Electric GoKart and Outreach Final Report - Spring 2022

Electrical Engineering:
Nikola Durand, Vani Kapoor, David Neitenbach, Rico Barela

Computer Engineering:
Andie Groeling, Ryan Guidice

Vertically Integrated Program Students
Matt Gilmore(EE)

Supervising Professor
Olivera Notaros

Engineer in Residence
Doug Bartlett

Approved By: Olivera Notaros **Approval Date:** May 6th

Abstract

Electric GoKart is a fully electric small vehicle, built from the ground up, with a focus on high modularity and data collection to promote semi-autonomous actions. This year's iteration includes a new operating system, upgraded controls, and multiple new sensors. The GoKart was developed by Electrical Engineering students focusing on controls and power systems, with Computer Engineering students focusing on operating systems, data management, and control programming. During this year's project development, it was important that we kept close records of our progress and decisions so we can use this vehicle to teach a younger audience about ECE*, STEM*, and how these fields intertwine.

Throughout the project, the biggest constraint that the team has been working around is having no mechanical engineering students on the team. As stated above, the work around this was to set our focus on additions we can make to the Kart rather than larger mechanical improvements. On the electrical engineering side, the improvements made mainly revolve around our power management systems. The task at hand was to alter last year's team's design of the main system of the Go-Kart and through the process of schematic documentation and testing, the team has been able to diagnose and fix the issues that we began with at the start of the semester. This has given the Go-Kart a safer and more efficient power system that can now be improved upon by future teams. Through research and calculations, the EE* team has made the decision to add additional power systems to the front of the Go-Kart to show off PV* charging and to create a more stable system for us to be able to power the bulk of our control systems. These new additions will allow us to have a more stable and reliable system that is easy to debug in future years, and provide a great talking point for the team and project to teach a younger audience about how the power systems would work. Further work and research has led the team to design an object detection system through machine learning techniques as well as implementing a fully functional touch screen user interface UI* using a Raspberry Pi 4B. More control research has been conducted as well to learn how to use a CAN bus for from the ESCs*, allowing us to poll real-time kart data.

The team also was tasked to have an organized plan and budget so that each of the above tasks could be accomplished. The use of team meetings and EIR* mentor check-ins proved the most helpful to keep the team on track and ensure that progress had been made in every aspect of the design. Budget management was taken on by one member in communication with the whole team to ensure that each purchase was not only feasible, but also was either completed smoothly or a solution was found to ensure the team's materials were purchased. Difficulties arose as the supply chain shortage greatly affected purchasing and ordering in many ways. These shortages put the team off schedule by generating a lot of uncertainty about when parts would arrive. This had to be overcome by working and improving on the items that we had and learning about what more could be done with what was already in the lab.

New safety features were implemented in the back portion of the Go-Kart, such as chain guards and ensuring our ground connections will not cross any high potential. We successfully completed our machine learning module and worked to fully integrate the objection detection technology into the new user interface system of the Go-Kart. After debugging and careful consideration, the battery system has been rewired to ensure the safety of our electronics and functionality of the Go-Kart. We also completed a new power system design to isolate the motor control and dashboard components from one another.

*Appendix A

Table of Contents

Abstract	**2**
Introduction	**4**
Review of Previous Work	**5**
User Interface (EKartUI) Design	**6**
Autonomous Pedestrian Detection (APD)	**7**
EKartUI* and APD* Deployment and Integration	**9**
Component Communication and Data	**10**
ESC and EKartUI Communication	**11**
Front Solar Panel Battery Charger	**13**
Power Redesign	**14**
Standards	**16**
Conclusion	**16**
Future Work	**17**
References	**19**
Appendix A: Abbreviations	**20**
Appendix B: Budget	**22**
Appendix C: Timeline	**24**
Acknowledgements	**27**

Introduction

This project acts as a second year continuation project from last year’s team including students from both Electrical, Computer, and Mechanical engineering. This project serves as a senior design project for this year’s six Electrical and Computer engineering students, and more importantly as a project to be used for student outreach to teach students who are interested in STEM*. Starting with just a simple electric vehicle from last year’s team, this year’s team is adding on modern-day features that show off the skills and knowledge of electrical engineering topics. When used outside the scope of senior design, this team is able to demonstrate the wide variety of topics that encompass engineering.

The team has found in previous outreach events that students typically just think of circuits when ECE* is brought up. This kart intends to break that common misconception by serving as a combination of the wide array of ECE* fields there are available. Between power systems, solar charging, user interface design, control systems, machine learning, and electronic communication, the kart covers a large amount of ECE* topics that are intended to appeal towards a younger audience. By showing young students the range of opportunities possible in ECE* disciplines, the team intends to motivate them to further pursue STEM* concepts and engineering as a whole. Throughout the few outreach events the team has attended with the kart, this goal is certainly on its way to being accomplished.

With roughly $6000 remaining from last year's team, as well as the additional $200 per senior design student given to the team, we were able to start out in a comfortable place to purchase and buy any additions to the Go-kart that we needed. In addition to these funds, the CSU* ECE* Outreach team has granted us an extra $3000 in budget to add to our existing budget from last year's team. Our full planned out budget can be found in Appendices B and C. At the end of the year, the team received an additional $11,000 from Ball Aerospace for next year's team’s use.

The design restraints this year revolved around the fact that we were unable to collaborate with the mechanical engineering department. This constraint became known early enough in the process that it allowed us to shift gears quickly, and we were able to move on and begin with troubleshooting last year’s design. At the same time as the troubleshooting, the controls team were able to begin their work on the machine learning and UI* aspects of the Go-Kart. Throughout the year, there were power complications that a sub-team took on. The final iteration of the project no longer has these issues and comes fully packed with solar charging, a powerful buck boost system that can power multiple stand-alone interchangeable devices, and serve as a backup unit to separate the front and back of the vehicle. The central brain of the vehicle is powered by an array of batteries that use regenerative braking to ensure a long travel distance and life. The speed controllers are on the verge of operating in unison allowing for a massive increase in vehicle torque and power overall.

Electronically, the biggest constraint for the project is to ensure that the project is safe enough to demonstrate to young kids, and ensuring that everything is sound enough for ease of travel with the Outreach team. This is a high power system that can harm not only the electronics on board, but also those around the vehicle, should something happen. Mechanically, the vehicle must be able to pass through doorways and other size limiting areas. The team needed to make sure that the focus of the project stayed on teaching people the countless fields within engineering. Because this is a university sponsored project, the continuation of the project also became a constraint. There was a need to ensure a transfer of knowledge to teams that came after this year. These constraints have been carefully considered and the team is always looking for new ways to ensure the safety of ourselves and our audience.

The following report is an in-depth look and analysis of our progress this semester. The first section, titled Review of Previous Work, is a summary of the work completed before the project was handed off to us, but the following sections will be all that we have completed throughout the semester. The first section highlights the difficulties we encountered as well as the initial improvements we had to make to the Go-Kart. This was a large portion of the work that was done by our electrical engineering team who also take care of power improvements and management. After this section, this document will go into more detail of all of our technical aspects we are adding to the Go-Kart. The first subsection will highlight the work done for our UI*. This section will speak to the use of Qt designer and how the design of the UI* has been completed through this program and the tasks completed and faced with this. The item will be looking at our machine learning model that has been implemented. This will cover the use of the Raspberry Pi 4B and the programs and systems used to accomplish object detection. After this we will go over the new serial to CAN communication and will cover the reasoning behind the change and how that was implemented. The following sections will introduce the new front solar panel charging circuit. This will go over both the design and implementation. Next, this document will detail all of the power management and any design that is used to power individual systems on the Go-Kart. Included in each of these sections will be the difficulties encountered during the design and implementation processes of each subsection of the vehicle.

