AutoPark - Autonomous Parking Golf Cart

User Manual

Team YoungWoods
Elijah Schwartz, Eunice Tan, Juan Garza, Nick Thevenin, Nico Rossi, Oscar Reynozo, Son Nguyen, Vedant Patwari

ACKNOWLEDGEMENT

We would like to thank Dr. Joseph Young and Dr. Gary Woods for their mentorship in this project, the Rice OEDK for providing resources and support, as well as the Rice University Parking Office for providing testing space. We are also very grateful to Mark Arbore and Laura Smoliar for providing funding for the golf cart.

ABOUT THE 2022 - 2023 TEAM

The 2022-2023 team is composed of 8 talented electrical engineering students, each bringing unique skills and perspectives to the project. The team is subdivided into 3 subteams, namely embedded, machine learning, and vehicle control, and each student is assigned to at least one of those subteams.

The embedded subteam focuses on the development of hardware and software systems that are essential to the golf cart's operation, while the machine learning subteam works on the algorithms and models that enable the golf cart to park autonomously. The vehicle control subteam is responsible for designing and implementing the control system that regulates the cart's movement.

Although the 2022-2023 team is divided into three subteams, they work side by side and closely together to achieve the project's objectives. The embedded, machine learning, and vehicle control teams collaborate and communicate frequently to ensure that their work is cohesive and aligns with the overall project goals. For example, the embedded team's work on the ultrasonic sensors to ensure safety distance from objects is critical for the vehicle control team's motor control system for brake, throttle, and steering. Similarly, the machine learning team's work on implementing DonkeyCar for machine learning aligns with the embedded team's work on

developing hardware and software systems that are essential to the golf cart's operation. By working together, the subteams can leverage each other's strengths and overcome challenges more effectively, ultimately leading to a successful project outcome.

left to right: Nick, Oscar, Elijah, Son, Juan, Nico, Dr. Woods, Eunice, Vedant

Table of Contents

I. Introduction

The purpose of this user manual is to provide readers with the necessary information to become acquainted with and understand the design process of Project AutoPark. The team compiled all the work done in the 2022-23 school year to pass off the project to a set of new engineers, hoping they could build and improve upon the work already done. This manual provides qualitative descriptions of all the project's systems and the technical process of how the systems were brought up and implemented. The goal is that anyone who is interested in working on Project AutoPark should be able to read this manual and have knowledge of the end-to-end engineering of the project and contribute in a meaningful way, even with no prior knowledge of the project.

The final working model engineered by Team YoungWoods was a full-scale golf cart that featured vehicle control (steering, throttle, and braking) driven by motors in various mechanical systems. Machine learning and computer vision were leveraged to autonomously control the vehicle based on input data from attached sensors, such as a stereo camera. Microcontrollers were used to interface and connect the subsystems: serving as a communication hub for the entire project's technology stack. More information on the systems and how they operate will be provided in detail later in the manual.

In this manual, you will find all the working code used to operate the project's systems. As AutoPark was born from a team of electrical engineering students, and features many complex state-of-the-art architectures, one should have knowledge and experience in the following areas: embedded systems & programming, machine learning, computer vision, PCB design, and classical mechanics.

1.1 Important Repositories + Links

Repositories:

- All repositories should be linked in this main repository:
    - https://github.com/nicolascrossi/AutoPark
- BeagleBone Black:
    - Control Loop: nicorossirice/rcPWM (github.com)
    - EthernetAPI: nicorossirice/EthernetAPI (github.com)

- Jetson Nano:
    - Current Fork of the DonkeyCar: https://github.com/nicorossirice/YoungWoodsCar
    - Our YoungWoodsCar working folder: phuson135/YoungWoodsCar (github.com)
- Everything Arduinos related:
    - phuson135/YoungWoodsArduinos (github.com)
    - https://github.com/ett2/autopark/tree/cec73dc86fb7ccf2ed5b886c2b8d354f66ad9d7c ← use final design folder
- Donkey Car Documentation:
    - https://docs.donkeycar.com/utility/donkey/

