

CEDARVILLE UNIVERSITY DELTA BEE DESIGN REPORT

Kaitlyn Wiseman,* Ren Murakami,† James Payne,‡ and Madelyn Torrans§

May 15, 2023

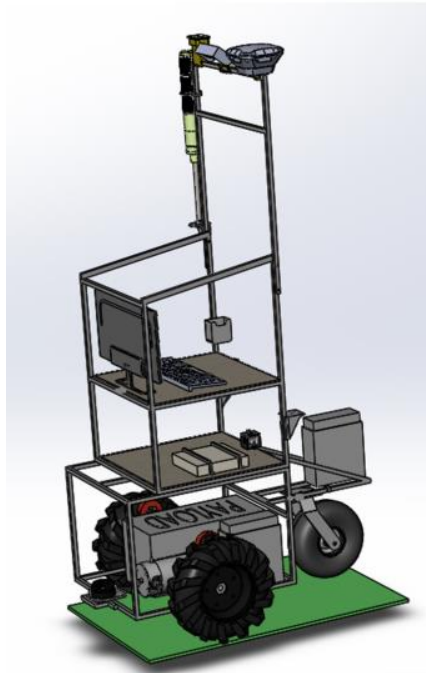
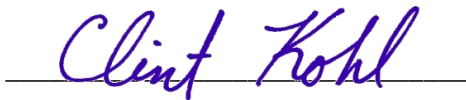


Figure 1. Delta Bee Vehicle Rendering.

I hereby certify that the design and development of the vehicle Delta Bee described in this report is significant and equivalent to what might be awarded credit in a senior design course.

Clint E. Kohl, Ph.D., Faculty Advisor



* Team Captain, Undergraduate, School of Engineering and Computer Science, kaitlynwiseman@cedarville.edu.

† Team Member, Undergraduate, School of Engineering and Computer Science, rmurakami@cedarville.edu.

‡ Team Member, Undergraduate, School of Engineering and Computer Science, jamespayne@cedarville.edu.

§ Team Member, Undergraduate, School of Engineering and Computer Science, madelyntorrans@cedarville.edu.

CONDUCT OF DESIGN PROCESS, TEAM IDENTIFICATION, AND TEAM ORGANIZATION

Introduction

The Cedarville University Delta Bee team is an Electrical Engineering/Computer Engineering senior design team. This year's robot, Delta Bee, is on its third iteration. Last year's iteration experienced intermittent motor control issues, unreliable GPS navigation, and could not avoid sawhorse or pothole obstacles. Because of this, the major modifications to Delta Bee this year include a new motor controller, addition of an IMU sensor, integration of IMU and wheel odometry into the GPS navigation algorithm, LiDAR hardware changes, and the addition of sawhorse and pothole detection and avoidance.

Organization

Our team functions as a scrum team, with the team captain functioning as both scrum master and product owner. The team captain was responsible for task assignment and timeline management, as well as supporting team members when needed. More information about the team can be found in Table 1.

Table 1. Team Information.

Position	Name	Year	Major
Captain	Kaitlyn Wiseman	Senior	Electrical Engineering
Member	Ren Murakami	Senior	Computer Engineering
	James Payne	Senior	Computer Engineering
	Madelyn Torrans	Freshman	Mechanical Engineering

Design Assumptions and Design Process

As development planning took place in August 2022 and execution began soon after, our main design assumption was that the 2023 rules would not significantly differ from the 2022 rules. This assumption was substantiated.

We used Scrum to organize our execution. We started out the year with a proposal and set of requirements, the action items from which became our issue backlog. We used Atlassian JIRA to track our backlog. Then, we executed development in a series of 4-week sprints for which we had milestone goals. At the end of each sprint, we evaluated where we were at in development and adjusted our process and priorities as needed.

EFFECTIVE INNOVATIONS IN VEHICLE DESIGN

There were some significant and effective innovations to this year's design of Delta Bee. The motor controller was replaced with a significant upgrade, a new GPS navigation algorithm was successfully implemented that integrated a high end IMU and wheel odometry. Our LiDAR processing software was overhauled to handle a wider variety of obstacles including much better sawhorse detection and avoidance, and our image processing software was updated for better pothole detection and avoidance.

Motor Control

During last year's competition, the robot's motors would periodically experience a momentary overcurrent condition, causing the motors to suddenly stop then restart. This resulted in unpredictable movement and would sometimes cause the robot's wheels to spin in place, throwing off the wheel odometry.

To overcome this problem, we integrated a new motor controller with a much higher current rating. Throughout our testing with this new controller, Delta Bee does not jerk like it did with the previous motor controller.

Previous Motor Controller	New Motor Controller
Smart Drive Duo Smart Dual Channel 30A 30A continuously, 80A peak per channel Motor Voltage from 7V to 35VDC	Sabertooth 2X60 60A continuous, 120A peak per channel Motor Voltage 6-30V nominal

IMU Addition

The previous method Delta Bee used to calculate its heading during GPS navigation was to draw a line between the previous two GPS locations received and consider that to be its current heading. This method proved problematic for two reasons. First, because Delta Bee moves at relatively slow speeds (its peak speed must not exceed 5 miles per hour), the GPS data received is noisy and inconsistent, especially on startup. This led to Delta Bee's heading estimates being equally inconsistent and noisy. Second, the heading estimate would always be delayed from the actual heading, as it could not sense heading changes from pivot turns in place. If Delta Bee turned, its heading estimate would not be updated to reflect that turn until Delta Bee traveled a relatively significant distance.

