance:") + 10; // Add 10 to skip "Distance:"

// Extract and convert substrings to integers
int x1 = values.substring(x1Index, values.indexOf(",", x1Index)).toInt();
int y1 = values.substring(y1Index, values.indexOf(",", y1Index)).toInt();
int x2 = values.substring(x2Index, values.indexOf(",", x2Index)).toInt();
int y2 = values.substring(y2Index, values.indexOf(",", y2Index)).toInt();
float distance = values.substring(distIndex, values.indexOf(" ft", distIndex)).toFloat();
```

Fig. 9: Input separation

To find the middle of the x values that are given by the object detection code the x1 and x2 can be added then divided by two in order to get the general middle of the person's box. This middle will be the point that is tracked by the car and can be used for the car's turning system. Through testing through a hallway it was found that the minimum and maximum values for the camera from 8 meters away is from 172 to 462. Now these values with the input of x_d, or our middle point , can be used for turning by mapping them to the max left and right turning values in the car. By using the map function shown in figure 10 this can happen where in order for no turning to happen x_d has to be 320. Now if x_d is greater than 320 the car's wheels will turn right, varying on how far, and less than 320 to turn left. This map function can also be used for the distance variable where the farther a person is away from the car the faster the car will drive towards that person.

```
int turnX = map(x_d, 172, 462, -250, 250);
```

Fig. 10: Map function

Final Results

Path finding using GPS

Using the NAV-PVT data structure instead of the previous NAV-POSLLH allows for more variables to be used including the speed of the car and magnetometer data for direction. For the car the variables that are specifically needed are latitude, longitude, time constants of hour, minute, second, NED north velocity, NED east velocity, NED

down velocity, ground speed, heading of motion or magnetometer, and the accuracies of all these variables. These variables can help in the future development of a self-driving car path and a greater accuracy of the car's location. Now while using the NAV-PVT latitude and longitude values a straight path of the car is able to be obtained which is seen in figure 11. Now when looking at the car's path of figure 11, which has the same y and x limits as figure 4, and comparing it to the straight path on figure 4 it is clear that NAV-PVT's location data has more precision.

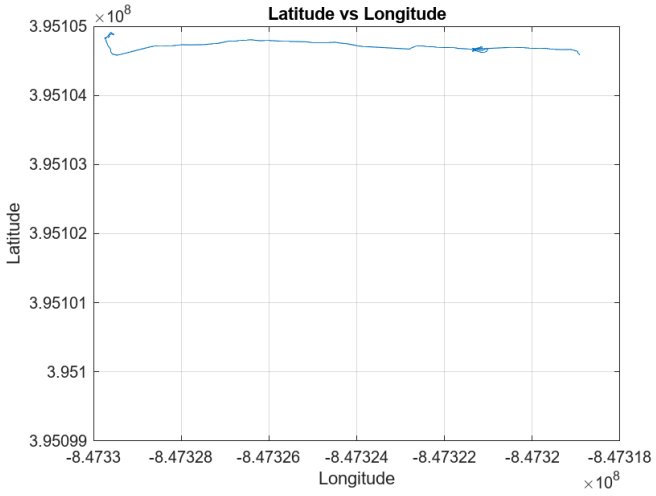

Fig. 11: Path with NAV-PVT

The other parts of the NAV-PVT still need to have more time spent on them to get more and better data points. These include the magnetometer, ground speed, and finally the accuracy of the variables that are collected. The accuracy variables can guarantee a way to determine which variables need improvements and which ones don't. The speed variables including ground speed will tie into future developments of the dead reckoning software used on the car. While these collections of data are not present the current data for latitude and longitude will help any future progress on the car itself. Coordinating with some of the work done on the person tracking part of the self-driving car will help in the future to create a self-driving path.

Object Detection Using YOLOv8

The final implementation of our autonomous detection system produced highly encouraging results, successfully integrating real-time video capture, efficient data transmission, and accurate object detection. The integrated system, consisting of the Raspberry Pi, the object detection model utilizing YOLOv8, and the network communication infrastructure, performed outstandingly. The system consistently maintained a frame rate of 5 fps at a resolution of 640x480 pixels, aligning with our processing needs and achieving our target frame rate. However, while the object detection model achieved an impressive accuracy rate in detecting individuals within the video frames, there is room for improvement. Developing a new model specifically focused on detecting people could enhance this accuracy further.

Regarding speed, the system managed a preprocessing time of 1.3ms and a postprocessing time of 0.5-0.7ms. The most critical component, the inference time, ranged between 65.3ms and 68.2ms. This reflects the time taken by the model to analyze an image and make predictions, coming to a total object detection speed of about 70ms per frame. The system can process approximately 14-15 frames per second. Despite these successes, the quality of the camera and the calibration of the RGB channels could be enhanced to improve the overall image clarity and accuracy of detection. Network communication between the Raspberry Pi and the server works tremendously. The system demonstrated its capability to track individuals reliably over various tests, accurately calculating distances up to 20 feet. The system effectively triggered safety protocols when a person was detected within an 8-foot radius of the autonomous vehicle. Feedback from these trials highlighted the system’s potential to enhance safety and operational efficiency in autonomous vehicle navigation.