II. Safety & Maintenance

2.1 Electrical (Hardware) Components

Here's a bulleted list of the safety measures to take when working with electrical components in the golf cart:

- Cover the Golf Cart: When not operating the golf cart, cover it with a tarp to prevent water from getting onto the electrical components. Make sure to secure the tarp onto the golf cart by bungee cables.
- Avoid Water Exposure: Keep electrical components dry and avoid exposure to water, as moisture can cause damage, corrosion, and short circuits.
- Check for Moisture: Regularly inspect electrical components for signs of moisture or corrosion.
- Disconnect Power: Before performing any maintenance or repairs on electrical components, disconnect the power source to avoid electrical shock or damage to the components.
- Follow Manufacturer Recommendations: Always follow the manufacturer's recommendations for maintaining and using electrical components referring to the golf cart’s original electrical design.
- Install a Fuse: When working with the two 12-volt batteries, install a fuse in the circuit to prevent overloading and battery-related accidents.
- Be Careful with the Steering Motor: Avoid stalling the steering motor for a prolonged time, as this can generate heat and cause the motor to burn out.

By taking these safety measures, you can help ensure that the golf cart's electrical components remain safe and functional for a long time.

2.2 Software Components

It's important to always have a backup of your code in case of any unexpected incidents that could lead to data loss. One way to ensure this is to use a source control method like GitHub or any other version control system.

Here are some safety precautions to consider:

- Back up your code regularly: Always make sure to back up your code regularly, preferably after each significant change, to ensure that you have the most up-to-date version in case of any data loss.
- Store your software-related data in a safe place: Make sure you have a secure and reliable storage location for all your software-related data. This can be a cloud-based storage system, an external hard drive, or any other reliable backup storage.
- Document as much as possible: It's essential to document your code as much as possible. Documenting your code helps you and your team understand what each line of code does and why it was implemented. This also makes it easier to troubleshoot any issues that may arise in the future.
- Use descriptive commit messages: When making changes to the code base, always use descriptive commit messages that explain the changes made. This helps you and your team understand what was done, why it was done, and when it was done. This also makes it easier to revert to a previous version of the code if needed.
- Use diagrams to illustrate changes: Diagrams can be incredibly helpful in illustrating changes made to the code base. Use diagrams to show how the code works and what changes were made to the code. This helps you and your team understand the code better and makes it easier to troubleshoot any issues.

By following these safety precautions, you can ensure that your code is always up to date, and you have a reliable backup in case of any data loss.

2.3 Mechanical Components

- Regularly inspect mechanical components for signs of wear and tear, damage, or defects before you take the golf cart for a drive or train.
- Make sure to have two or three backups of important mechanical parts, if they are easy to make, just in case they break, you can replace them on the go.
- Have a dedicated team member looking out for the worse case scenario if a part breaks.
- Keep the work area clean and free of clutter to avoid accidents or unnecessary pain. This is very important as it helps you layer down the road when things become more complicated.
- Keep in mind an emergency plan in place in case of accidents or mechanical failures.

2.4 Training/ Testing

Certain precautions must be taken at all times in order to ensure the safety of both the golf cart user and the golf cart equipment, surroundings and bystanders. These include, but are not limited to:

1. Having at least one person in the cart at all times while the machine learning model is running.
2. Ensuring that all batteries and power cords are secured and that no open wires are exposed.
3. Inspecting wiring and motor mounts before use to verify their integrity.

2.5 Maintenance

The maintenance procedure that was followed for the golf cart, and that we recommend for any future users, is as follows:

1. Covering the golf cart and all electrical components with a tarp and bungee cord to prevent water damage.
2. Waterproofing component casings as regularly as needed.
3. Enhancing motor mounts when deemed necessary, and 3D printing spare parts in case of failure.
4. Regularly charging the golf cart batteries, 12V batteries, and 5V battery packs.
5. Placing cinder blocks behind golf cart tires whenever the golf cart is not in use.