To improve the accuracy of our heading calculations during GPS navigation, we added a Bosch BNO055 IMU (Inertial Measurement Unit) to Delta Bee. The BNO055 was selected because it includes a high speed ARM Cortex-M0 based processor which performs sensor fusion tasks producing much more easily useable data. This IMU was carefully mounted in the center of the robot between the two drive wheels (on the axis of rotation) and on the top shelf of the chassis to simplify calculations and minimize any electromagnetic interference from the motors and other electronics.

The IMU is connected via I²C to our Teensy microcontroller and periodically polled for orientation data. This orientation data is fused into the ROS GPS navigation algorithm to maintain a much more accurate heading estimate.

GPS Algorithm

To integrate the new IMU data into our GPS navigation calculations, we implemented a new sensor fusion algorithm. A diagram of this algorithm can be found in Figure 2. An independent heading estimate is calculated for each data source (wheel odometry, IMU-supplied Euler angles, and GPS latitude/longitude data), and the results are summed using a weighted average. The weights for each data source change depending on how close Delta Bee is to the GPS waypoint; for instance, we trust the GPS more when we're closer to the waypoint than when we're further away. Implementation of this algorithm was successful and has made Delta Bee's GPS navigation much more reliable.

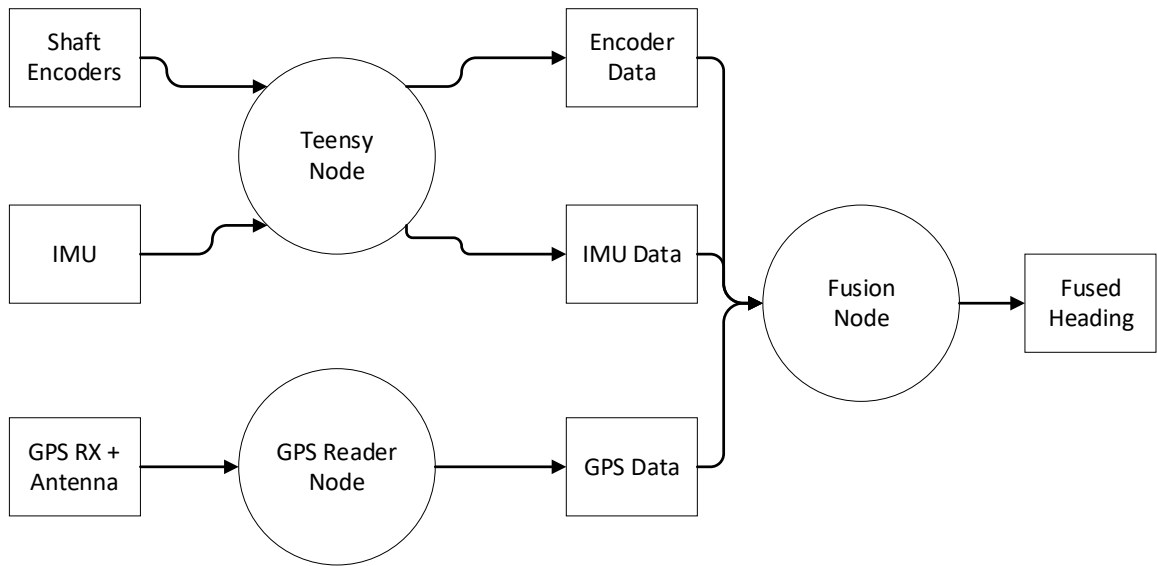


Figure 2. GPS Sensor Fusion ROS Node Diagram.

GPS Ground Plate

Although our GPS receiver and antenna are of high quality and do a good job, the GPS data drifts with time and suffers from a variety of error sources. One known source of error is due to multipath interference. To help reduce this error source we integrated a 15cm-radius metal ground plate onto our GPS antenna. This effectively eliminates signals bouncing off structures and entering the antenna from underneath. We were very pleased with the results of this simple addition and found the ground plate halved the standard deviation of our static GPS data error.

Sawhorse Detection and Avoidance

In the previous iterations of Delta Bee there was no implementation for sawhorse detection and avoidance – only barrel detection and avoidance. We decided to improve on this baseline and determined that LiDAR would be the best course of action for extending the detection capabilities to sawhorses as well as barrels. Therefore, we were able to reduce the amount of code by effectively half as well as solving both detection problems at the same time. Several significant innovations were implemented this year. First the LiDAR unit itself was physically rotated 90 degrees on its mounting plate. This change allowed for much more efficient data handling as the data of interest (the scan that is in front of the robot) was taken during the first 180 degrees of rotation. The back half of the scan was simply eliminated from all processing. Previously the first 90 degrees of rotation and the last 90 degrees were used. This change also made the LiDAR orientation more conventional (unit circle) and easier to implement a good solution.