*Appendix A

Review of Previous Work

Last year's team provided us with a semi-working kart that served as a great start to our project. Two mechanical engineering students on their team designed and fabricated the chassis in-house, which then continued to serve as the chassis for our team. They also left behind a kart power system consisting of six LiPo* battery cells, two ESCs*, and two motors. The front dashboard was also in a semi-usable state, with several LCD displays and switches to display data and control components on the kart.

Despite their work, we felt it necessary to change almost everything about the power system and dashboard. While the kart was functioning for last year's E-Days* event, components wired in after the event caused this year's team significant issues at the start of the year. A lack of clear wiring documentation also made the power system an unsafe piece of the kart. The ESCs* also ended up needing to be replaced due to electrical issues towards the beginning of the year. The dashboard also faltered due to its low noise tolerance, and failed to deliver consistent information to the kart's driver.

While this year's team did continue to use the chassis, this was more due to a lack of mechanical engineering students with design and fabrication expertise. There were several questionable design decisions made on the chassis that provided unique engineering challenges to this year's team. Those challenges are discussed further later in this report. While last year's team gave our team a good starting point, we ended up needing to replace many aspects of the kart.

*Appendix A

User Interface (EKartUI) Design

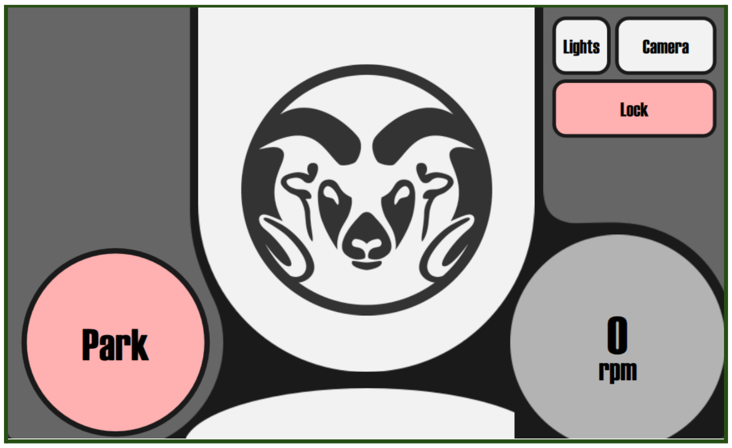

Figure 1: Mock-up of the User Interface

The user interface, EKartUI*, was first designed in Inkscape to create the display that will be put onto the Raspberry Pi touchscreen display. This design was then programmed using Python3, PySide6, and Qt 6.2.3, which gives the design more functionality and usability as the sole user interface controlling many of the components on the Go-Kart.

The purpose of this dashboard is to display many features that the team decided to implement and take measurements from. This screen includes information such as speedometer, tachometer, and battery charge levels. Since this will be displayed on a 7" touchscreen mounted into the front dashboard panel, the next biggest feature to be included is that students will be able to interact with the screen and then begin viewing the APD* machine vision output as talked about in the next section. Other controls exist for controlling direction, putting the kart in park, locking the interface, and changing interface settings. While these options that rely on the Raspberry Pi sending output to other systems do not function yet, the implementation for these controls would be as simple as controlling a GPIO* pin with a button press. The interface also uses a modular design that can easily be expanded upon by future teams. This allows the new UI* to serve as a platform for future development and improvements, rather than something that needs to be scrapped by a new team due to a lack of functionality.

A key component of this dashboard that we would like to implement next year is also to display when there will be any warning or faults in the control system. These faults will range from faults that the devices we have choses already give warning too as well as faults not yet fully chosen such as "battery critical" and motor and ESC* faults.

The dashboard also features some fault-detection behavior. If either the APD* system or the CAN parsing system discussed later on fail to start, the UI* will show elements to convey the failure to the user. In the case of the APD* system failing to start, the camera mode that shows APD's* view simply shows a black screen. This is a satisfactory error-handling method because APD* is not a safety-critical feature on the kart. If the CAN* parsing script that displays RPM* and MPH* fails to start, the tachometer will display "ERROR" to let the user know that something is wrong. This method of error communication is more applicable in this scenario because knowing the speed of the kart is somewhat safety-critical. As more features get implemented into the kart, this is an area that will continually need to be improved.

*Appendix A

Autonomous Pedestrian Detection (APD)

The Autonomous Pedestrian Detection (APD*) system is a machine learning model monitoring a webcam feed to perform object detection on a variety of vehicular classes, most notably pedestrians. This system was developed in partnership with Andrew Helmreich of the ML* on the Edge senior design team. The main goal of this feature is to provide a real-time demo of bounded-box object detection on the kart. While this data could theoretically be used to implement autonomous braking functionality (and we consistently evaluated the system's performance in this capacity), we decided this was too complex of a scenario to put our model into. Based on the overall goal of the kart, to demonstrate ECE* applications to students, we decided it was more important to have a live demo of this feature, rather than have it actually make decisions for the driver.

The base model of APD* is YOLOv5*, a machine learning model developed by Joseph Redmon and Ali Farhadi for real-time object detection. This model was selected due to its impressive performance in the real-time object detection area, where it consistently matches the accuracy of competing models while doing so in much less time. This was important for our application, as we theoretically want to keep the kart's stopping distance as low as possible in a braking scenario, and decreased model latency results in a shorter stopping distance. As for the data used to train the YOLOv5* model, we went with the KITTI* dataset. Developed by the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, the KITTI* dataset is heavily geared towards autonomous driving. With built-in classes for pedestrians, cars, cyclists, and other vehicle and traffic-oriented objects, this was the perfect choice as we didn't have to edit the dataset's built-in classes. We also added some of our own data using the Roboflow tool. By importing a few hundred of our own pictures of people, we used the tool to draw bounding boxes around them and label them as pedestrians, which added some extra data to the main class we were targeting with the APD* system.