2.6 Maintenance

In order to test the motor components of the golf cart, it was often necessary to jack the golf cart. Our golf cart operates on RWD, meaning that the rear end of the golf cart must be raised if testing the golf cart throttle. When testing the steering component, we recommend avoiding raising the rear wheels, as additional stress will be placed on the front wheels while steering. We

believe that this caused one of our steering motors to short. The jack points that were used are shown below: Elijah Schwartz

III. Microprocessors & Microcontrollers

3.1 Jetson Nano

The Jetson Nano's function in our project is to run our Donkeycar machine learning model to predict steering and throttle angles.

Username: youngwoods

password: nvidia

This guide will assume that you know the basics of Linux OS, if not please find resources online that will help you with getting to know the basic operations in Linux OS. Make sure to refer to this link regularly for how to use features in donkeycar.

Drive the golf cart for data collection or testing models.

Make sure the Front Camera and the Xbox-controller are connected to the Jetson Nano

- Step 1: login into youngwoods
- Step 2: *cd* **/YoungWoodsCar/**
- Step 3: *python manage.py drive --js*
- Step 4: Follow this guide for further information.

Any questions about this step could be answered by contacting Son Nguyen via his email (phuson135@gmail.com) please use the header YoungWoods.
Notice the **YoungWoodsCar/** folder is backup by Github linking to my repository right now at https://github.com/phuson135/YoungWoodsCar
The current donkeycar version we used in our project is forked into this repository: https://github.com/nicorossirice/YoungWoodsCar

Jetson Nano GPIO: How to Use GPIO Pins on Jetson Nano Developer Kit | Nvidia Jetson | Maker Pro
Jetson Nano could be powered via the two GPIO 5V power pins at 2.5A each (Jetson Nano 2GB Developer Kit User Guide | NVIDIA Developer).

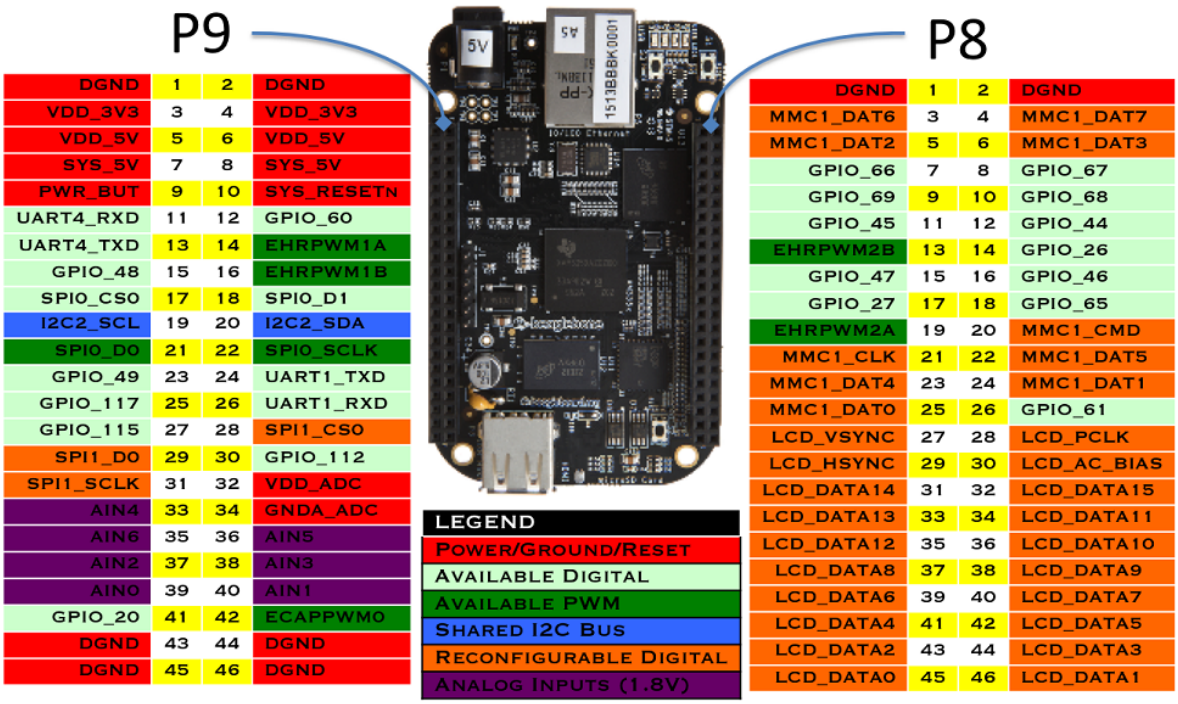
You can treat Jetson Nano as a laptop by plugging in an HDMI or DP cable to it to see what is running on the Jetson Nano.