Sawhorses and barrels then needed to look like the same type of obstacle to the LiDAR. Our team accomplished this in software by implementing a detection algorithm that takes in the LiDAR scan and passes over the entire scan in two passes. In the first pass, the loop goes over each point, which, if it is within the predefined distance window (2.9m - 0.25m), is saved, and the loop continues to the next point. Else, the point is assigned with an infinity value because that is either noise (<0.25m) or that it is too far away from the robot and there is no need to avoid it (>2.9m).

Once all the useful points have been collected, through the first pass, the corners of the object(s) are saved to two other variables outside the first loop. In the second pass, another loop passes over the scan again but this time creates onto the scan an artificial object by connecting the dots between

the corners of the object with a straight line that is at the same Cartesian distance as the closest point of the object(s) in the original scan. This effectively “paints” a non-solid obstacle like a saw-horse into a solid wall of points. This same algorithm effectively works for many obstacle types barrels, saw horses etc.

We also paint in the direction that we are line following because it doesn’t matter which objects there are or what orientation there are in, they can be considered as one object to navigate around. If we are left line following, we find the point/corner that is furthest to the right in the scan and project an object from that point to the edge of the scan (i.e., to the line). And vice versa if the robot is right line following.

Pothole Detection and Avoidance

Delta Bee could not previously avoid potholes. There had been attempts by previous years’ teams to implement a solution, but as our core strategy is to avoid mapping and persistence, it had been very challenging to do so. This year, we implemented a different strategy.

To detect the pothole, we first crop and then process the image we receive from our camera using a series of filtering techniques: grayscale, auto contrast, black and white, erosion, and dilation. The resulting image will be entirely black and white with most, if not all, of the image filtered except for any potholes. Then, the Hough Circle Transform is applied to the image and returns the center of any circles that match the expected radius. Examples of these image processing steps are shown in Figure 3 below.

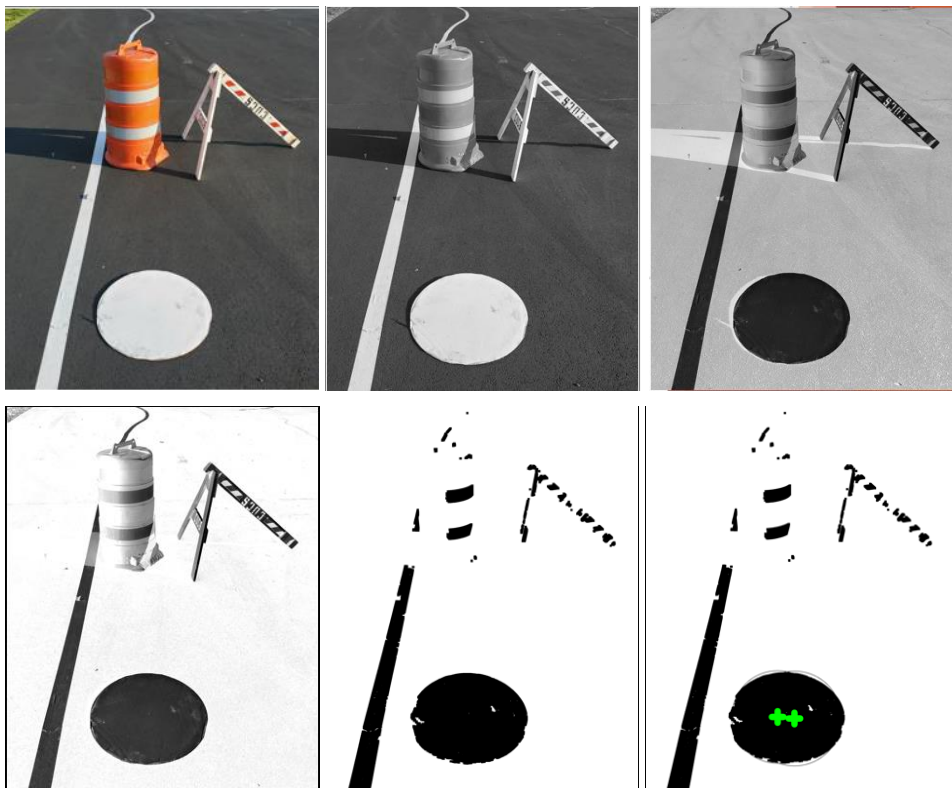


Figure 3. Pothole detection image processing steps.

The pothole avoidance routine is triggered once a circle of the expected radius within a certain distance in front of the robot has been reported. A diagram of the routine triggered can be found in

Figure 4. We use data from our shaft encoders to drive in a half-box around what we determined was the pothole.