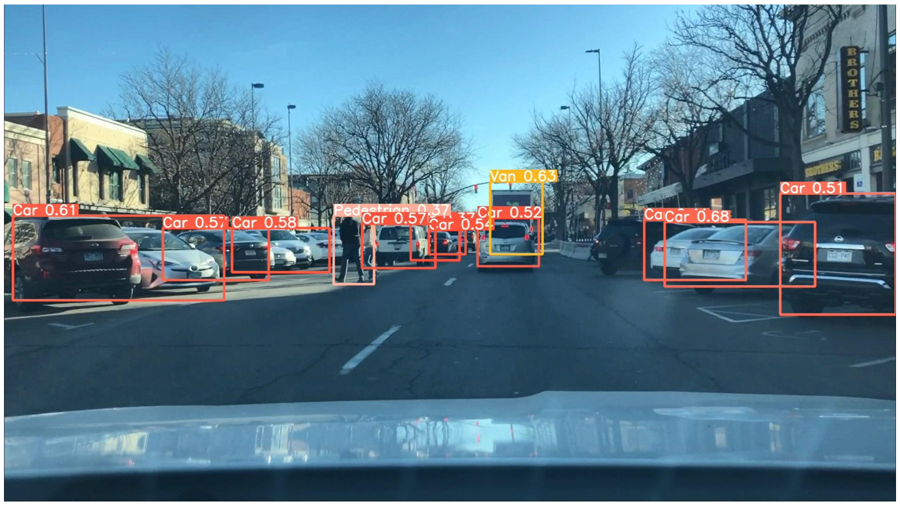

Figure 2: Bounded-Box Object Detection in Downtown Fort Collins

To deploy this model, we are using a Raspberry Pi 4B with a Coral USB* accelerator and a Logitech C920X USB* webcam. The Raspberry Pi 4B was selected due to our familiarity with the Raspberry Pi line of single-board computers and the versatility they allow in these types of applications due to their small size, low power consumption, and relatively high performance.

The CPU* of the Raspberry Pi 4B isn't quite powerful enough for our needs in machine learning inference performance though, so we purchased a Coral USB* accelerator to help in this category. By just plugging this accelerator in over USB*, making slight modifications to our deployment code, and re-compiling the model for Coral operations, we went from inference times per frame of ~3s to ~0.03s. Before we had the Coral, all 283 operations in our model's inference process had to be run on the Raspberry Pi's CPU*, but after re-compiling for the Coral, just 21 now run on the CPU*, with the remaining 262 operations being executed on the Coral accelerator. In addition to the Coral's custom Edge TPU* accelerating inference by being specifically designed for that application, the re-compilation also employs quantization on our model. Quantization approximates our original model from using 32-bit floating point numbers down to 8-bit integers, which only slightly hurts accuracy while drastically improving inference times. We also purchased a Logitech C920X USB* webcam to use as video input to our model due to its high resolution and good performance in autofocus and lighting correction.

Input Resolution	Inference Time (s)	FPS*	CPU Ops*	TPU Ops*	mAP* (%)
256x256	0.021	47.62	0	285	45.7
320x320	0.032	31.25	21	262	53.7
384x384	0.048	20.83	21	263	54.2
448x448	0.092	10.86	21	263	56.4
512x512	0.163	6.13	21	263	58.1
640x640	0.275	3.64	40	246	61.3

Table 1: APD Performance at Various Resolutions

As shown in Table 1, we evaluated several model resolutions for their performance in both inference time and mAP*. We settled on the 320x320 option as the best balance between FPS* and mAP*, as it maintains a respectable mAP score while maxing out the highest possible FPS* we can achieve with our deployment (since our webcam is capped at 30 FPS*). The 30 FPS* (due to the webcam limit) of this model means that if the kart was traveling at 20 MPH*, our system is able to make detections in each frame every 11.73 inches, which we believe is certainly fast enough for our demo. When testing this model in the lab and on footage we recorded from a car, the bounded boxes are fairly accurate, as shown in Figure 2. The image shows some inaccuracies in certain boxes, as well as missing a few cars, but it does notably well on the pedestrians and detecting the van in the background.

*Appendix A

EKartUI* and APD* Deployment and Integration

One step that took significant work during the second semester was the deployment of EKartUI* and APD*, and then the integration between the two systems. We were able to demonstrate APD* working by itself on our Raspberry Pi 4B with a Coral USB* accelerator and webcam, but this simply displayed the output in a window created by the OpenCV package. We were also able to run EKartUI*, but only on our Windows development machine, not Raspberry Pi 4B, which runs a distribution of Linux.

Deployment of the EKartUI* system on the Raspberry Pi came first, and took up a significant portion of our time. We made the mistake of not thoroughly checking compatibility and ease of deployment before writing EKartUI* in PyQt 6.2.3, and paid dearly for such a mistake. After several weeks of trying to work around the problem, we ended up having to build Qt from source on the Pi then install the Python aspects on top of that. We were able to follow an experimental guide for installing Qt from source on the Pi, but this process was not straightforward and involved a lot of dependency issues and waiting for compilations to finish. The majority of the install scripts provided by developers simply do not work due to incompatibilities with the ARM* CPU* embedded in our Raspberry Pi 4B, which then required extra debugging work to get everything into the right place at the right time. Overall, the process follows a flow of: install all of the Qt dependencies, download and build Qt 6.2.3 from source, install all of the PySide6 dependencies, download and build PySide6 from source, then install the Qt submodules necessary for our specific UI*. This process was extremely frustrating and time-consuming, and could have been prevented with more thorough research geared towards our known deployment environment. In the end, we were able to complete a writeup of the entire process to prevent future senior design teams from having to deal with the same problems.

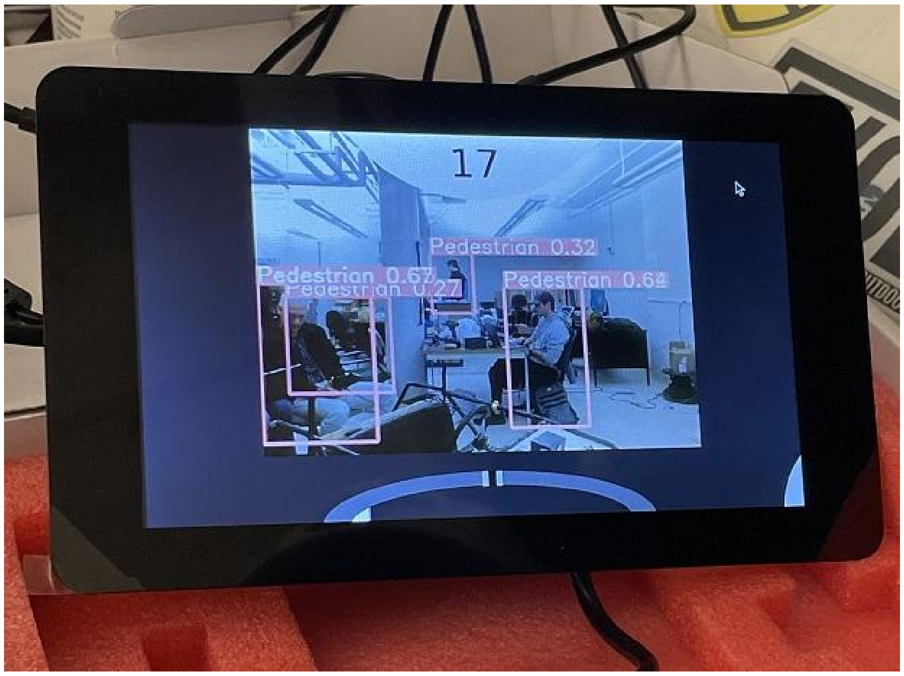

Figure 3: APD running inside of EKartUI

Once EKartUI* was able to run on our Raspberry Pi, we turned our attention to getting the APD* system integrated into it. After researching and trying several options, we ended up settling on using shared memory to pipe new APD* output frames from the APD* process into EKartUI*. Our solution generally works by having APD* do the computation to render a new frame, then write it to an established area of shared memory. The EKartUI* process then attempts to read from this shared memory to retrieve a new frame to display. If the read is successful, it shows it inside of the "Camera" mode of EKartUI*. If the read was not successful, say due to reading during the middle of a frame being copied or some other corruption, the UI* will simply throw the frame out and try again. This adds some slight latency, but provides higher memory safety to prevent a corrupt or incomplete frame from crashing EKartUI*. While we could not measure this added latency quantitatively, we observed the two versions side by side, and estimated it to be around 0.3-0.5 seconds of added delay, a value we were fine with due to the APD system not making any time-critical decisions on the kart.