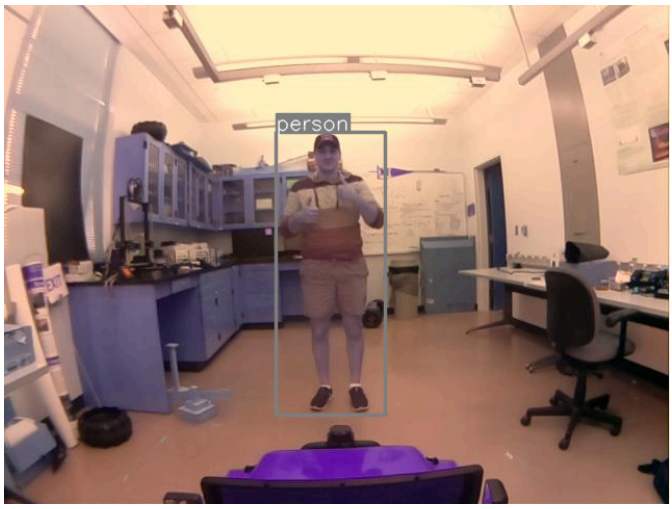

Fig 12: Object Detection of a person

Person Tracking

The final factors of person tracking that were implemented was being able to detect when to stop the car itself and the final calculations of steering that the car uses. To start with having the car stop the inputString variable looks for a specific input on the serial monitor that comes from the raspberry pi. The input that is being looked for is the string “Stop”. Now when the inputString variable is equal to this the car’s speed is set to 0 and its turn is set to straight. Now this will happen when the car or more accurately the camera on the car is approximately 8 meters away from any person in the car’s view. There are also a few more else if statements that set the car’s speed to 0 including there being no inputs in the serial monitor and if there is a “null” in the serial monitor. Now there still needs to be more calculation done in order to turn the turn variable into a value that the car’s 3d parts can use shown in figure 13.

```
if(revTurn == 1)
{
  desA = steerA - steerB * tan(PI * turnX / 750.);   //max is pi/3
}
else
{
  desA = steerA + steerB * tan(PI * turnX / 750.);
}
ST_PWM = floor(steerKp * (desA - pot) + .5);
```

Fig. 13: Steering calculations

Using all these final variables and detection function a proper person tracking software is made. As seen in figure 14 the car is able to track the direction of a single person in whichever direction they are. The car will correct itself, turning either left or right, spending on which side the person is on the camera and will face straight forward towards the person. Depending on how far a person is away from the car it will increase its speed to reach the person and slow down the closer it gets. While this tracking method works well in the hallway that is where it was tested it works even better in an outside environment where it is able to turn more without hitting any walls.

Fig. 14: Car following

Future Work

Path finding using GPS

It can be seen from looking at the close up view of the NAV-PVT path, seen in figure 15, that it still doesn't have the greatest accuracy and still needs to be improved upon. Now there are many ways that this can be done by adding new elements into the gps systems of the car. One of these ways of improvement is by introducing differential gps into the system. Now differential gps is a gps technique that utilizes two gps chips instead of just relying on the one attached to the car. In this technique in addition to the gps chip attached to the car there is another chip that acts as a ground chip that remains stationary. Since the ground chip's location never changes it can be used to

compare the distance between the two gps chips and allow for a more accurate location. The next thing that can be implemented is the already made dead reckoning algorithm. This will help to determine the difference between the actual data points taken from latitude and longitude and compare it to the dead reckoning algorithm. With this the data can be compensated to represent the dead reckoning algorithm more.

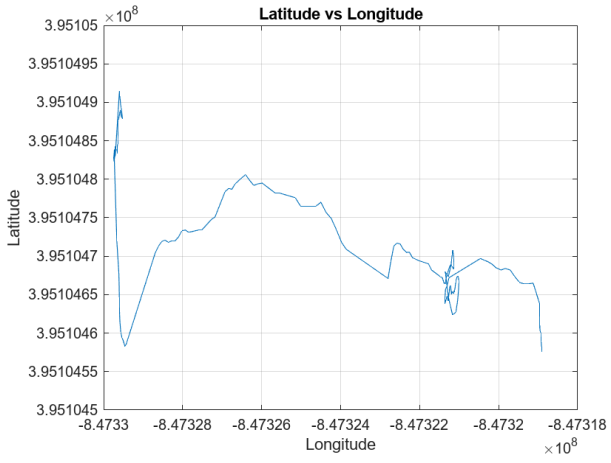

Fig. 15: Close view NAV-PVT path

Now the magnetometer can help to keep the car straight on the sidewalk keeping it from swerving off the sidewalk. The magnetometer first needs to be calibrated first in order to get correct data sets. This can be done by driving the car in a circle to get a 360 degree data circle which is likely to be offset due to influence by metal. The degrees can then be calibrated to be an accurate circle where north is 0 degrees, east is 90 degrees, south is 180 degrees, and the point just before north is 359 degrees.

