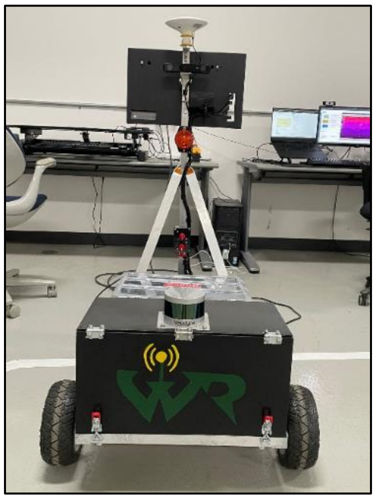
Warrior Robotics Team

Aasha | IGVC 2022



Faculty Advisors

Dr. Abhilash Pandya [ECE] | apandya@wayne.edu

Dr. Marco Brocanelli [CS] | brok@wayne.edu

Dr. Azad Ghaffari [ME] aghaffari@wayne.edu

Team Captains

Nnamdi Monwe [CS] | President | nnamdimonwe@wayne.edu

Lloyd Brombach [CS] | Vice President | lloyd@wayne.edu

Keena Pandya [ISE] | Managing Director | kpandya@wayne.edu

"I certify that the design and engineering of the Wayne State University Robotics Team has been significant and equivalent to what might be awarded credit in a capstone design course."

Abhilash K. Pandya, Ph.D.
Professor, Department of Electrical & Computer Engineering

A K Pandya

Marco Brocanelli, Ph.D.
Assistant Professor, Department of Computer Science

[illegible]

Azad Ghaffari, Ph.D

Assistant Professor, Department of Mechanical Engineering

Azad Ghaffari

Table of Contents

1. Our Team

Warrior Robotics is a student organization housed under the College of Engineering at Wayne State University. We are a team of engineering students comprising both undergraduate- and graduate-level students with a wide variety of disciplines this is shown in the table below, Table 1.

Table 1: Table of Student Members

Student(s)	Department
Nnamdi Monwe	Computer Science
Lloyd Brombach	Computer Science
Mark Slattery	Computer Science
Arun Jayaramen	Computer Science
Nini Ola	Computer Science
Manavendra Desai	Mechanical Engineering
Maysara Elazzazi	Robotics
Luay Jawad	Robotics
Abhishek Shankar	Robotics
Gowtham Kanneganti	Robotics
Keena Pandya	Industrial and Systems Engineering
Venkata Sirimuvva Chirala	Industrial and Systems Engineering
Dimitri Van Well	Electrical and Computer Engineering
James Yoon	Electrical and Computer Engineering
Alexander Politis	Electrical and Computer Engineering

This year, we are introducing a new vehicle AASHA for IGVC 2022. We focused on using the detailed lessons learned and advice from previous years by creating a lighter frame and adjustable system with improved sensor placement techniques. We modeled our team using an AGILE production method with scrum meetings using the Trello Kanban boards as task management tools. Using this pull system, we had the ability to maneuver around the non-deterministic needs of our robot throughout our development cycle, providing clear deliverables and tasks for each of our team members.

2. Design Process Overview

The team adheres to a flexible design cycle to develop solutions to our software and hardware-related problems. Utilizing the design cycle allows for a clean and repeatable path to creating a solution that would otherwise require a longer time to completion. It enabled us to look at failure modes from an engineering-first approach instead of a fix-as-you-find approach at every stage of Aasha's development. We ensure to use this cyclical process before creating our backlog tasks in our Kanban board within our AGILE framework. The design and development cycles are shown below in Figure 1.

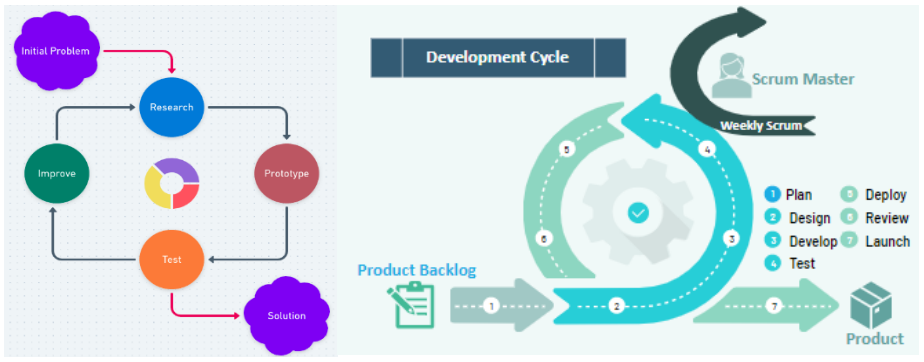


Figure 1: Process flow of design process and AGILE Production System

3. Mechanical Systems

This year we focused on creating a more modular design with a lighter and simpler frame to allow for better maneuverability, flexibility, and modularity. This approach allowed us to add on systems to our platform in a systematic way. We divided our mechanical system into four key components: the chassis design, drivetrain, sensor placement, and weatherproofing.

3.1. Chassis Design

We wanted our design to incorporate a minimal payload bay with easy access to stow the competition payload. We also wanted a flatbed design to accommodate the storage of electronics and other critical components in a modular manner, and we wanted a reinforced mast to hold our sensors and screen. We began the CAD design process once we created a needs checklist to satisfy our robot's critical function. We experimented with sensor and payload placements during the CAD design phase until we were confident that the design met our criteria. After validating our CAD model and going through several design iterations, we began the fabrication process. The results from our research and implementation realized an 8 percent smaller displacement than the previous robot and being approximately 12 pounds lighter.

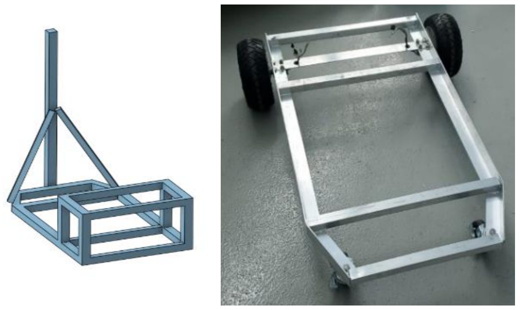


Figure 2: CAD Model of Chassis Frame and Lower Chassis Build Picture

3.2. Drivetrain

Unlike our previous designs, we decided to use a set of compact brushless hub motors for our drivetrain. These hub motors, shown in Figure 3, are essentially high torque motors that are embedded into a wheel with a reinforced hollow axle to bear the weight of the robot's load. The axle is fixed to the motor's stator, and a set of hall effect sensors are positioned in a perpendicular manner which is used for reading changes in magnetic fields and sending them to our onboard controller. The outer part of the wheel is attached to an aluminum frame with a set of alternating permanent magnets and a solid rubber tire that act as the motor's rotor. Utilizing a hub motor for our drivetrain makes it more robust and modular as we have fewer moving parts and only need to detach the axle clamp to perform maintenance to the system.