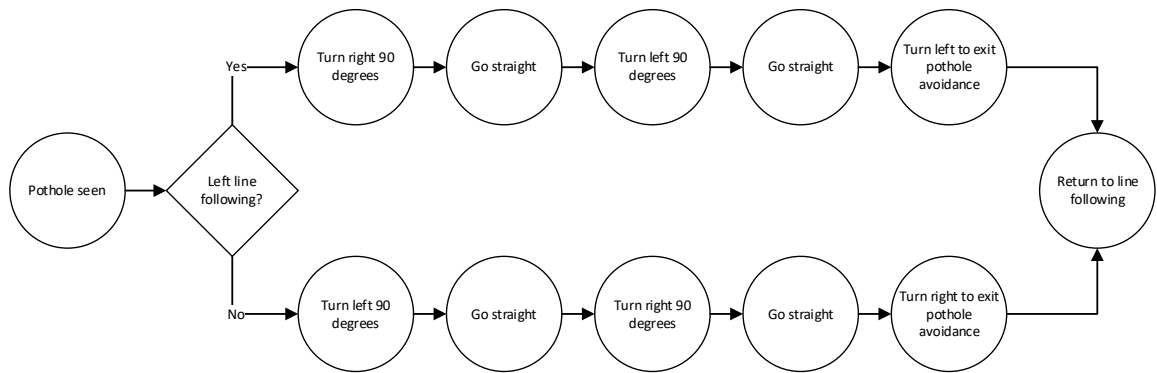


Figure 4. Pothole Avoidance Routine Diagram.

We recognize that this is not a very dynamic solution. We decided that being able to handle pothole obstacles in very limited cases was better than not being able to handle them at all.

DESCRIPTION OF MECHANICAL DESIGN

Overview

Other than minor additions including a case for the IMU and a mount for the GPS ground plate, Delta Bee’s mechanical design has not changed from previous iterations. The major mechanical design is sound and has proven to be reliable and robust.

Decision on Frame Structure, Housing, Structure Design

When designing the robot frame, careful consideration was given to the placement of sensors, weight distribution, and ease of access for internal electronics. Only two drive wheels were used to provide the most maneuverability with a back caster wheel for stability. The frame was made to be lightweight and provide the greatest flexibility for future design.

The primary concern when designing the chassis was that the robot would be able to house all the needed sensors and allow them to be placed at ideal angles and positions. To accommodate this, the robot was only given a skeletal frame that would be able to house the components. The frame was made from ½” square steel tubing according to Figure 5. The frame is lightweight and sturdy and allows for easy access to the inside components, making removal, movement, and addition of parts quite easy.

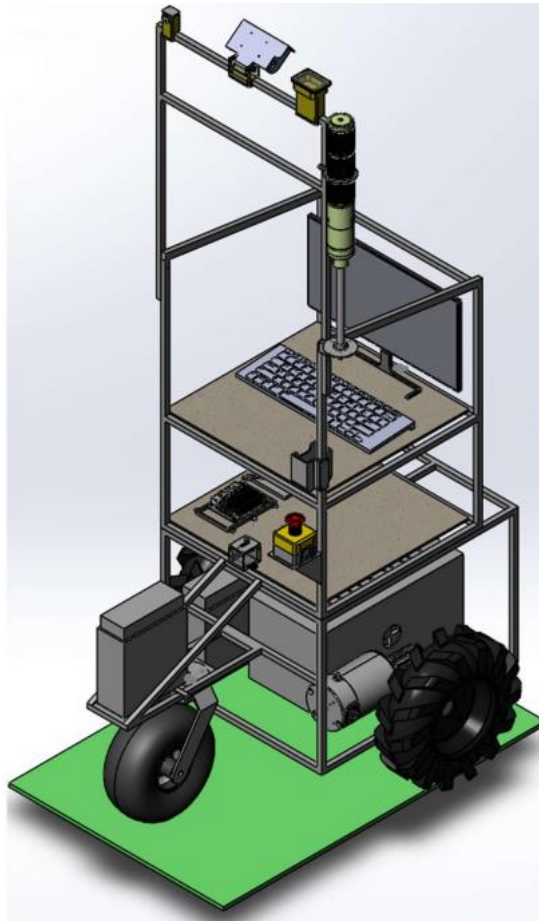


Figure 5. Delta Bee Chassis.

The general structure was made to be as tall as possible within specifications in order to give a clear vantage point for the RealSense Depth Camera which is being used for detecting lines and objects. This also allows our GPS to be placed high and away from the main chassis, minimizing interference in their measurements. The back castor wheel gives stability to the weight distribution and moves the center of gravity towards the back of the robot, helping to prevent tipping on inclines.

The camera, GPS, IMU, and monitor are all secured to the robot via 3D-printed custom plastic housings or clamps. These give a professional look and provide ample strength for each need.

The payload was remade this year to reflect a weight in keeping with the rules. Our previous payload was significantly overweight. It is designed to fit on the bottom shelf of the robot near the motor controllers, keeping much of the weight low to prevent rocking. The batteries were also kept low to keep a low center of gravity. One battery was placed in the back to shift the center of gravity behind the drive wheel axis.

Suspension

The soft rubber tires provide sufficient mechanical suspension for the robot. Rather than build in an advanced mechanical suspension system the robot was found to handle most small bumps and obstacles with minimal bouncing.

Weather Proofing

For weather proofing, custom cut acrylic shielding fits around the robot to provide light rain resistance. The front of the robot has a hinged panel that can be swung up to insert the payload and access the electronic components.