Communication between Jetson and other components:

- Fob State Driver: via two GPIO pins (2 bits total for 4 different states). These GPIO pins cause an interrupt on a High to low or low to high edge allowing the Jetson to read the sent state (emergency stop, parking, idle, reverse parking). One way communication from fob state driver to Jetson Nano
- Reverse Arduino: via two GPIO pins, one allows Jetson to switch direction the other lets the Jetson know that the switch has been completed or not.
- XboxController: via USB cable, could use a Bluetooth adapter in the future for this, make sure to by one that come out of the box with support for Linux. The buttons are programmable in my_joystick.py file under /YoungWoodsCar to do many things within the context of DonkeyCar environment. In our project we used it to switch states (act as the fob), emergency button, initiate reverse/ forward switch command.
- Two Cameras (Front/ Back): via USB cable, the frames from these camera are capture according to which state you are in (Forward/ Reverse).

3.2 BeagleBone Black

Cape Expansion Headers



In this project, we use it as another level of abstraction to offset all the throttle, steering control loop into another single chip computer so the Jetson Nano task is only to do Machine Learning on the input image frames and spit out throttle/brake as well as steering command to the BeagleBone Black via EthernetAPI (nicorossirice/rcJetson (github.com)) communication. The BBB armed with other sensor information such as CV value or Emergency brake button input or Current Vehicle Speed, can make the necessary decision to fine tune the control loop.

- EthernetAPI: an ethernet communication scheme created by Nico Rossi for this project allowing the Jetson Nano to send information via the Ethernet Cable to the BeagleBone Black. In this project, we use this communication protocol to communicate throttle/brake as well as steering command from the Jetson to the BeagleBone Black. Check out nicorossirice/rcJetson (github.com) for more information and examples.
- Computer Vision Laptop to BBB communication: done via UART4 (P911 and 13), essentially CV sends a multiplier (between 0 to 1) inversely proportional to the distance

of objects it detects to the BBB. The BBB could then slow down the drive loop by applying this multiplier on the original throttle/brake command.

- Speed Encoder Arduino to BBB communication: done via UART1 (P924 and 26), essentially the Arduino sends two number in the form of "S{current_speed}|{current_acceleration}". Look at this code for more detail as to how it does it (YoungWoodsArduinos/SpeedEncoder.ino at main · phuson135/YoungWoodsArduinos (github.com))
- BBB to Throttle/Brake Arduino communication: done via I2C2 (P919 and 20), directly set what throttle or brake position we want the pedals to be at.
- BBB to Steering Arduino communication: done via I2C2 (P919 and 20), essentially directly send what steering position we want the steering wheel to be at.

BBB logins:

- ssh debian@192.168.7.2
- Password: **tempwd**

BBB login from within the Jetson Nano.

- Make sure you are in the YoungWoodsCar folder and the Ethernet cable is connected.
- then do **./BBB** to ssh into the Beaglebone Black via the Jetson Nano terminal.

Notice the BeagleBone black control loop code is located inside folder **/rcPWM**

- source controlled by nicorossirice/rcPWM (github.com)
- Contact Nico Rossi for more information about this github.

3.3 Arduinos

Our Arduinos Codes are source controlled here: phuson135/YoungWoodsArduinos (github.com)

- SteeringControl.ino: source code for the Arduino controlling the steering motor.
- ThrottleAndBrakeControl.ino: Source code for the Arduino controlling both Throttle Motor and Brake Actuator.
- LEDs.ino: Source code to display on the side color strips for throttle/brake vale and the middle color strip for steering value.
- SpeedEncoder.ino: Source code to take spit out encoder value of the golf cart motor in the form of "S{speed}|{acceleration}"

Other stuff is here:

https://github.com/ett2/autopark/tree/cec73dc86fb7ccf2ed5b886c2b8d354f66ad9d7c

Look at the final design folder for everything that is relevant.