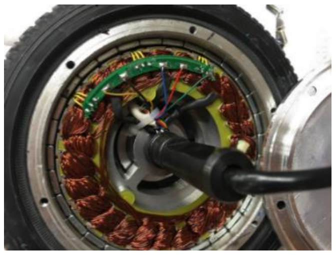


Figure 3: Image of Brushless hub motors used in the drivetrain

3.3. Sensor Placement

Another critical area we overlooked that hindered the performance of our previous vehicles is the placement of our sensors this is shown in Figure 4. Our lower latency sensors and instrumentation that require USB access are all housed within reasonable proximity to the host computer. We made sure to make the mast tall to accommodate the GPS Antennas and help increase our ZED camera's field of view. We also expanded the frontal area of the chassis to accommodate a 3D Lidar without having to modify the chassis.

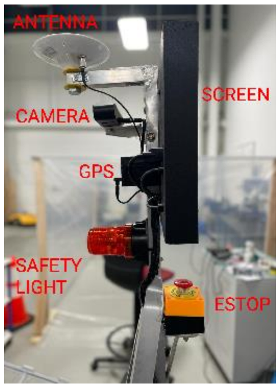


Figure 4: Sensor Placement on Robot Mast

3.4. Weather Proofing

To ensure we created a functional machine focused on weatherproofing for adverse weather conditions, we utilized IP66-rated enclosures to house our batteries, laptops, smaller sensors, and wires within our robotic system. In addition to this, external sensors such as the ZED Camera and the Velodyne Lidar, that were not housed within the enclosures are rated with an IP66 rating to ensure that they can survive adverse outdoor weather conditions. The monitor that is fastened to the mast of the robot is protected by using a plastic partition secured to the mounting platform of the monitor. Lastly, we ensured that the wheel bearings of our brushless hub drive motors were sealed to ensure that they do not experience wear and tear due to outdoor conditions.

4. Electrical Systems and Power Distribution

Aasha uses two 36-volt, 20 amp-hour lithium-ion battery packs. The first battery pack provides 36 volts to the motors and its control board and 12-volt and 5-volt DC-DC converters that power the rest of the auxiliaries. The second battery is dedicated to powering the laptop via a 19-volt DC-DC converter. Both batteries can be easily changed with XT-60 quick connectors or charged in place with weather-resistant connectors mounted on the shell of the lower electronics compartment. The laptop does retain its factory internal battery, allowing for the auxiliary 36-volt battery to be changed without powering down. Each battery has its own disconnect switch.

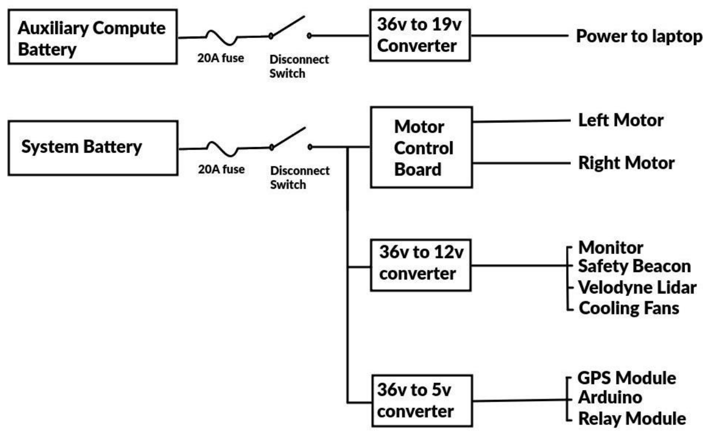


Figure 5: Auxiliary and System Battery Power Diagrams

4.1. The Electronics Suite

Aasha uses a Lenovo Legion 5 gaming laptop with an eight-core, 3.2Ghz AMD CPU, an Nvidia GeForce 3050Ti GPU, and 32 GB of RAM for the main processing unit. Automatic power control to the motor driver board, Velodyne Lidar, cooling fans, and a safety beacon is provided by relays and an Arduino microcontroller interfaced to the main computer running a custom ROS node to manage it.

The motor driver board is a repurposed hoverboard control board that has been reprogrammed with firmware from an open-source ROS hoverboard project. Communication with the motor driver board, which also provides odometry feedback to the main computer, uses USB serial and an FTDI USB to TTL serial converter.

The sensor suite consists of a Velodyne VLP-16 360-degree 3D LIDAR, a ZED2i stereo depth camera, an embedded IMU, hall effect sensors (embedded in the brushless DC motor/wheel unit), and a Sparkfun GPS-RTK module. The computer auxiliaries communicate via USB, except for the Velodyne VLP-16, which uses ethernet.

4.2. Power Requirements

For the power requirements for Aasha, we separated the criteria into the main battery requirements and the laptop battery requirements.

Table 2: Aasha Power Requirements table

Main Battery Requirements		**Laptop Battery Requirements**
Total Required	340 watts	Max consumption 230 watts. 160 watts typical.
9.43 typical amps required.		
Limiting discharge of our 20 AH main battery to 50%, we can expect 10AH/9.43A = 64 minutes of runtime per charge.		Battery duration is 2.65 hours typically. 1.8 hours minimum.

4.3. Recharge Rate

The chargers supplied by the battery manufacturers charge the batteries at two amps. Given our 50% depth-of-discharge maximum, 5 hours are required to achieve a full charge. To ensure minimum interruption in operations, we have acquired multiple batteries and can quickly swap batteries with quick connects—further, the laptop's internal battery charges from the auxiliary laptop battery. The computer does not need to be powered down to swap the auxiliary laptop battery. The batteries can be changed, and the robot is put back in service in less than two minutes.