DESCRIPTION OF ELECTRONIC AND POWER DESIGN

Overview

Delta Bee is powered by two 12 volt, 20 Amp hour rechargeable SLA batteries. These batteries provide many hours of reliable robot operation. All sensors and devices connect to a central computer, though some of which are first processed by a microcontroller.

Power Distribution System

The power source for Delta Bee is two 12V batteries connected in series, providing up to 24V for devices requiring higher voltages. The main computer requires 19 Volts and is powered through a 24V to 19V DC to DC converter. The motor controller is powered directly from the 24V power-line. Most of the sensors are either USB powered (through the computer) or use the 3.3V generated by the Teensy 4.0. A diagram of the power connection and distribution can be found in Figure 6.

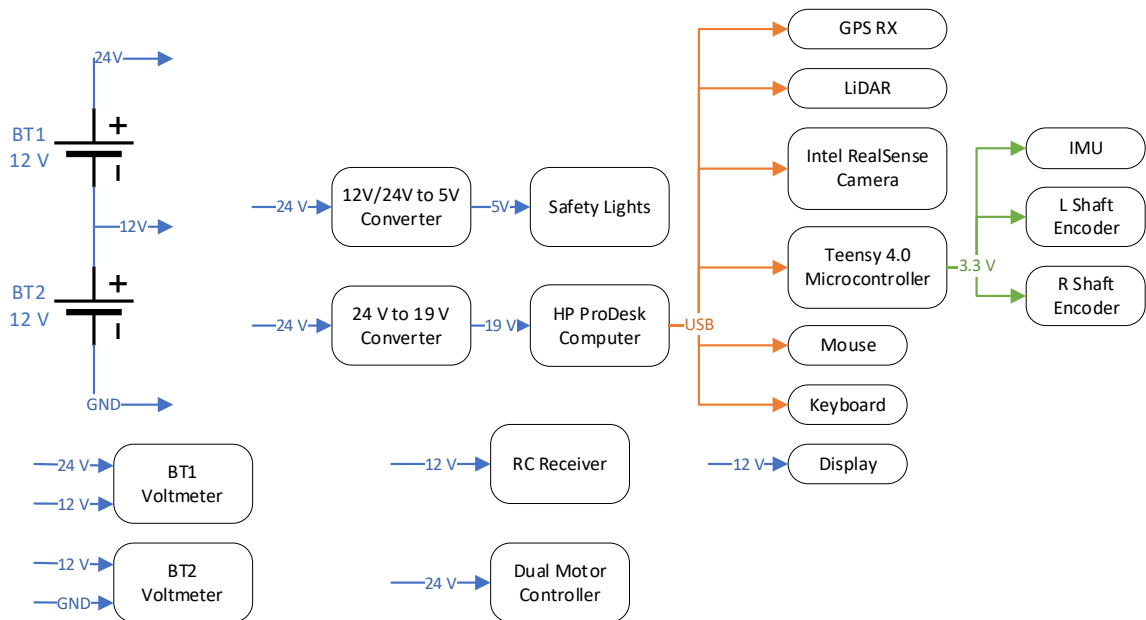


Figure 6. Delta Bee Power Distribution System Diagram.

Electronics Suite Description

All of our sensors communicate directly with Delta Bee's computer or the Teensy 4.0 microcontroller, which in turn communicates with Delta Bee's main computer. Delta Bee's main computer is an HP ProDesk. An Intel RealSense camera provides the visual data needed for line following and pothole detection. Our ZED - F9P GPS Receiver provides absolute position and heading information. Our RPLiDAR sensor provides distance to obstacle information. The Teensy 4.0 microcontroller communicates with the optical shaft encoders to provide wheel odometry information to aid in position and heading estimation. The Teensy microcontroller also communicates with our BNO055 Absolute Orientation Sensor to provide heading information. In addition the Teensy

receives communication from the main computer to active lights for safety, and communicate commands to the motor controller. A signal diagram can be found in Figure 7.