One final added touch the team completed was having APD* and EKartUI* run automatically when the Raspberry Pi first boots. This is managed through a Linux service, which runs a script developed by the team. This script first starts our CAN* parsing software (discussed later) and APD*, waits until they start up correctly, then starts EKartUI*. This ensures EKartUI* runs correctly since its interprocess dependencies are already running correctly. Error detection for this process also exists. If EKartUI* cannot open the shared memory used between itself and APD* and the CAN* parsing script, it will display errors according to the process that failed. If APD* fails to start for whatever reason, EKartUI* will simply show a black, blank screen when switching to the camera mode. If the CAN* parsing script fails to start correctly, an "ERROR" message will appear inside the tachometer UI* element. Since the tachometer reading is far more important to the kart's operation than the APD* demonstration, we went with an active error message to inform the user immediately that something is wrong.

Overall, the final product demonstration with EKartUI* running correctly and having the APD* demo just a touch of a button away was a massive achievement for the team. We believe this will be a key feature for demonstrating the power of machine learning to future engineering students, and building a system where this great demo is within one button's press was extremely gratifying.

*Appendix A

Component Communication and Data

Most component communication in our system falls into the category of time-sensitive or time-insensitive. Time-sensitive parts could be operated using any of the protocols suitable for the component. For our system, we started using serial data and SPI* protocols for synchronous interfaces, time sensitive or not. With a single master, we would be able to have numerous slaves and a respectable data speed up to 10Mbps*. This was ideal because we could link sensors, controllers, and additional modules to one location for communication. I2C* was also being tested in conjunction to determine if it was a more economical solution. A typical I2C* bus would have a slower data transmission rate, but was still above minimum parameters to test. A great benefit of I2C* is its 7 bit or 10 bit address system, this would allow us to target specific devices on the bus. Compared to previously applied systems, I2C* was looking great. The need for only two wires and a device limit we wouldn't come close to made the protocol look like a good option. However, the required speeds to run the brakes, throttle, pedestrian detection, motor response, and other time sensitive functions were not there. On the test bench, devices were working at acceptable rates and data could be collected, parsed, and displayed. Upon installation into the full-scale system, noise became a major factor and rendered a considerable portion of the system inoperable. During this process, the team was already looking into other routes of transmission including UART* and CAN*. Due to the rapid increase in noise from the back of the kart to the front, we settled on trying an approach to CAN* transmission. This would allow for fast, time sensitive data to transfer very reliably and completely eliminate the risk of noise. There are already CAN* protocols in the motor controllers and we already have TEENSYv.4 microcontrollers that allow us to adjust component communication from serial to CAN* for use in this bus. In the case we establish communication with new components or systems and something is not working correctly, CAN* allows us to see the IDs of each message coming from the bus and allows us to change the priority of IDs so we can see all data from them uninterrupted. There are multiple ways of parsing this data and having a central error diagnosis and configuration system makes this process a lot more manageable.

For non time-sensitive electronics, the communication is mainly left to the designer. Currently, all of the non time-sensitive systems are stand-alone and do not communicate with the bulk of the vehicle. These systems include the radio, LED*s, and distance/motion sensors. Even though this means there is more work when trying to pull data from the system, it simplifies design and implementation. For the current state of the vehicle, this works well for us. Subsystems in progress follow the general rule that the communication is up to the designer. If there are components that do not have native CAN* communication already, we have the materials to transition them to CAN*.

There are a few things to be cautious of when integrating new systems onto the vehicle that need to be set up to communicate. Most importantly are wiring diagrams and power considerations. When wiring, we learned the hard way to be extremely cautious not to cross wires. There have been incidents of sending RxTx signals through power cables and incapacitating electronics on either end. Power is also something to consider, and there are two power banks on the vehicle; one in front and one in back. The lead-acid battery in the front is used to power mostly time-insensitive electronics, as more and more are attached the team will need to continually evaluate if its performance is up to par and capable of its load.

*Appendix A

ESC and EKartUI Communication

Once EKartUI* and APD* were both running correctly and integrated with one another, the computer engineering side of the team turned their focus towards making the EKartUI* elements update live based on data from the kart's electronics speed controllers (ESCs*). The main target UI element was the tachometer, because live RPM* data could be read from the main ESC* and used to derive the kart's speed. As discussed previously, following the issues last year's team had with noise interference with I2C*-based data signal propagation on the kart, we decided to use the CAN* protocol due to its robustness, speed, and integration with our ESCs*.

The main issue with using CAN* communication with EKartUI* comes down to the Raspberry Pi host EKartUI* is run on. Raspberry Pis do not natively support the CAN* bus, requiring the use of add-on boards to facilitate the physical side of the CAN* specification. To allow our Raspberry Pi to communicate on the CAN* network, we purchased several of

MikroElektronika's CAN* SPI* Click boards. These boards have a SN65HVD230 CAN* transceiver and MCP2515 CAN* controller that are able to talk on the CAN* bus, while also supporting a SPI* interface for devices like the Raspberry Pi. This setup allows us to establish a connection with the CAN* SPI* Click board via a SPI* connection on the Raspberry Pi, then initialize this board as a network interface that can communicate using CAN* data frames. A software package called candump was used to facilitate CAN* data frame viewing and construction.

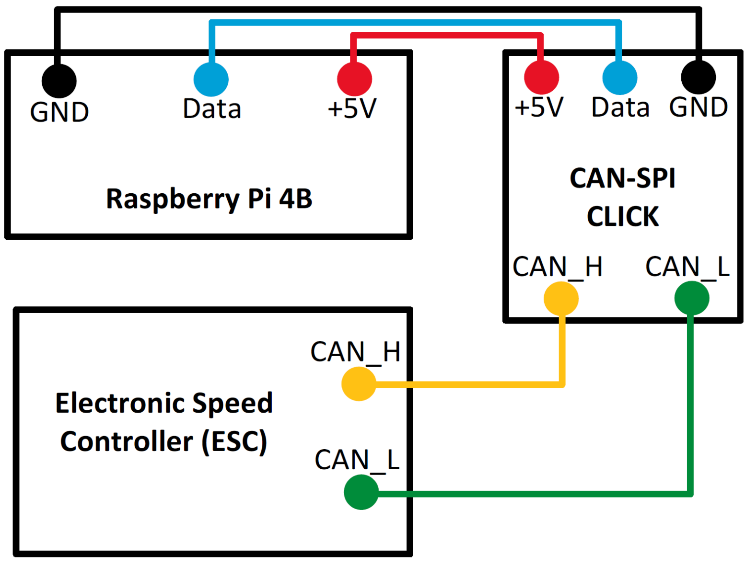

Figure 4: Go-Kart's CAN Bus Wiring Diagram

The team started off by using two Raspberry Pis, each with a CAN* SPI* Click board, to demonstrate the sharing of CAN* dataframes back and forth between the two. This allowed us to iteratively debug the entire communication setup. Once it was shown that we could pass CAN* dataframes back and forth, we knew both the physical and software sides of the CAN* protocol were working on our Raspberry Pi, and we could proceed to testing it with actual ESC* data.