4.4. Safety

Our robot complies with IGVC Auto-Nav rules standard and has a mechanical and wireless E-stop. The two emergency stop systems work both independently and in tandem to ensure proper vehicle disengagement in the event of an anomaly. Both local and remote E-stops are wired in series, so either can interrupt the control signal to the motor control board. Without a control signal, the control board powers down the motors.

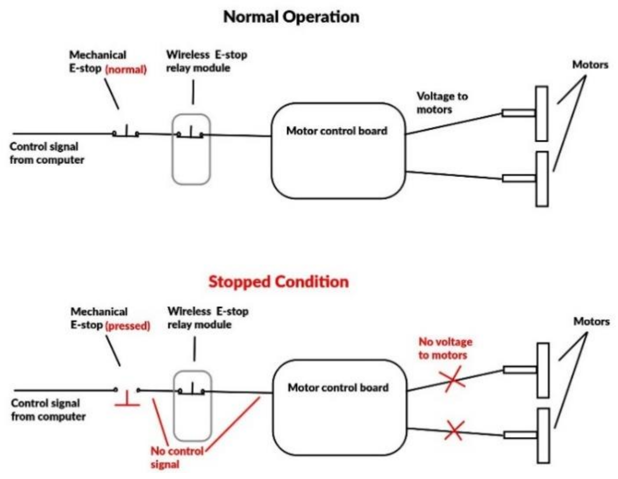

Normal Operation
Mechanical E-stop (normal)
Wireless E-stop relay module
Motors
Voltage to motors
Motor control board
Control signal from computer
Stopped Condition
Mechanical E-stop (pressed)
Wireless E-stop relay module
Motors
No voltage to motors
Motor control board
Control signal from computer
No control signal

Figure 6: EStop Wiring Configuration Diagram

In addition to compliant E-stops, we decided to issue all movement commands through the ROS navigation stack instead of directly through our own software instead of issuing lane-following speed and steering commands directly. This adds the complexity of calculating coordinates to pass to the navigation stack but provides an extra layer of obstacle avoidance through a much more mature software than our own.

5. Software System

Building on previous years' experience, we again implement a Robot Operating System (ROS) software stack to encapsulate different tasks and handle data traffic between the many software components. We leverage the ROS navigation stack and behavior trees in a way that does not require us to map the entire course. Instead, Aasha uses snapshot maps of its immediate surroundings to develop movement plans that incrementally take the system along the course until it is close to a GPS waypoint. The lane–following branches of the behavior tree are suspended, and the navigation stack is commanded to find the GPS waypoint. However, lane marking data are still plotted as local obstacles to be avoided.

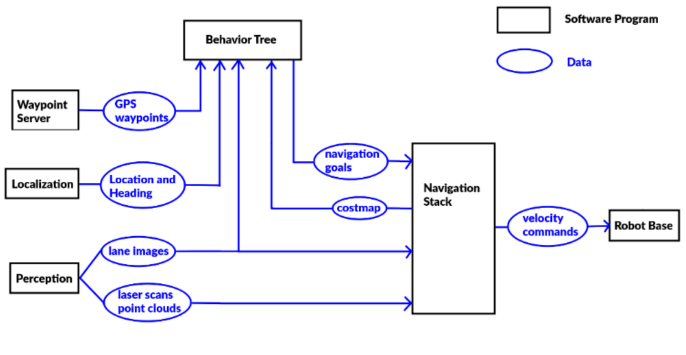


Figure 7: Software Architecture Overview

5.1. Obstacle Detection and Avoidance

The Velodyne LIDAR unit easily detects physical obstacles (barricades, barrels – anything with height projecting up from the ground). The Velodyne passes a point cloud of data to the ROS navigation stack that plots obstacles on a cost map relative to the position of the robot when a given scan is received. The Velodyne is capable of detecting obstacles out to 100 meters. Still, the mapping software is configured to ignore the barriers greater than 10 meters away to conserve RAM and processing power.
Detecting lane-marking obstacles and potholes is a more challenging task that we accomplished with a combination of techniques. Lane markings were first found using YOLOP which is an open-source panoptic convolutional Neural Network. The model was trained using the BDD100k dataset which has a very diverse pool of training data, so it is very accurate and doesn't need to be retrained or fine-tuned.

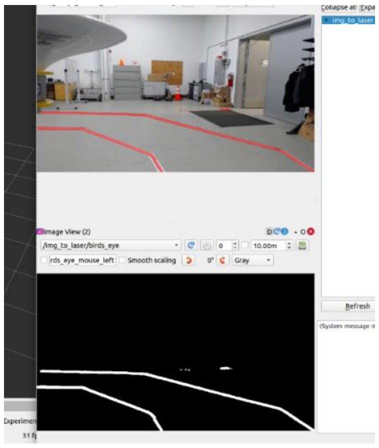


Figure 8: Lane Marking using YOLOP

The YOLOP package doesn't work natively with ROS. Thus, it needed to be modified for our specific application. We converted the package into a node that accepts incoming frames from the Zed2 camera as an Image message. The node then performs the segmentation using the neural network. Then a binary image is returned to us, which will later be converted into a laser scan topic for easier integration.

Pothole detection

The image taken from the right-hand side camera of the Zed2 is used to detect the circular potholes. Zed2 is coupled with ROS to subscribe to and publish pothole-only images for obstacle avoidance. OpenCV image processing and computer vision libraries are utilized to detect and isolate potholes in the field of view of the Zed2.

A Zed2 RGB (red-blue-green) image is first converted to a grayscale image. Using a tuned thresholding parameter, binary thresholding is applied to correct the grayscale image into a binary image (black and white only). At this stage, both lanes and potholes will be detected in white. An elliptical mask of kernel size three is iteratively applied over the binary thresholded image to isolate the pothole. Since the circular pothole, and not the rectangular lane, appears as an ellipse to the Zed2 camera, the mask eliminates the lane and yields a pothole-only binary image. This image is published to a ROS topic that converts the detected pothole to laser-scan data for obstacle avoidance. Sample results of the algorithm developed are shown below.