- transceiver: fob
- receiver: state driver
- ultrasonictest: ultrasonic sensor stuff

The receiver and the ultrasonic sensor stuff are connected by a flag signal from the ultrasonic sensor module (see PCB schematic if you have questions)

IV. Subsystems

4. 1 Reverse:

- The reverse uses an Arduino connected to a motor controller, there are four pins from arduino to motor controller → En ENA, ENB and GND
    - These pins are responsible for your direction and whether the motor controller is enabled (the motor controller can control two motors at once)
- The Arduino will have two pins from the two different sensor
    - These sensors can be buggy, the exact position of the sensors is vital such that the motor does not keep trying to move when its already at max position, this can cause a fire or burn the motor/ wires because the motor will try to reach its position
- In the Arduino code there are two states, one for forward and reverse, when you
- The Arduino has a command pin input and request pin output to the jetson.
    - The command pin high corresponds to reverse and Low to forward, keep in mind how you set this to high in the code because if you do high current out you could break jetson
    - The request pin is high when arduino detects when in forward state or reverse and that it is consistent with the command from jetson
- The states latch,
    - There is a forward and reverse state, whenever you want to go forward and the sensor detects any coverage, it will latch until you change command, this is because of noise from sensor and that the motor doesn't keep going when it detects noise and possibly burn something

4.2 Machine Learning:

4.2.I - Training the Model:

For our machine learning model, we use DonkeyCar since it already has custom CNN models that can be easily used and modified with different options. Additionally, all terminal commands provided are used in a Windows operating system as opposed to a Linux operating system. Once the training needed for your model is collected, we will use DonkeyCar on the desktop to train the model. First, all data collected from the Jetson Nano must be imported into the /AutoPark/YoungWoodsCar @ 6ca36d4/data/ directory of the Desktop as a single folder. The

easiest way to transfer the image data from the Jetson Nano to the desktop is by setting up MobaXterm on the desktop to communicate with the Jetson Nano.

If the data folder does not exist, it must be created for DonkeyCar can easily retrieve the training data for training the model. In addition, make sure that the /AutoPark/YoungWoodsCar @ 6ca36d4/models/ directory of the Desktop exists or create the folder for DonkeyCar can easily load the trained model into the folder. Once the training data is imported and the data and models directories are created, then make sure to cd into the /AutoPark/YoungWoodsCar @ 6ca36d4 directory. Now that you are in the /AutoPark/YoungWoodsCar @ 6ca36d4 directory, you can run the following command to train the model:

1. C:\location_to_YoungWoodsCar_directory\YoungWoodsCar activate donkey
2. C:\location_to_YoungWoodsCar_directory\YoungWoodsCar donkey train --tub ./data/data_folder --model ./models/model_name.h5

Within these commands, location_to_YoungWoodsCar_directory specifies the directory to the YoungWoodsCar folder where the model is trained. In the desktop in Ryon B12 that is below the YoungWoods shelve (storing components from this year), there exists a Desktop with DonkeyCar and Young where the exact directory to YoungWoodsCar is C:\Users\MECE3\Desktop\YoungWood2022\YoungWoodsCar. Moreover, the data_folder directory indicates the specific name of data folder where the training data is stored. If there are multiple training data folders used, you can alternatively add more training data folder locations as shown below:

C:\location_to_YoungWoodsCar_directory\YoungWoodsCar donkey train --tub ./data/data_folder_1 ./data/data_folder_2 --model ./models/model_name.h5

You can add as many training data folder or training data as desired. However, more data will significantly increase the training time. Lastly, model_name indicates the name of the model that will be trained and saved under the models directory. Once the model begins training, you must wait and monitor the training to ensure the model is properly trained.