Once the main ESC* was configured to properly send out information via its CAN* bus, we hooked it up to the Raspberry Pi and started dumping all the data we could get. Since the ESC* documentation fails to mention which CAN* dataframes are which and what the data inside of them was referring to, we had to run through several tests to develop theories about where the RPM* data was. The team accomplished this by logging CAN* data during an acceleration of kart's motors, then importing that data into Excel and walking manually through the set of four CAN* dataframes, looking for data that matched our acceleration pattern. This was tedious but successful, and told us the CAN* ID and byte pattern of the data we were looking for.

From there, the team developed a parsing script in Python that would poll the incoming CAN* data from the ESCs* and parse out the bytes containing RPM*. The script then passes this data over to EKartUI* using shared memory, and the EKartUI* can display the data. One problem we ran into was that the ESCs* were outputting ERPM*, not the actual RPM* of the kart's motors. ERPM* is electrical RPM*, and must be converted to get the actual motor RPM*. The conversion is simple - just divide ERPM* by the motor's polar pair count (6 in our case) - but this did confuse us for quite awhile and served as a good learning opportunity.

Now that EKartUI* has the motor RPM*, the kart's speed in MPH* can be easily derived using the diameter of the wheels and the gear ratio between the kart's motors and the rear axle. These conversions are all done in software on the EKartUI* side, and are re-computed every time a new ERPM reading is brought in over the CAN* bus. Overall, this system allows EKartUI* to display motor RPM* and kart MPH* data very close to real-time, further pushing the kart's progress towards a real vehicle.

In order to communicate and calibrate the ESCs* we used an open source software developed by VESC® Project called VESC Tool. This software allows the user to make a significant amount of modifications to the operation of the ESCs* including speed, power, and communication. To calibrate the ESCs* you need to have an estimate of the size of the motor and if it is an inrunner or outrunner. We are using a 1950g inrunner motor, so the large inrunner is good for calibration. Choosing the wrong motor size is a quick way to tear the motor, or worse, the ESCs* apart. The next step in the calibration is the battery parameters, of which we have a large LiPo* set totaling 12 cells with an 18Ah capacity. The motor pulley direct drive gear ratio is 10 and the wheel pulley is 20. Our tire diameter is 300mm. Running detection with CAN* immediately provides working data for the EKartUI* to display.

From the previous year's team, we inherited a trio of battered ESCs* which were replaced with Flipsky FSECS 75200 72V 200A speed controllers. These provided the team with reliable data through testing about the draw of the motors and helped ensure that no more startup damage could be done to the system. Prior to the new ESCs*, there was a grounding issue unknown to us that would frequently break the ESCs* or other controllers connected to the system. With the new ESCs*, the built in spark protection and power unit rewire gave us redundancy of safety in the system.

Communication between the ESCs* works with CAN*, having a master ESC and a slave ESC. Because of the single axle, dual motor layout of the vehicle, we had to ensure that the motors would operate at the same speed to avoid fighting and damage to the system. The ESCs* measure the resistance it takes to complete one turn and the power it takes to rotate and sync up with each other for optimal performance. The master ESC* is configured to ADC* and UART* with a PID* speed control type. The PID* speedtype is what really allows the ESC* to operate optimally. With an ability to employ a control loop with constant feedback measurement, we can calculate the error value in the system between our setpoint and process variable. As described earlier, this means we can check the motors constantly to ensure a smooth movement, all through the ESC*.

*Appendix A

Front Solar Panel Battery Charger

The idea behind the solar panel charger first came from the decision to have a separate power system for the electronics being placed up front. For the voltage rails needed, the team felt that if we were to overuse the 5V supply voltage from the buck converter it would not supply enough power to our control system. This would then render the Go-Kart unusable in many aspects. To combat this issue, the design came to add a simple 12V lead acid battery to the front of the vehicle that is easily accessible, and something we are able to buck down to a steady 5V voltage supply. The choice to use a lead acid battery in the front of the vehicle verses using the same LiPo* batteries that are used in the back is because of the way these batteries charge and how they react to overcharging. LiPo* batteries are very sensitive to overcharging, as they have a threshold voltage that they cannot exceed and the same case for the voltage when discharged. This would cause the battery permanent damage and therefore we foresaw a far greater need for replacement with these batteries. Lead acid is much more tolerable, as they do not over charge to the point to where they would break and have no issue being recharged if the battery voltage drops too low.

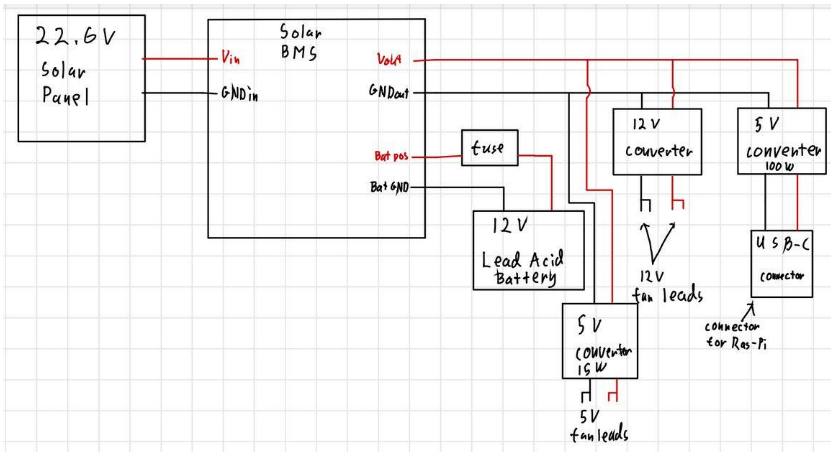

Figure 5: Solar Panel Charging Schematic

The above design includes a 22.6V 50W solar panel which was given to us by the Outreach team, a DC20838A BMS*, a 12V-13V lead acid battery, and a 5V regulator. The solar panel will be providing anywhere from a 18.75V to the max rated 22.6V input voltage to the BMS* which takes this voltage and supplies charging voltage back to the battery. As well as charging the battery, this device will also be outputting 12V which will serve as the input to our 5V regulator, which will then be powering our Raspberry Pi 4B controller hub and the 7" touchscreen display that goes with it.

The design has been fully implemented on the hood of the kart with slight variations to the original design. The only main difference from the original is the amount of outputs and type of outputs. The 5 volt converter is still powering the Raspberry Pi but some thermal considerations were taken into account for the Raspberry Pi, Coral USB* accelerator, and the ESCs*. Based on concerns about downstream current consumption on the Raspberry Pi, a 5V, 100W converter was used to power it. The 15 watt 5 volt converter was then used to power two fans in parallel to cool the Raspberry Pi and Coral USB* accelerator. A 12 volt converter was used to power two fans in parallel to cool the ESCs*. The only issue we ran into was on the morning of E-Days when there was an overvoltage issue that shorted the chip on the solar BMS*. To remedy this problem day-of, the BMS* was not utilized and wires were run straight from the battery to the converters. In the future this is easily remedied by replacing the BMS* circuit board.