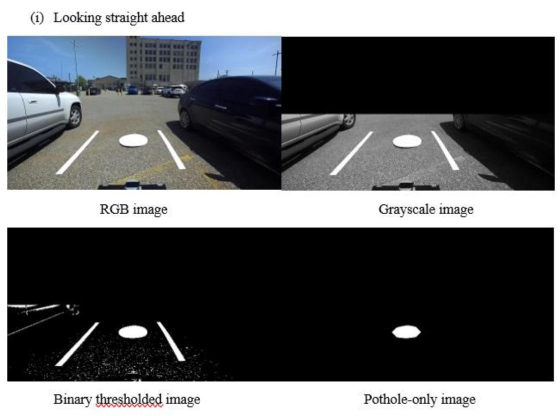


Figure 9: Image looking straight ahead of camera view

In the scenario above Figure 9, the robot is looking straight ahead. Even in the presence of a white car on the left, only the white pothole is extracted in the pothole-only image. The elliptical mask used to isolate the pothole successfully removes the rectangular lanes and noise that appear in the binary thresholded image. To aid pothole detection, undesired bright objects situated far in the scene are eliminated by zeroing the pixels in the top half of the picture. This results in the black rectangular space seen in the grayscale and binary thresholded images. Lastly, it is ensured that the non-zeroed portion of the image always contains the entire path generated by the motion planner.

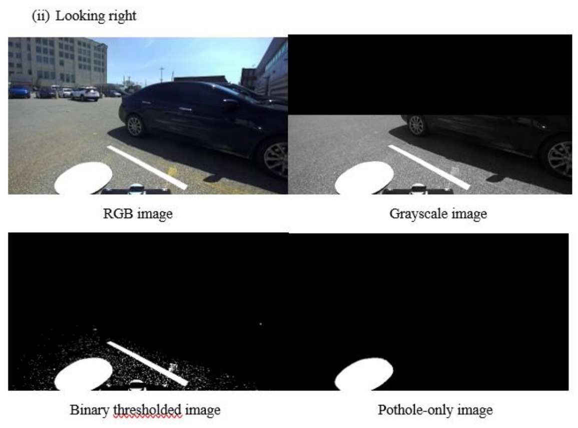


Figure 10: Image looking to the right of the camera view

In the scenario above Figure 10, the robot is looking right. Once again, only the white pothole is extracted from the scene.

(iii) Looking left

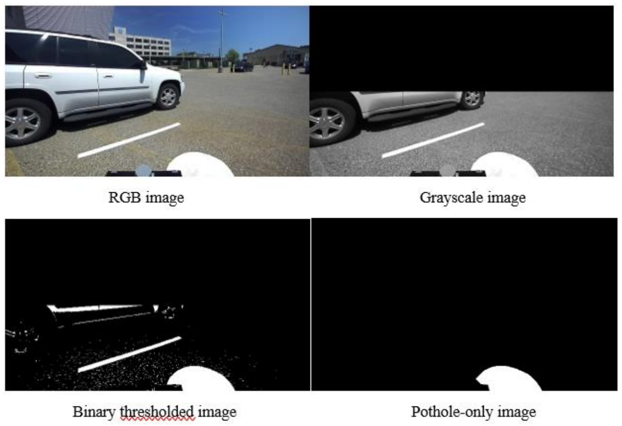


Figure 11: Image of the left of the camera view

In the final scenario above, Figure 11 , the robot is looking left, straight at the white car. However, only the white circular pothole is extracted from the scene. This can be attributed to using an elliptical mask that eliminates the white rectangular side of the car. The result is a black and white image of only the pothole.

5.2. Perspective Transform

The separate lane and pothole result images are merged into a single image that undergoes a perspective transform to obtain a top-down / birds-eye view of the lanes.

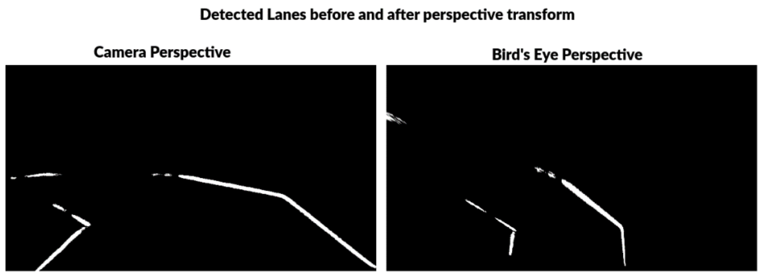


Figure 12: Detected Lanes before and after perspective transform

The transformed image is then passed to the laser scan conversion function, which creates a standard ROS sensor_msgs::LaserScan message from the image by calculating the distance and angle of each pixel from the transformed image relative to the robot.

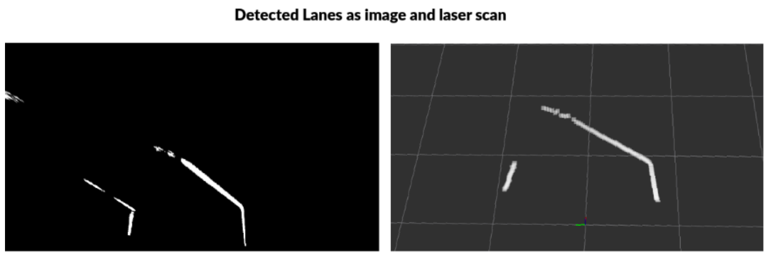


Figure 13: Detected Lanes as Image and Laser Scan

The result is a laser scan message that the ROS navigation stack understands and plots on the same cost map on which the Velodyne obstacle data are plotted. This cost map is used by the navigation stack’s A* global planner and DWA_Local_Planner to seek a goal within the mapped area while avoiding all obstacles. Obstacles detected by all sources are plotted, and the navigation stack reacts within .1 seconds. A maximum time of .1 seconds is also required for the motor controller and driver to respond. The total time to respond to new obstacles is a maximum of .25 seconds, including the scan and processing rate of the sensors and detection algorithms.

5.3. Mapping and Localization

No attempt is made to map the entire course, and there is no need to localize relative to the course. Instead, Aasha uses the Costmap2d plugin in the navigations stack to create a rolling-window map of the immediate surroundings and is always considered the center of that map. Obstacle data is sent to the Costmap2d as a point cloud from the Velodyne Lidar and a laser scan message generated from lane and pothole detection data. Goals sent to the navigation stack are always relative to the robot, i.e., "Navigate to a spot that is three meters forward and two meters to the left of your current position." When seeking or testing proximity to a GPS waypoint, transform data is used to make the necessary conversions.