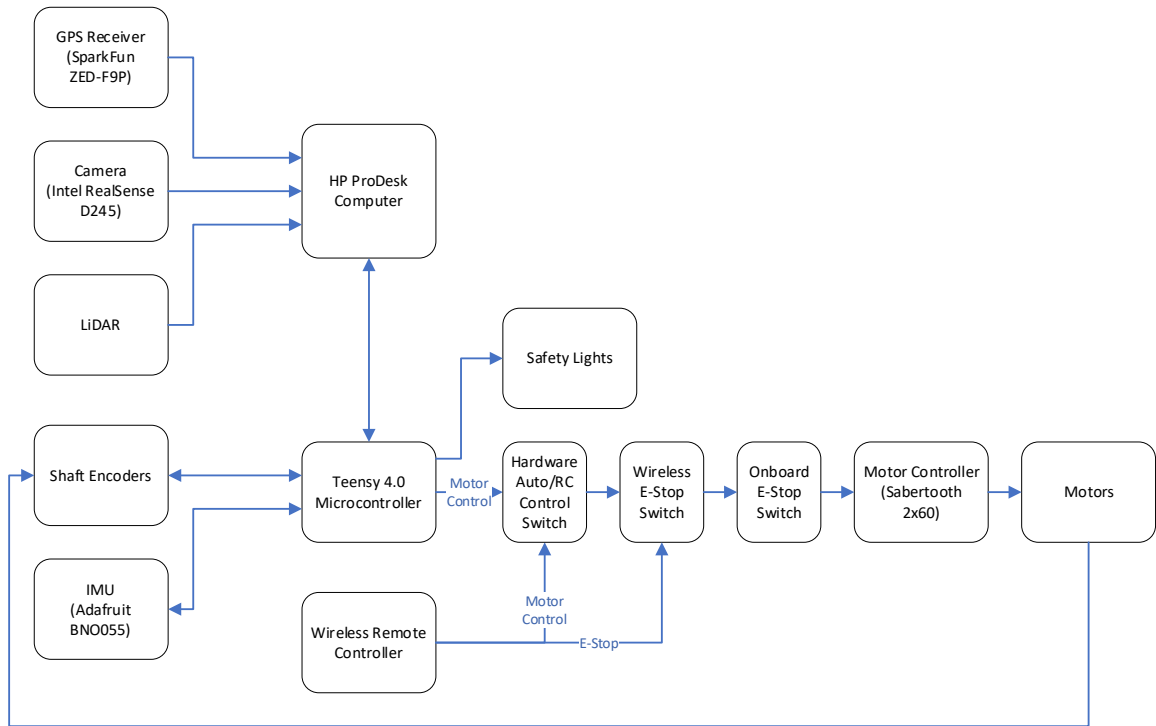


Figure 7. Delta Bee Sensor Diagram.

Safety Devices and Their Integration

Delta Bee’s main safety devices are the onboard and wireless E-Stops and the safety light tower. The two E-Stops function like two switches in series. If either one of them is triggered, the power line to the motors is disconnected, causing the wheels to immediately stop. The onboard E-Stop is a push button; when the button is depressed, the power is disconnected. The wireless E-Stop is simple, reliable and effective. A standard servo activates a light switch; when the remote controller wirelessly moves the servo, the servo flips the switch disconnecting all power to the motor controller. A diagram of the E-Stop hardware can be found in Figure 8.

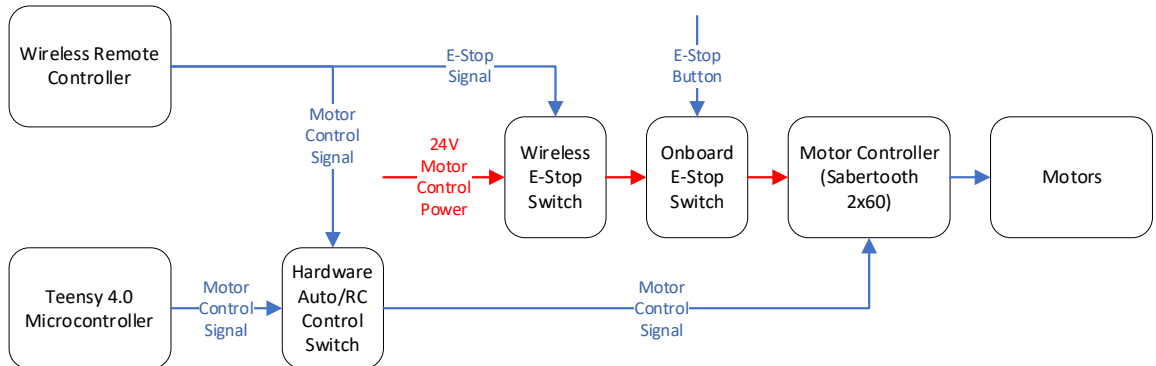


Figure 8. Delta Bee E-Stop Diagram.

Delta Bee also has a safety light tower. It helps indicate to bystanders what navigation mode it is in. When the light is off, Delta Bee is off. When the red light is solid, Delta Bee is in remote-control mode. When the red light is flashing, Delta Bee is in autonomous mode.

DESCRIPTION OF SOFTWARE STRATEGY AND MAPPING TECHNIQUES

Overview

We utilized the Robot Operating System (ROS) framework for connecting the various software components of our robot system. Each of our input and output devices are initialized as a ROS node. We also have a main node which is set up as a finite state machine (FSM) and functions as the primary controller for the robot. We have not implemented a global mapping algorithm; rather, our robot navigates the course solely using the sensors and logical state transitions. We are able to know the distance traveled along the course and when the GPS waypoints have been reached.

Obstacle Detection and Avoidance

To detect obstacles other than potholes (mainly barrels and barricades), Delta Bee uses the LiDAR sensor. When the LiDAR detects an object within a reasonable distance in front of the robot, it triggers an obstacle avoidance state change. The robot turns and then maintains a certain distance away from the object using a proportional controller. Then, once there is no longer an obstacle seen, the robot re-orientes to the line and returns to line following. The same basic structure is also implemented for GPS navigation, except the state machine will return to GPS navigation mode instead of line following mode.