*Appendix A

Power Redesign

The design of the power systems in the kart had to change quite a bit from what it was last year. After documenting the wiring schematic from last year we found design choices that were causing issues with starting the kart this year, and also may have been causing issues for last year's team.

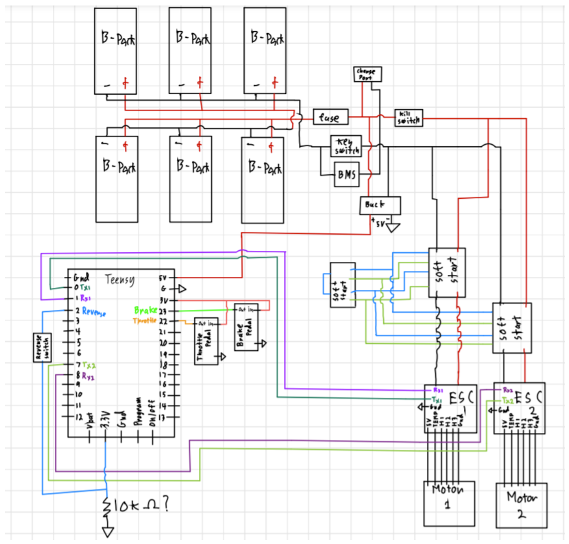

Figure 6: Wiring diagram of last semesters Go-Kart Electronics

From the wiring diagram and testing we found issues with grounding and connection points. The grounding issues were mostly due to the buck converter. With the buck converter, the ground for the TEENSYv.4 controller and the rest of the battery power system were shared. This caused power voltage spikes to be present on the control side of the power system which broke the chips of the TEENSYv.4 as well as the transistor on the throttle pedal. The buck converter could have also been the cause of signal noise issues from last year's team due to it being linked to the control ground. The duty cycle from the buck converter was linked to the signal ground of the TEENSYv.4, which we speculate contributed to unnecessary noise. The other grounding issues happened on the power side of the system with the power switch and the soft starts. The power switch being on the ground had a potential to have a current spike through the buck converter when switched off which could potentially break our controller chips. The problem with the soft starts took a lot of testing to find out what the problem was. It turned out the soft start used a MOSFET with a PWM* to create a slow rising voltage differential between the positive and negative wires. The problem was that the MOSFET was on the ground side making it so that the ground voltage for the ESCs* was higher than the control ground on the TEENSYv.4 until the soft starts hit steady state. This is a problem because the ESC* and TEENSYv.4 are supposed to share a ground and when the grounds have a voltage differential they cause a voltage larger than the ESC* and TEENSYv.4 chips could handle. Other than grounding difficulties, we had a problem with the ESCs* losing power at random times for no apparent reason. At first we thought that the connection problem was due to poor soldering but upon closer inspection it was the connector male plugs getting deformed with use that was causing them to be loose, with the capability to become open circuited while plugged in.

To fix the grounding issues we had to completely redesign the power systems of the Kart. As mentioned earlier in the report, we moved the control power system to a separate power source in the front of the kart to avoid all the issues with having the ground tied to the large voltage at the front of the kart. The redesign of the power side of the kart made it so things were less complicated and streamlined to avoid potential errors due to unneeded complexity.

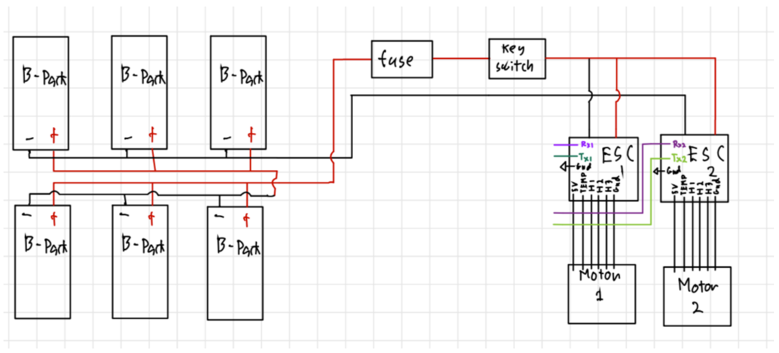

Figure 7: Redesigned Power system for the Motors

As seen in Figure 7, the battery is a lot less complicated than the previous system. The main changes made were the moving of the key switch and the removal of the charge port and the soft starts. The key switch was moved so that the ground is never broken, making for a more reliable circuit. The soft starts were removed to completely take out the ground voltage differential error, as well as implicating more advanced ESCs* that have their own built-in soft starters. Since then we have not had any grounding problems. The charger port and BMS* were removed as we could plug them into the batteries directly at any point to make the physical wiring less cluttered and easier to maneuver. The connection issue took a long time to find because the initial thought was that the wires themselves were faulty. As replacements were being made it showed that the male banana plugs in the connectors had the possibility of collapsing in on themselves and not putting pressure on the female banana plug. This was easily fixed by pushing the male plugs out with a screwdriver at all loose connection points.

*Appendix A

Standards

This team did not have to adhere to any standards. We operate at what is considered dangerous voltage; however, there are no standards that need to be followed unless the voltage is considered "high voltage".

Conclusion

- The Go-Kart is fully operational and all of the subsystems are implemented and working
- New UI fully integrated onto Raspberry Pi, including the APD system
    - Allows for increased user interactivity and an improved user experience
- Solar Charging of front system fully functional
    - Solar BMS chip shorted on E-Days, but is fully replaceable
- The power systems were completely redesigned and implemented
- CAN communication allows RPM data to be sent to the Raspberry Pi from the ESCs
- Added extra "flashy" features, such as a radio

Future Work

This semester the team was able to put together their version of an Electric Go-Kart. This project is now to be passed down to a new senior design team to do the same. With a vehicle designed for the intention of teaching younger students about ECE* topics, the team next year has a lot of flexibility in both adding on to all of the work highlighted in the above sections, or adding new items to the vehicle. We thoroughly recommended that this project not only be continued next year, but potentially even further due to the wide array of improvements and new features that could be added on to the kart. With this section, we would like to present some ideas we have about what future work could be taken on by new teams.

A key item that would be invaluable to future teams would be the fabrication of a new chassis. The current chassis is extremely limiting in several ways. First off, the solid axle in the back presents several engineering challenges, most notably when being driven by two separate motors. A transition to a new chassis with a split rear axle would allow for much more mechanical development to be made. This fabrication could also allow the team to reinstall mechanical brakes. The kart in its current form relies on regenerative braking to slow down. Our current configuration is such that when a driver is not accelerating the kart using the throttle pedal, then the regenerative braking kicks in, and the kart is braking. While this performs quite well, mechanical brakes would be a good improvement for the safety side of the kart. Whether the fabrication of a new chassis is something that can be done in-house by mechanical engineering students or needs to be sent outside of CSU* remains to be seen, but the limitations of the chassis remain the biggest constraint of kart improvements, even outside of the mechanical or electric areas.