RTK-GPS and a heading solution from fused visual, inertial, and magnetic sensor data are used to find Aasha's position and orientation relative to GPS waypoints. The waypoint list includes the finish line, which is saved as a waypoint upon the start of every run. GPS latitude/longitude waypoints and fix coordinates are both converted to UTM coordinates, which are expressed in meters. Since meters are the distance unit convention used by ROS software, this allows direct use by both the navigation stack and our own navigation software. Our GPS is accurate to 20cm with correction data, although the software is set to accept a 50cm tolerance when determining whether a GPS waypoint has been reached or not. Testing has shown that Aasha can reliably navigate to GPS waypoints within 50 cm.

5.4. GPS

Our GPS implementation consists of 2 components: the GPS board and NTRIP. For the board, we used a SparkFun Dead Reckoning RTK-GPS. In perfect conditions and with corrections for satellite errors (drift, atmospheric conditions, etc.), this board delivers ~0.2-meter accuracy. Additionally, it maintains a continuous position during poor conditions or complete signal loss, making it an ideal solution for challenging environments, such as dense cities.
To gather correction data, we use a 3rd service called NTRIP, composed of 3 parts: the NTRIP server, the NTRIP caster, and the client rover. The server base sends correction data to the caster, and then the client rover (in our case, the SparkFun GPS) accesses the caster to obtain correction data.

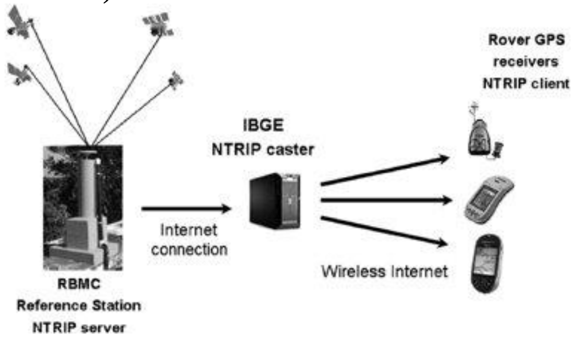

Rover GPS receivers NTRIP client
IBGE NTRIP caster
Internet connection
Wireless Internet
RBMC Reference Station NTRIP server

Figure 14: Relationship between NTRIP server, caster, and client

The correction data is sent directly to the GPS board, which then calculates and returns a global position in terms of latitude, longitude, and altitude at roughly 10Hz. We use ROS to bundle and publish that information as NavsatFix messages used in localization.

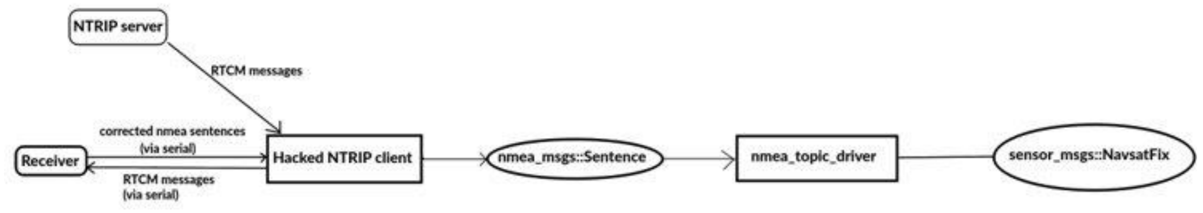


Figure 15: GPS Software diagram

5.5. Goal Selection and Path Generation

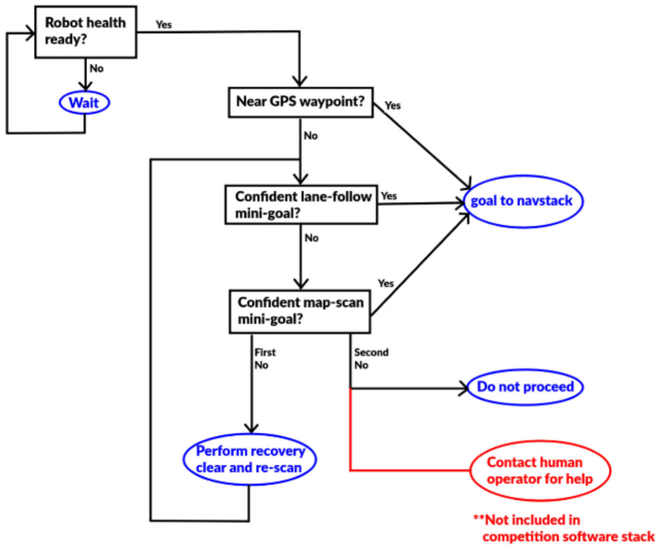

Robot health ready?
Yes
No
Wait
Near GPS waypoint?
Yes
No
Confident lane-follow mini-goal?
Yes
No
goal to navstack
Confident map-scan mini-goal?
Yes
First No
Second No
Do not proceed
Perform recovery clear and re-scan
Contact human operator for help
**Not included in competition software stack

Figure 16: Behavior tree of Goal Selection and Path Generation

While the ROS navigation stack is used for path planning to a given goal, a behavior tree shown in Figure 16 is the backbone of Aasha's decision-making process that decides which goal to issue next. The behavior tree's method is somewhat analogous to a driver on a freeway heading to an unknown place in a distant city. The details of the entire map are unimportant if it is known that the highway has an exit in the general area of the destination or waypoint. A driver would likely start by ignoring all but lane data and obstacles that create an immediate risk of collision. When near a destination or waypoint, or if otherwise disoriented, the driver would then use their senses to observe the drivable areas in greater detail and navigate to the exact waypoint or goal.

The behavior tree first checks with the robot's health monitor to confirm that the necessary sensors are online and reliable enough to carry out the mission. If the health monitor or other data checks indicate a deficiency, a human operator is notified through the GUI interface so the flaw can be corrected. If the decision is made to proceed, a series of wayfinding submodules are queried to select a suitable "mini-goal" location that moves the robot along the course and toward the next GPS waypoint or finish line. The concept of scanning data ourselves to find a mini goal in free space, then sending it to the navigation stack does cost some computational redundancy but comes with the benefit of allowing our software to make high-level decisions while enabling us to leverage the proven ROS navigation stack for low-level navigation details.