4.2.II - Evaluating the Model:

Now that the model is trained, there are a few ways to evaluate the model. First, when the trained model is saved under the models directory, there is a .png image saved with the same name as the trained model. This .png image provides a loss plot of the training and validation data as the model is trained with more epochs (batches of training data). Aside from the loss plot, there are other metrics that can be used to evaluate the effectiveness of the trained model. One useful feature of DonkeyCar is that it can create a movie of the trained model predicting the steering angle and throttle value of given test data. Hence, for each labelled image of the test data, a vector with a magnitude and direction indicates the predicted movement of the golf cart overlaid with another vector indicating the actual movement of the golf cart. To generate the movie, the following command lines command could be used:

- donkey makemovie --tub=./data/test_data --out=video_name.mp4 --model=./models/model_name.h5 --end=1000 (For one test data folder)
- donkey makemovie --tub ./data/test_data_1 ./data/test_data_2 --out=video_name.mp4 --model=./models/model_name.h5 --end=1000 (For multiple test data folders)

In those commands, the test_data indicates the name of the test data folder used under the data directory, the video_name indicates the name of the output video stored under the YoungWoodsCar directory, and the model_name indicates the name of the trained model used under the models directory. Aside from making the movie, a predictions plot can be made to compare the predicted and actual steering angle and throttle values for a trained model on a test data set. To generate this plot, either command line command could be used:

- donkey tubplot --tub=./data/test_data --model=./models/model_name.h5 (For one test data folder)
- donkey tubplot --tub ./data/test_data_1 ./data/test_data_2 --model=./models/model_name.h5 (For multiple test data folders)

The test_data argument is name of the test data folder provided within the data directory and the model_name argument is the name of the trained model that is tested within the models directory. Thus, the predictions movie and plot provide an interpretable visualization of the model's performance with predicting test data.

Aside from evaluating performance, investigating the training data's distribution and CNN filter activations can provide a clearer understanding of the features that are best captured and how training data could be collected to improve performance. To generate a tub histogram, the following command could be used:

donkey tubhist --tub=./data/data_folder --output=output_file

The data_folder argument indicates the name of the data folder within the data directory that is used to plot its histogram and the output_file is the desired name of the output file. Lastly, to visualize the filter activations of the trained model which shows retained features, the following command line command is used:

donkey cnnactivations --tub=./data/data_folder --model=./model/model_name.h5

The data_folder and model_name arguments represent the same meaning as the tub history command. With the these tools, significant analysis could be performed on the trained model and training data.

4.2.III - Testing the Model:

Once the model is trained and evaluated, the model must be uploaded from the Desktop to the Jetson Nano under the /YoungWoodsCar/models/ directory where it can be tested on the golf cart. Just like importing the training data into the Desktop, MobaXterm provides another simply method to import the trained model from the desktop to the Jetson Nano. Once the trained model is loaded to the Jetson Nano, under the YoungWoodsCar directory, the following command could be entered to load the trained model into the golf cart's autonomous mode:

python3 manage.py drive --js --model=./models/model_name.h5

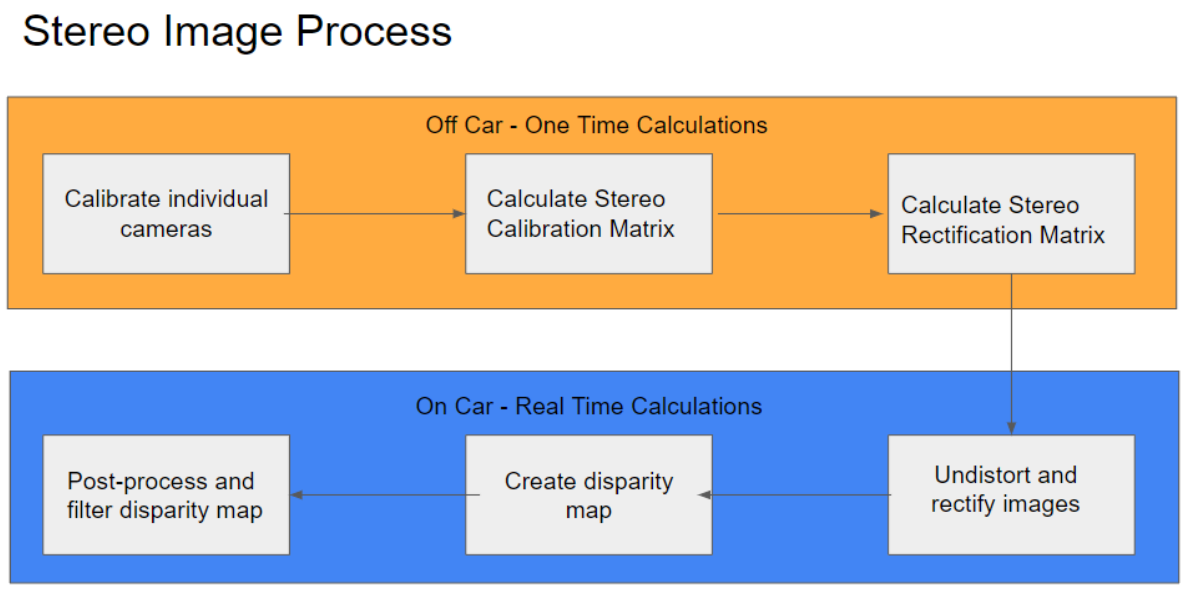
Once the DonkeyCar UI appears, set the mode to Full Auto where the vehicle runs autonomously with the trained model predicting the throttle value and steering angle given the real-time camera frames.