As this team discovered on E-Days*, there are some thermal considerations that still need to be taken into account on the kart. The cooling system designed for the Raspberry Pi and Coral USB* Accelerator worked really well, but other portions of the electronic system did not fare as well. We found that the CAN* communication between the ESCs* and Raspberry Pi broke down when outside, and we believe thermal issues may be to blame. This may be due to the CAN* SPI* Click board overheating, or an increased number of ESC errors brought on due to heat. The ESCs* do monitor internal MOSFET temperatures, but we were unable to retrieve that data off of them this year. An analysis of those temperatures and errors on the CAN* bus during high temperature loads may be needed to determine the root cause of the issue. While the kart runs normally very well inside, we were surprised with the issues we experienced out in the sun.

One area we attempted work in but failed to complete before the end of the year was continuously-functioning synchronous communication between the two ESCs* mounted on the kart. As previously mentioned, the solid axle design of the kart means that the two rear motors must drive with exactly the same torque at exactly the same time. We were unable to get this synchronization to work quite correctly this year. The ESCs* were occasionally able to sync-up in terms of RPM*, but unfortunately not torque quite yet. This configuration also made the motors spin randomly by themselves every few minutes, a bug that needs to be ironed out for obvious safety reasons. Experiments late in the second semester showed that communication between the two ESCs* is possible, but due to a lack of documentation, we were unable to achieve it. Recent research by the team indicates that the ESCs* we currently have may not support a master to slave configuration, at which point it may be necessary to purchase ones that do have support for that arrangement. If a new chassis is fabricated that removes the solid axle constraint, it may be possible to continue as-is and have each ESC drive a motor independently.

Our recommendation is to acquire ESCs* that communicate with each other correctly though, as even in a split axle design, communication between the ESCs* would help reduce the motor wear that can come from extraneous forces.

While the APD* system is a great demonstration of machine learning technology, a natural path for the system would be to have it influence the kart's operation in some way, as opposed to just being a passive demo. This is a concept we explored significantly at the beginning of the year, but ended up needing to scrap due to time and mechanical engineering constraints. The addition of many more sensors and improved data collection could serve as extra input to APD*. APD's* model could be re-trained to recognize multiple inputs, or a second model could be trained to target other datasets. The key aspect that APD* is missing any sort of ranging functionality. Currently, it detects objects entirely through the imaging provided by a USB* webcam, but the system has no way of knowing how far away a detected object is. This would likely be a necessity if APD* was to make decisions based on its output. There are other concerns too, such as the presence of detectable objects in the camera's field of view that sit outside of the kart's path. Would the kart theoretically brake automatically for these as well? This possibility opens up an entire field of new challenges for future teams to take on.

On the topic of the dashboard area, further integration of the dashboard into a control system for the kart would be a great improvement. As is, EKartUI* features many elements that are not connected to anything physical on the kart yet, such as transmission and parking control, PIN*-based locking, and control of lights. These pieces are already built into the user interface, the backend infrastructure to carry out these operations from the Raspberry Pi's GPIO* pins simply needs to be built out as well. Another key element that does not have any backend support yet is monitoring the kart's battery levels. While the current battery indicator was designed initially to monitor the rear battery system during kart use, another battery indicator would likely be useful to monitor the front battery. Our team experienced the front battery dying quite quickly during E-Days*, so the ability to monitor when the entire dashboard system might shut off by itself due to low battery voltage would be a great improvement to make. EKartUI could even theoretically make decisions based on battery level, such as shutting it off preventatively before the battery gets too low to protect the Raspberry Pi's configuration from issues due to direct shut-off. EKartUI* also features indicators for regenerative braking, which we believe could be detected through improved CAN* data parsing from the ESCs*. As discussed in the thermal issue portion of this section, monitoring and displaying the temperature of the ESCs* would also be valuable.

*Appendix A

References

[1] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, Aug. 2013, doi: 10.1177/0278364913491297.

[2] Ultralytics, "YOLOv5," *GitHub*, Aug. 21, 2020. https://github.com/ultralytics/yolov5 (accessed Dec. 07, 2021).

[3] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv*, vol. 1804, no. 02767, Apr. 2018, doi: arXiv:1804.02767.

[4] K. Yamazaki, "pytorch-armv7l," *GitHub*, Nov. 29, 2021. https://github.com/Kashu7100/pytorch-armv7l (accessed Dec. 08, 2021).

[5] Hailong Li, Zhendong Wu and Jianwu Zhang, "Pedestrian detection based on deep learning model," 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2016, pp. 796-800, doi: 10.1109/CISP-BMEI.2016.7852818.

[6] Z. Liu, Z. Chen, Z. Li, and W. Hu, "An Efficient Pedestrian Detection Method Based on YOLOv2," Mathematical Problems in Engineering, vol. 2018, p. e3518959, Dec. 2018, doi: 10.1155/2018/3518959.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," arXiv.org, 2015. https://arxiv.org/abs/1506.02640

Appendix A: Abbreviations

ADC - Analog-to-Digital Conversion

APD - Autonomous pedestrian detection

ARM - Advanced RISC Machines, a processor architecture company

CAN - Controller Area Network

CPU - Central processing unit

CSU - Colorado State University

ECE - Electrical and Computer Engineering

EIR - Engineer in resident

EKartUI - our team’s new user interface for our Go-Kart

ERPM - Electrical Revolutions Per Minute

ESC - Electronic Speed Controller

E-Days - Engineering Days

FPS - Frames per second

GPIO - General purpose input/output

I2C - Inter-Integrated Circuit

KITTI - short for KITTI Vision Benchmark Suite, an autonomous driving dataset

LED - Light emitting diode

Mbps - Megabits per second

ML - Machine learning

MPH - Miles per hour

Ops - short for Operations

PID - Proportional-Integral-Derivative

PIN - personal identification number

PV - Photovoltaic

PWM - Pulse Width Modulation

RPM - Revolutions Per Minute

SPI - Serial Peripheral Interface

STEM - Science Technology Engineering Mathematics

TPU - Tensor processing unit

UART - Universal Asynchronous Receiver/Transmitter

UI - User Interface

USB - Universal serial bus

YOLOv5 - version 5 of the You Only Look Once machine learning model

mAP - Mean average precision

Appendix B: Budget

Description	Quantity	Unit Price	Projected	Remaining
Initial				$4,116.79
Raspberry Pi 4B 4gb from pishop.us	1	$112.00	$112.00	$4,004.79
ROCKPro64 4gb from pine64.com	1	$79.99	$79.99	$3,924.80
ROCKPro64 20mm Mid Profile Heatsink from pine64.com	1	$3.29	$3.29	$3,921.51
ROCKPro64 12V 3A US Power Supply from pine64.com	1	$8.99	$8.99	$3,912.52
2-pack 32gb class 10 microSD cards from amazon.com	1	$15.99	$15.99	$3,896.53
RPi4B Aluminum Heatsink Pack from pishop.us	1	$1.95	$1.95	$3,894.58
3ft Micro-HDMI to HDMI cable from pishop.us	1	$3.95	$3.95	$3,890.63
HDMI cable from pishop.us	1	$2.45	$2.45	$3,888.18
Raspberry Pi 7” touchscreen display from pishop.us	1	$64.95	$64.95	$3,823.23
Raspberry Pi 15W power supply from amazon.com	1	$9.99	$9.99	$3,813.24
Teensy 4.0 amazon.com	3	$20.00	$60.00	$3,753.24
Microcontroller for solar panel ti.com	2	$39.99	$79.98	$3,673.26
Buck boost for solar panel ti.com	1	$99.99	$99.99	$3,573.27
ESC Chips	5	$23.99	$119.95	$3,453.32
8-40V to 12V 6A Regulator amazon.com	1	$22.99	$22.99	$3,430.33
6.3-22V to 5V Regulator amazon.com	1	$12.59	$12.59	$3,417.74
5 Pack Large Wire Connectors amazon.com	3	$12.23	$36.69	$3,381.05
ESC Debugger digikey.com	2	$8.70	$17.40	$3,363.65
6 Gauge Wire Black/Red 15 ft. amazon.com	1	$46.21	$46.21	$3,317.44
Coral USB Accelerator amazon.com	1	$171.97	$171.97	$3,145.47
Logitech C920x Webcam amazon.com	1	$64.52	$64.52	$3,080.95