The mini goal is selected first by checking if the robot is close enough to a waypoint or simply passing that to the navigation stack as an immediate goal. If not close enough, by analyzing the most current images of lane data and attempting to stay within them. If the confidence in the visual lane data falls below a certain threshold, the behavior tree passes the request to a submodule that analyzes the most current cost map. The cost map includes both lane and obstacle data and has some memory of recently detected lane lines that may have fallen out of the robot's field of view. Suppose this second wayfinding layer returns a result with very low confidence. In that case, a recovery behavior is called to clear the maps and perform a slow-scanning rotation in place, and one more attempt is made. In all cases, the mini-goal is passed to the navigation stack to handle the details of path and trajectory planning.

5.6. Simulation

This year, we used a combination of URDF files, Gazebo, RVIZ, and RQT to implement a simulation environment.

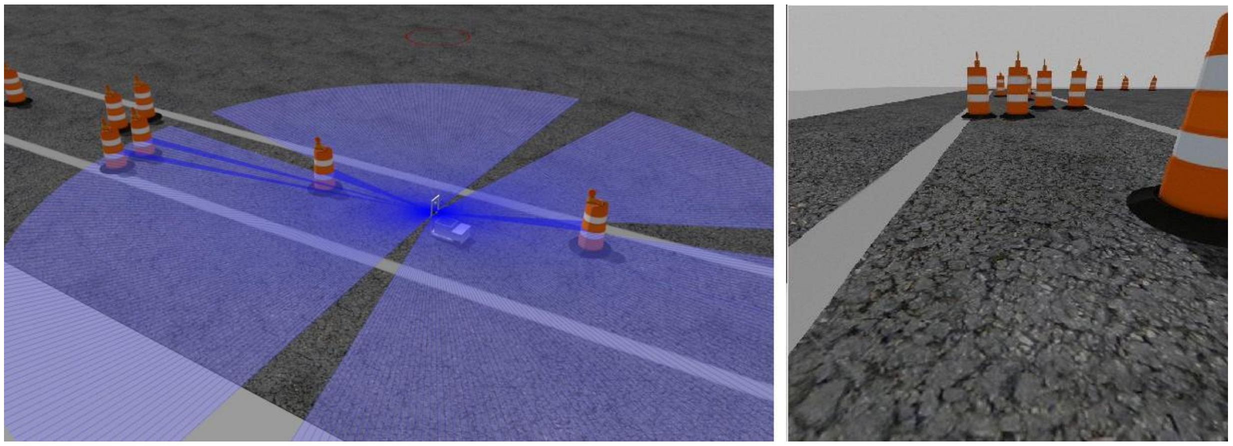


Figure 17: Simulating robot collecting data in virtual world

Our simulation team was able to create a simulated robot that is able to interact with the virtual world just as a physical robot on the actual competition course. Figure 17 shows our simulated robot collecting sensor data of the course from simulated laser scanner and camera. These advances in our simulation abilities allow us to test various algorithms in a safe, quick, and cost-effective manner.

6. Failure Modes

To complete our understanding of actual and potential failure modes, we decided to conduct a full FMEA (Failure Modes and Effects Analysis), found in Appendix A. However, this report will discuss our failure points identification and resolution methods. We created a phased approach to conducting this by first understanding the scope of our failure mode analysis. We separated our failure modes into three key categories to account for all significant potential failure modes that could affect our final goal of the 2022 IGVC competition. The three areas of our scope were: Electromechanical failure, Localization failure, and Vision Failure.

6.1. Electrical and Mechanical Failure Modes

Listed below in Table 3 are the failure points that we identified for the electromechanical subsystem and the subsequent resolution modes that we established to mitigate these actual or potential failures.

Table 3: Electromechanical Failure Points and Resolution Modes

Failure Point Identification	Resolution Modes (Actions Completed)
The laptop Battery is not Charged as battery charge depletes over time	Custom Electrical Stepping Solution to Charge Battery using Lithium-Ion Batteries on Board
Hardware Failures (Any Hardware that breaks)	Buying Backup Hardware and ensuring all modifications or code is uploaded to backup hardware

The monitor is not waterproof	Utilizing a plastic screen when needed to prevent any weather-related damage
Robot Moving/harming individuals in the way	Easy to see Safety Light is mounted with onboard and wireless Estop
Sharp edges harm people	Ensure that sharp edges and Deburred and Filed Down
Batteries exploding - Lithium-Ion / Lead Acid	Having Baking Soda and Proper PPE (Personal Protective Equipment)
Software Crashing for unknown reasoning	Health Monitor, which describes the status of sensors

6.2. Localization Failure Modes

For the localization failure modes, we began to have more specific failures relating to the software that was being integrated into the main system. This is shown in our failure points chart in Table 4.

Table 4: Localization Failure Points and Resolution Modes

Failure Point Identification	Resolution Modes (Actions Completed)
Waypoint file becomes corrupted	Ensure that protected files are written
Files Incorrectly named in the software	Isolate competition files (UTF) test, ensure two names are reserved for the course, Test course
NTRIP Failure that can occur on a large state level for NTRIP	Utilize a permanent Dynamic Reconfigure
Duplicate TF Publishers	Ensure that testing ahead of time was done
Incorrect Launch file tree	Utilizing health monitor listing nodes to show which is being run
Low GPS Hz	Slow down the speed of the robot
Bad Lane to Laser	Outdoor Testing
Not Crossing the finish lane and ending up right before the finish line	Waypoint server starting after it has started
Robot seeing self as obstacle	Utilize a permanent Dynamic Reconfigure
Trajectory planner error where robot hits obstacles	Ensure that proper testing is done and adjustments are made to prevent poor trajectory planning
Extreme LIDAR conditions (bad weather, etc.)	Use backup system of 2D LIDAR rather than 3D LIDAR on board

6.3. Image Processing Failure Modes

For the vision subsystem, the failure modes included both failure modes in the vision integration system as well as failures that could potentially occur with the ZED camera system that is onboard Aasha. The failure points chart is shown below in Table 5.