To detect pothole obstacles, Delta Bee uses the RealSense Depth Camera. Once a pothole is detected at the correct distance in front of the robot, it triggers a pothole avoidance state change. The robot executes the maneuver described earlier (see Figure 4) and then returns to line following. The same basic structure is also implemented for GPS navigation, except the state machine will return to GPS navigation mode instead of line following.

Software Strategy, Path Planning, Map Generation, Goal Selection, and Path Generation

Our robot takes a strategy of using a Finite State Machine (FSM) to navigate the obstacle course. Rather than planning and mapping a path through the obstacle course, the robot will handle obstacles in the course as it encounters them. This strategy of focusing on what is directly in front of the robot helps to simplify the software and avoid complex computations. A high-level state machine diagram is shown in 9, and an in-depth state machine diagram is shown in Figure 10.

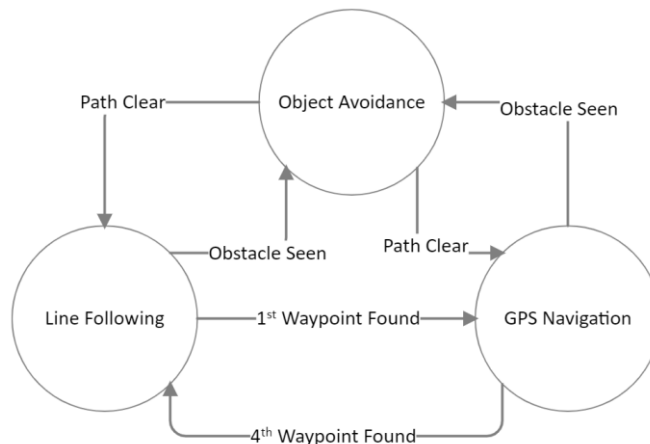


Figure 9. High-Level State Machine Diagram.

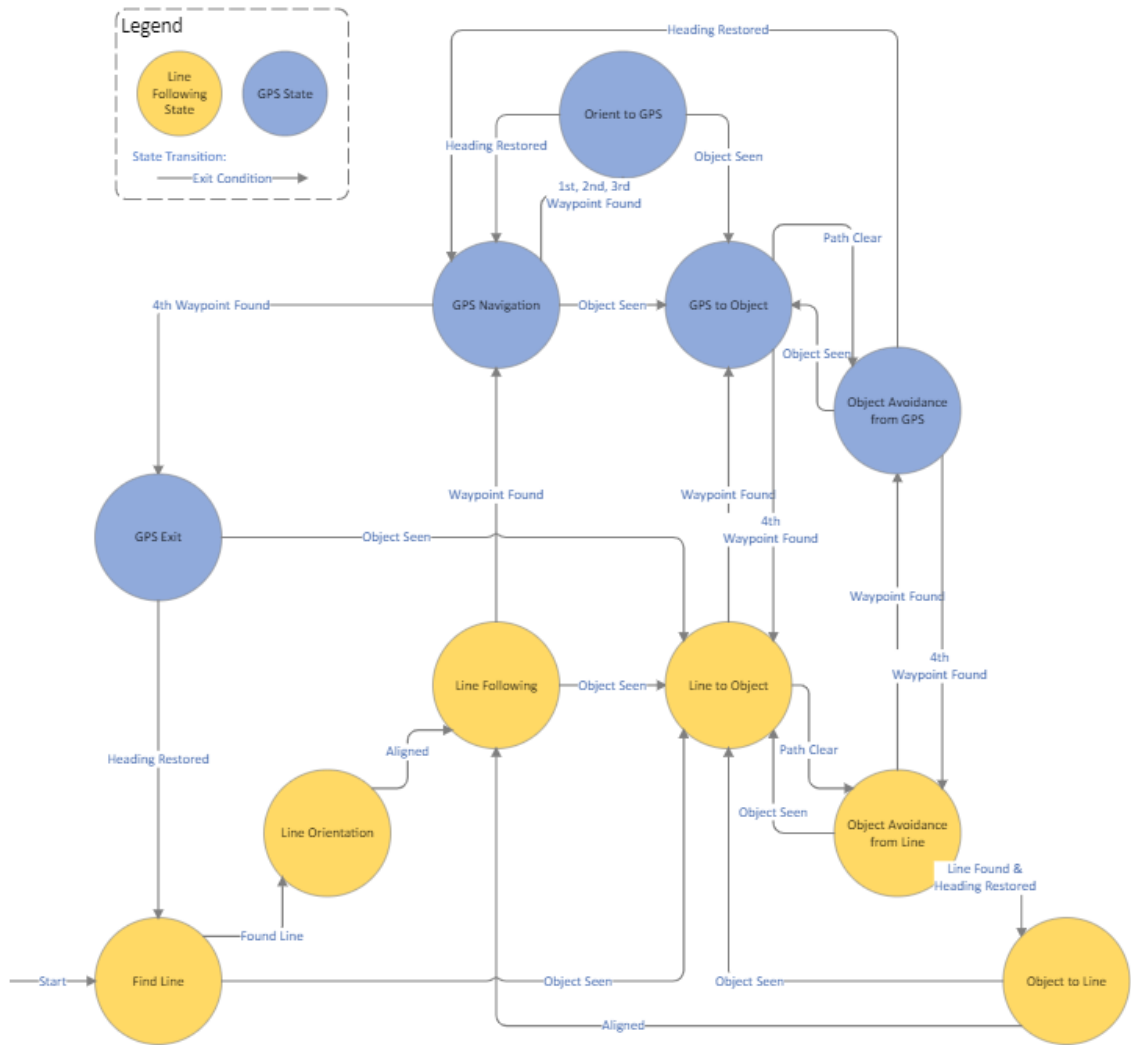


Figure 10. Detailed State Machine Diagram.