The final goal for path finding with gps is for the car to be able to drive through a predetermined path without any human aid. This includes the car being able to drive straight and keep aligned while driving. It also includes the turning aspects of the path including knowing when to turn onto the crosswalk and how to drive in a circle to turn around. The hope is to also have a greater accuracy of gps data in order for the car to know which side of the sidewalk it is currently on. The accumulation of all this work will allow the car to complete this final goal and eventually it will be able to drive without a planned path.

Object Detection Using YOLOv8

The integration of object detection allows for a continuous loop of real-time object tracking and decision-making, which is crucial for autonomous systems. However, challenges such as maintaining real-time processing speeds, ensuring data integrity over potentially unreliable networks, and specifically locking onto the first detected person despite the presence of others pose significant hurdles. These challenges involve advanced tracking logic and potentially the integration of advanced tracking algorithms like Kalman filters to follow the initially detected person consistently. Refining the balance between frame rate and resolution based on experimental data will be crucial for optimizing system performance in future developments.

Further advancements should also consider the development of a specialized YOLOv8 prediction model tailored specifically for human detection. By focusing the model on detecting people and any other important object we want to detect like signs, crosswalks, etc., we can significantly improve its efficiency and accuracy for our specific use case, as the current base model of YOLOv8 encompasses multiple classes that are not necessary for our application.

Upgrading the camera technology used in our system could greatly benefit our objectives. Implementing a dual-lens camera would enhance the accuracy of distance measurements and facilitate more sophisticated tasks such as edge detection for sidewalks. This enhancement would be instrumental in developing a 'vanishing point' program for the car, improving its navigational capabilities in different environments. These technological improvements will pave the way for more advanced and reliable autonomous vehicles.

Person Tracking

Conclusion

Path finding using GPS

This part of the project explored the ways in which to accurately receive location using methods of gps and implementing them for use in a self driving car. Gps is a

critical tool used for this project in ways of location finding and the car's own location detection. This can be said for the many other objects that in fact use gps in an everyday concept. The outcomes that have come out of using gps in the specific part of path finding has helped and will help every other aspect of the car. Now increasing the accuracy and precision of this gps data will guarantee that the car's crosswalk algorithm, object detection, and even person tracking will be used at the correct times and places. Gps will also allow for the correct detection of where intersections, roads and stops are located at. This all allows for the correct placement and safety of the car itself when it does eventually drive without any human interactions.

Object Detection Using YOLOv8

This project made significant strides in integrating advanced object detection into an autonomous car. By using the YOLOv8 model and the Raspberry Pi, we were able to capture and process video in real time effectively.

Throughout the project, I have successfully implemented YOLOv8 for real-time object detection, which is crucial for navigating the car safely. The Raspberry Pi played a significant role in capturing video frames and handling data transmission to a server. Using socket programming was crucial in establishing reliable communication between the Raspberry Pi and the server, ensuring that data was transmitted efficiently, which is important for maintaining the system's responsiveness.

This project's results show that integrating object detection technology can enhance the functionality of autonomous vehicles. It has also opened up new opportunities to advance our system further.

Person Tracking

This project has made a great step in integrating a tracking system into an autonomous car. By using serial communication with the raspberry pi via the serial monitor the tracking system successfully connects with the camera. This system has been able to turn to face a person in each direction, speed up to catch up to the person,

and stop when the person gets too close to the car. The only improvements that could be done on this system would be to implement the crosswalk algorithm and lidar when used outside to be able to stay on the sidewalk and off the grass. This tracking system can also be used in the path finding part of the project to not run into people and be able to know how to turn with the car.

Works Cited

https://www.youtube.com/watch?v=ylxwOg2pXrc&feature=youtu.be

https://www.youtube.com/watch?v=TwhCX0c8Xe0&t=11s

Socket - low-level networking interface. Python documentation. (n.d.-a). https://docs.python.org/3/library/socket.html

Socket programming how to. Python documentation. (n.d.-b). https://docs.python.org/3/howto/sockets.html

Ultralytics. (n.d.). *Ultralytics/ultralytics: New - yolov8 🚀 in PyTorch > ONNX > OpenVINO > CoreML > TFLite*. GitHub. https://github.com/ultralytics/ultralytics