Table 5: Image Processing Failure Points and Resolution Modes

Failure Point Identification	Resolution Modes (Actions Completed)
Too much glare from sun	Use internal filtering (ZED), angling (moving camera), deep learning -- too much effort, hood over camera (sunglasses), variable ND filters
Moisture on Camera lens	Use enclosures / ingress protection
Run out of Memory	skip frames (minutes of frames we are processing) , YOLOP, zed may have a controllable frame rate, refactor code
Node fails within the pipeline	Feedback to health monitor to see where the failure occurred
Compute overheats	Throttle CPU down / cooling fans

7. Innovations

7.1. Hybrid Lane-Following + Navstack

A hybrid lane-following + navigation stack method that attempts to identify lanes in a conventional manner but sends an output to the navigation stack instead of directly commanding steering. This hybrid method adds a layer of safety because the navigation should refuse to navigate through obstacles our software might miss, such as potholes and obstacles detected by the laser scanner. Additionally, some of our lane-following program's recovery behaviors are greatly simplified by issuing goals to the navigation stack rather than handling velocity calculations and obstacle avoidance internally.

Lane following starts by applying a morphological operation to clean up small gaps and eliminate all but the two largest lane candidates. The location that falls between the end of the two is calculated and passed to the navigation stack. If only one lane is detected, the program picks a spot that is 1.5 meters inside the end of that lane and reports a reduced confidence to the behavior tree.

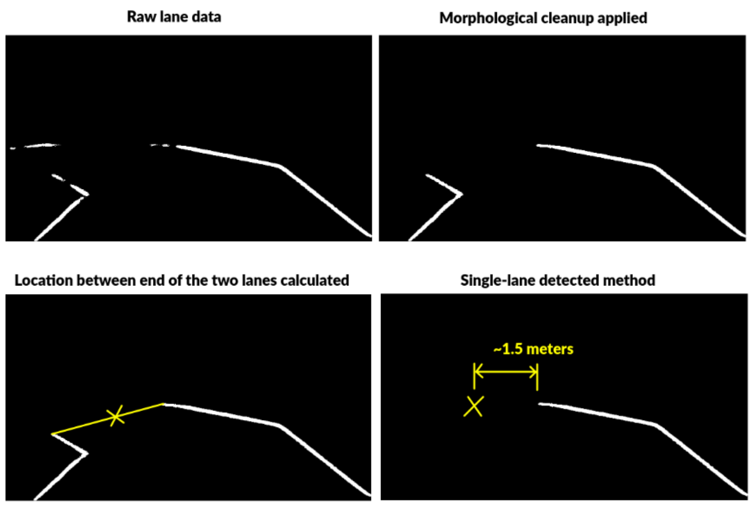

Raw lane data
Morphological cleanup applied
Location between end of the two lanes calculated
Single-lane detected method
~1.5 meters

Figure 18: Lane Detection Program in Action

Should the robot find itself facing a single lane that is too perpendicular to presume a correct direction of travel, it is able to request a rotation toward the next GPS waypoint. This new perspective provides new data for a more-informed decision to be made. If the confidence is still too low, a recovery mini-goal can be issued to the navigation stack simply in the general direction of the next GPS waypoint, and the global planner can use the A* algorithm to consider all known obstacles and proceed toward that mini-goal. Once the robot is past the switchbacks or obstacles that caused the disorientation, the lane-follower program can resume control. For this reason, the recovery mini-goal does not have to fall between the lane markings to be effective.

7.2. Health and System Status Interface Graphical User Interface

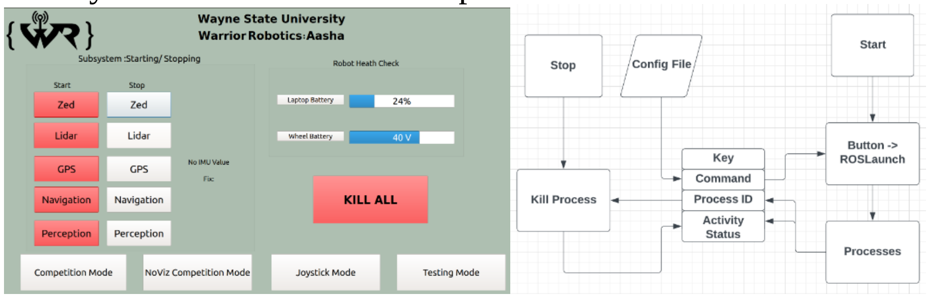


Figure20: (Left) Graphical User Interface design (Right) Process flowchart for GUI

The purpose of this interface is to provide an easy way to monitor the health information of the robot (battery state, GPS fix status, roslaunch process status, etc.). It will also provide the ability to start or stop individual and group components of the robot. Two configuration files are provided in a JSON format that stores the component names as keys with the roslaunch commands as values. These files easily allow the configuration of the roslaunch commands as associated with each button. The system keeps track of the root process IDs of all the roslaunch commands and can easily kill subprocesses to ensure clean starting and stopping of system commands. It also provides a systematic method to start roslaunch files in the correct order. There are also two primary startup modes, each corresponding to a different configuration file. The competition mode starts up the robot with visualization processes, whereas the NoViz (no visualization) Competition mode starts up the robot without these processes for a performance boost. All launch files provide a beacon signal reflected in the GUI as a blinking icon of stable operations.

7.3. Robot Recovery Feature

A final recovery feature is not enabled for the competition mode but is available to align with industry standards for recovery behaviors for commercial autonomous delivery robots. The final recovery behavior has the robot notify a remote human operator that it needs assistance. The operator can then remotely make decisions, manually steer the robot, or dispatch a robot recovery vehicle to bring the machine for service.

8. Key Learning Experiences

The fabrication of Aasha proved to be a fantastic learning experience well beyond any classroom experience. We collectively have learned so much as a team about standard practices utilized in the field of engineering. In terms of hardware, we learned a lot about depth cameras and their data. We learned a lot about integrating hardware into the ROS environment. This allowed us to keep our software very modular.   In terms of software, we learned how to develop the software from the systems perspective and spent much time on design the block diagram of the software. We focused on what packages and nodes needed to be developed and how all the messages would be subscribed to and published by the various nodes.

There is much more to developing a robot than hardware and software. The essential item is team building and making sure that each team member feels like a contributing member and is always supported. This in our opinion, is of the most significant value. We want our team to grow and gain knowledge and skills well beyond the competition. Our team management approach was much improved. It resulted in a much more cohesive team where everyone contributed to a common goal. We used standard industry management practices, and it has been a much-improved team.   The main thing we learned is that teams are not just building hardware and software but of building relationships.