The robot begins in line following mode, switching to and from obstacle avoidance when required. The robot chooses its path based on where the line is in front of it. This continues until the first GPS waypoint is found; then, the robot switches to GPS navigation mode. The robot navigates the GPS waypoints in order, switching to and from obstacle avoidance when required. Once the final GPS waypoint has been found, the robot switches back to line following mode (with obstacle avoidance) until the end of the course is reached.

DESCRIPTION OF FAILURE MODES, FAILURE POINTS, AND RESOLUTIONS

Vehicle Failure Modes and Resolutions

The main failure in the software that could arise would be if the state machine gets out of sync with the course. If a state is entered by accident (due to an invalid sensor measurement or inadequate logical processing of the sensor data), the flow of the run could be disrupted, resulting in undefined behavior. For example, if the robot thought it saw an obstacle when there was no obstacle in front of it, that would cause the robot to turn away from the line it was following. We attempted

to minimize the effect of these types of errors by keeping all state transitions based on environment stimuli and by requiring multiple transition flags before changing states.

The other event that is most likely to cause a software fault is if one of the cues to leave a state is missed causing the robot to stay in the same state rather than leave when it should. We minimize this kind of failure by carefully setting the sensor read rates and keeping the speed of the robot at a level where we should always be able to detect a noteworthy change in the environment.

One last kind of failure that could happen is for one of the ROS nodes to shut down unexpectedly. This is handled by the software because ROS can detect if a node shuts down and can then attempt to restart the node (which usually takes about a tenth of a second). This keeps the robot from having to finish the run without an important piece of software such as the node that detects objects.

Vehicle Failure Points and Resolutions

One source of hardware failure would be loose connections in the wiring. These were mitigated by good solder joints and covering the solder joints with electrical tape and heat shrink wrap. Most other wiring such as the USB and UART lines have been very stable and not prone to errors.

Failure Prevention Strategies

To prevent failures as much as possible both during development and at the competition, the robot was designed with a very modular approach in mind. This helps in debugging and allows for replacing various components or software nodes should something fail. This was accomplished in part by keeping all sensors distinct from one another. Any one hardware component can easily be replaced with a substitute. In using ROS for software management of our system, each distinct software component is given its own node. All the nodes communicate by predefined communication channels and protocols. Not only does this allow each node to be tested individually, but it also makes it easy to debug and find failures.

Testing

The robot was tested incrementally throughout the entire design process. This was accomplished through real-time tests and small demo runs. For both software and hardware, most testing was first done in our lab with conditions that approximated as closely as possible the expected conditions on the course. Once this phase was completed, the robot was often taken outside to a improvised obstacle course and allowed to run on the course to test various functionalities. The process was followed during the entire design process to facilitate the testing of new code and its integration with the existing system. Our robot has proven to be highly reliable with dozens of hours of testing without error or breakage. Keeping the batteries charged is the only significant maintenance issue.

Vehicle Safety Design Concepts

The robot was designed to be as safe as possible for all bystanders. The primary means of these safety considerations was in the form of the E-Stop mechanisms, which shut off power to the motor controller at the user's discretion. Additionally, the robot is also made to move at a slow enough speed that it will not pose a significant safety hazard to anyone in the nearby vicinity. Different speeds are set for the various navigational modes to ensure the robots safe operation.

SIMULATIONS EMPLOYED

Simulations in Virtual Environment

We did not perform testing in a virtual environment.

Theoretical Concepts in Simulations

We used MATLAB extensively to simulate and test different algorithms. We used it particularly when evaluating and testing our new obstacle detection algorithm. We ran actual data we logged from our robot's LiDAR through a MATLAB implementation of the algorithm to test whether it truly could detect and avoid barricade obstacles.

We also used it when evaluating different filters to use on GPS data. We used both MATLAB-generated and actual data logged from driving the robot around manually to test MATLAB implementations of these filters. These tests helped us determine the filters usefulness before implementing them in ROS, thus saving time and improving performance.

PERFORMANCE TESTING TO DATE

We have tested each of the new subsystems extensively separately. Delta Bee can successfully navigate using GPS and takes a straight path to the next point instead of a roundabout curved path. Delta Bee can successfully avoid barricade and pothole obstacles while maintaining its existing barrel avoidance functionality. We currently are continuing to work on integration testing to ensure everything is ready for competition.

INITIAL PERFORMANCE ASSESSMENTS

From our progress so far, Delta Bee is on track to be ready for competition in June. The improved GPS navigation and the addition of barricade and pothole avoidance put Delta Bee on track to be more reliable and successfully complete the course more times than at last year's competition.