4.2 Computer Vision:

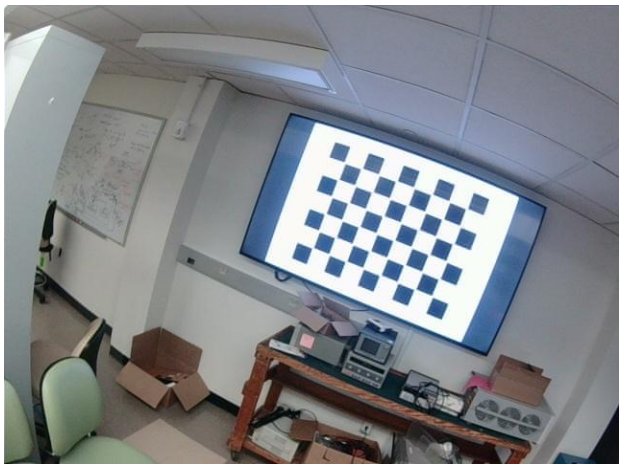
We decided to incorporate a stereo camera into our system in order to improve the overall safety of our system. Stereo cameras work in a similar way to the human eyes - through a process called triangulation, where the slightly-offset views of two cameras produce a difference in the pixel locations of each object. From this pixel difference, called the disparity, depth can be calculated through the following equation:

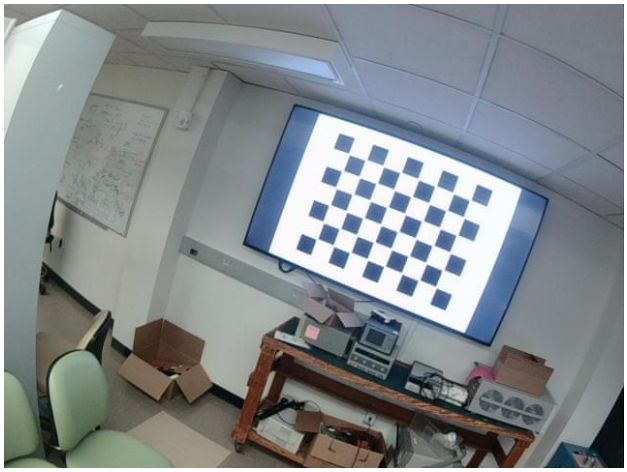
$$depth\ (mm) = \frac{baseline\ distance\ (mm) * focal\ length\ (px)}{disparity\ (px)}.$$

The process we used for implementing stereo vision is shown in the following figure:



The first and most important step of working with a stereo camera is camera calibration. Calibration allows us to obtain a highly accurate mapping of 3D world points to 2D pixel coordinates within the image. The calibration process consists of taking ~50 images of a calibration target (typically a checkerboard) at various angles, and at a relatively constant distance. The calibration target should be at a distance similar to the desired distance of the application. We chose a distance of around 10 - 15 feet. Make sure to take a wide variety of pictures from different angles and distances to ensure that the calibration is as accurate as possible.