Battery Charger https://www.digikey.com/en/products/detail/DC2038A-E/DC2038A-E-ND/9996134?itemSeq=381827675	1	$99.00	$99.00	$2,981.95
Anti Spark Switch	3	$76.00	$228.00	$2,753.95
Flipsky FSESC	3	$319.00	$957.00	$1,796.95
https://www.mikroe.com/can-spi-33v-click	5	$21.99	$109.95	$1,687.00
Conduit clamps 10 Pack	2	$14.36	$28.72	$1,658.28
Battery Terminal Covers	4	$2.47	$9.88	$1,648.40
USB-C Cable	2	$7.67	$15.34	$1,633.06
AS150 Connectors	3	$18.99	$56.97	$1,576.09
EC5 Connectors	2	$12.99	$25.98	$1,550.11
12V 5A power supply	1	$16.01	$16.01	$1,534.10
4X 40MM Heatsinks and USB adapter	1	$24.71	$24.71	$1,509.39
3X teensy 4.0	1	$63.68	$63.68	$1,445.71
1528-2711-ND Ultrasonic Sensors	5	$3.95	$19.75	$1,425.96
3X 10-60 to 5V Converter	3	$21.49	$64.47	$1,361.49
3X 12-5V Converter 100 W	3	$19.99	$59.97	$1,301.52
5 X 3.3V converter	5	$6.00	$30.00	$1,271.52
2 X Led Strips	2	$18.99	$37.98	$1,233.54
Phone Chargers for power supply	1	$21.99	$21.99	$1,211.55
1TB hard drive	1	$49.99	$49.99	$1,161.56
T-Shirts	1	$400.00	$400.00	$761.56

Additional funding sources:

ECE Outreach Funding	+$3,000
Ball Aerospace Grant	+$11,000

Total Starting Money: $4,116.79
Total Spent: $3,355.23
Total Remaining After Grants: $14,761.56

Appendix C: Timeline

Semester 1:

Date Range	Objectives
8/30 - 9/5	● Define responsibilities of team members
9/6 - 9/12	● Complete FMEA and risk analysis documents ● Order controls parts ● Take group photo ● Started buck boost control code
9/13 - 9/19	● Finalize project goals ● Finish project website ● Write project plan ● Start analyzing last year's codebase and microcontrollers
9/20 - 9/26	● Start writing controls software ● Decide on type of ranging sensors (LIDAR/RADAR/ultrasonic) ● Begin work on solar charging system ● Mid-semester notebook check
9/27 - 10/3	● Began GUI development ● Started work on bluetooth LEDs
10/4 - 10/10	● Began pedestrian detection model development ● Redocumented missing wiring diagrams from previous team
10/11 - 10/17	● Revised project plan ● Radio system designed and mostly functional
10/18 - 10/24	● Held project management review with Susan Hunter ● Got the kart working for the first time this semester
10/25 - 10/31	● Discovered cause of electrical issues ● Finished current and voltage drivers
11/1 - 11/7	● Got initial versions of automatic pedestrian detection (APD) system working in development environment ● Researched solar panel design concepts
11/8 - 11/14	● Got initial APD versions running the Raspberry Pi with the Coral USB Accelerator ● Completed initial solar panel design

11/15 - 11/21	● Redid most of the power wiring from the battery ● Drastically improved APD performance ● Started parsing CAN data from the ESCs with IDs mapped
11/22 - 11/28	**Fall Break**
11/29 - 12/5	● Finalized APD resolution selection based on benchmarks ● Ran APD over custom footage to test accuracy in tougher scenarios ● Achieved reliable CAN data from the ESCs
12/6 - 12/13	● Created 12V and 5V rails with solar charger for the front of the kart ● Tested all hardware for solar circuit ● Present mid-project progress ● Turn in mid-project report ● Second notebook check

Semester 2:

1/17 - 1/23	● Finished initial UI version
1/24 - 1/30	● Have initial sonar sensors working ● Started working on getting EKartUI and APD on the Pi
1/31 - 2/6	● No progress to report
2/7 - 2/13	● Got EKartUI and APD running on the Pi separately ● Completed an install guide for EKartUI and APD
2/14 - 2/20	● Completed front end power delivery ● Removed the old dashboard and replaced its equipment ● Created full system diagram of kart components ● Made control system communication map ● Demoed ability to run APD inside of Qt video window ● Held a kart demo at Engineering Exploration Day
2/21 - 2/27	● Added speedometer and tachometer to UI ● Reverse control present in UI

2/28 - 3/6	● Replaced kart seat back ● Fixed steering linkage ● Realigned steering column
3/7 - 3/13	● Got APD working inside of EKartUI ● Replaced accelerator pedal
3/14 - 3/20	● Finished power system design and documentation
3/21 - 3/27	● Demoed two Pis communicating over a CAN network via SPI to CAN boards ● Completed SPI to CAN board and Raspberry Pi documentation
3/28 - 4/3	● Got the kart running with someone inside (no UI yet)
4/4 - 4/10	● Retrieved data from the ESC CAN bus onto a Raspberry Pi via SPI to CAN board ● Isolated CAN data that shows motor RPM ● Start final documents and presentation
4/11 - 4/17	● Finished E-Days poster ● Got CAN data displaying in EKartUI ● Printed close to final versions of Raspberry Pi mount ● Finished ESC to Raspberry Pi wiring
4/18 - 4/24	● Updated e-kart manual ● Made small UI changes ● Converted from ERPM to RPM ● Finished Pi mount ● Printed Coral cooling system ● Mounted rear power system ● Mounted and finalized front power system wiring ● Added LEDs and radio

Acknowledgements

This project would not have been possible without the support from our vertically integrated student, Matt. We really appreciate all of the hard work that he has put in this semester and have made some great additions to the Go-Kart. We are very pleased to be passing down the project to him and wish him great success.

We would especially like to thank Doug Bartlett for his continuous guidance and support that he provides weekly to the team. Doug has really pushed the team far and taught us so many skills that have brought the project to a different level. His drive to continue to teach us and push us to our full engineering potential has been something that the team really values and the skills we have learned from Doug are some that are unforgettable.

Another thanks is due to ECE Outreach and Olivera Notaros. The funding from the Outreach team and the push to allow this project to continue another year has truly been amazing not only to the senior design projects, but the students who come to the Outreach events and learn from our team. We greatly appreciate the opportunities that we have been given with the continuation of this project.