Example pair of calibration images. The calibration target should be as flat as possible; therefore a checkerboard image was displayed on a tv screen.

After the images are taken, they can be uploaded to MATLAB's stereo camera calibration toolbox. The calibration process is relatively simple and is summarized here: Using the Stereo Camera Calibrator App - MATLAB & Simulink (mathworks.com)

After generating the camera parameters from Matlab, they can be imported into OpenCV or whichever environment you would like to generate the actual depth map. These parameters are called the camera intrinsic and extrinsic parameters, and from these it is possible to generate a depth map. The parameters we obtained from our calibration will be available in the GitHub. A simple guide for using a stereo camera with openCV can be found here: Making A Low-Cost Stereo Camera Using OpenCV | LearnOpenCV #

GPU Acceleration:

4.4: PCB Design

All PCBs were designed in EAGLE.

V. 3D Printing

Many of the components of our project were created using 3D printing. While this is great for rapid prototyping, there are some drawbacks, usually in terms of durability. This section will discuss some of the options you can change while printing to achieve different effects, as well as how we connect 3D printed components in high force environments.

5.1 Printing Options

The most significant decision when it comes to actually printing out an object is how tough you want it to be. The OEDK defaults will give you a print that is fairly resilient to normal wear and tear, but if the print will be under any stress then you will likely want to consider increasing the wall thickness and the infill density. The wall thickness refers to how thick the walls of the 3D print are. Walls exist in the first place because 3D prints are not solid; instead, they are filled in with some pattern to save on material. The infill density controls how much air is present inside the print.

There are two trade offs I encountered when increasing these values. The first is that the print will take significantly longer to print. In some cases, our parts were taking over 12 hours. This is still faster than ordering a part, but if you don't need the extra strength it's probably unnecessary. The other tradeoff is in print uniformity. I found that when I increased the wall thickness and infill to the maximum allowed amount, the prints started to show some small deformity, such as them not being completely level. This tended to happen in the vertical plane, not the horizontal, so something like a cylinder remained straight, but the top may not have been flat. However, this never actually caused any issues for us.

5.II Connecting Components

In some cases, even the strongest of 3D printed parts will fail. This happened to us when we tried to print an adapter from the motor shaft to the gears on the steering wheel. The motor shaft has a D-shape, so our adapter had the negative of the D-shape. However, the motor destroyed any print we tried. This forced us to instead order a metal adapter which we could then screw into the plastic. Our theory is that too much force was being applied to too small an area, causing the plastic to get crushed. The metal adapter can easily handle said force, and through the use of screws the force is transferred over a much larger area. The below image is an example of the adapter we used.

I found that for M4 size screw holes, a hole of diameter 4 mm tended to work well.

If trying to friction fit two components, be aware that the 3D printer is not perfectly accurate. Either give yourself some wiggle room, or be prepared to sand down the plastic. I would be careful about the second option, since you're wearing away at the outer walls of the print. Additionally, if you fit a part tightly it can be next to impossible to separate them, since the plastic is slightly rough and will want to stay put.

5.III Existing CAD Files

All the existing CAD files have been uploaded to the project GitHub repository found here. Feel free to tweak and modify these files to fit your needs, or just use them as inspiration for your own.

VI. TLDR + Others

6.1 Summary

We hope that this manual will help you avoid some of the headaches we encountered throughout the course of the year! If you still have any questions about any of the information in this manual, feel free to contact us.

6.2 Project Regrets

No meche :(

6.3 Further Ideas

Some ideas for a new team taking over the project include:

1. Improving the motor mounts and gears by converting to an all metal/wood design
2. Making a single PCB to house all of the electronic motor control components
3. Creating a new ML model using cutting-edge techniques such as transformers or reinforcement learning in order to improve the self parking and self driving capabilities
4. Increasing the number of stereo cameras or adding a LiDAR sensor, and implementing algorithms such as SLAM to improve the golf cart's spatial awareness
5. Adding GPS Modules to the FOB and golf cart so that the golf cart can be remotely summoned back to the user's location