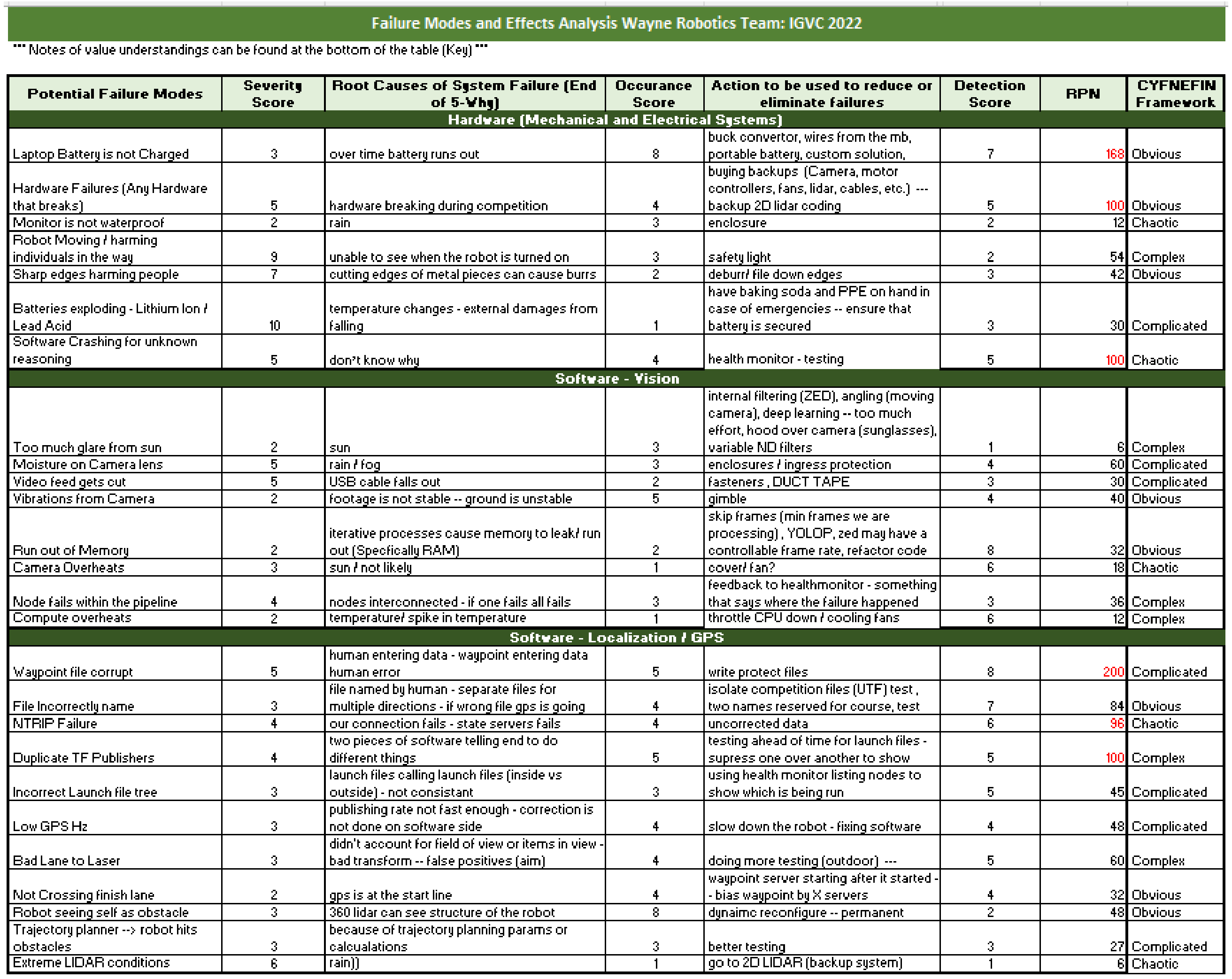
We hope to keep on further developing our team, knowledge base and skillsets in our respective areas and come next year with a more innovative and improved version of the robot. In all we are grateful to have had the opportunity to work on such an ambitious undertaking.

9. Acknowledgments

The creation and funding of this group could not have been possible without the expertise of advisors Dr. Marco Brocanelli, Dr. Azad Ghaffari, and Dr. Abhilash Pandya. They have contributed an invaluable wealth of knowledge and expertise to our club. We would also like to thank Dean Sondra Auerbach for helping us facilitate the necessary resources that helped keep the club going. A debt of gratitude is greatly owed to all the faculty and members of Wayne State University's College of Engineering, who have continuously helped to support us in our endeavors. Finally, we would like to thank the dean of the college of engineering, Dr. Farshad Fotouhi, for facilitating the approval and support of the organization and giving us an amazing space and resources to work. Without him, none of this would be possible.

Appendix

Appendix A: Complete Failure Modes and Effects Analysis with CYNEFIN



Failure Modes and Effects Analysis Wayne Robotics Team: IGVC 2022

""" Notes of value understandings can be found at the bottom of the table (Key) """

Potential Failure Modes	Severity Score	Root Causes of System Failure (End of 5-Why)	Occurance Score	Action to be used to reduce or eliminate failures	Detection Score	RPN	CYFNEFIN Framework
Hardware (Mechanical and Electrical Systems)							
Laptop Battery is not Charged	3	over time battery runs out	8	buck convertor, wires from the mb, portable battery, custom solution,	7	168	Obvious
Hardware Failures (Any Hardware that breaks)	5	hardware breaking during competition	4	buying backups (Camera, motor controllers, fans, lidar, cables, etc.) --- backup 2D lidar coding	5	100	Obvious
Monitor is not waterproof	2	rain	3	enclosure	2	12	Chaotic
Robot Moving / harming individuals in the way	9	unable to see when the robot is turned on	3	safety light	2	54	Complex
Sharp edges harming people	7	cutting edges of metal pieces can cause burrs	2	deburr/ file down edges	3	42	Obvious
Batteries exploding - Lithium Ion / Lead Acid	10	temperature changes - external damages from falling	1	have baking soda and PPE on hand in case of emergencies -- ensure that battery is secured	3	30	Complicated
Software Crashing for unknown reasoning	5	don't know why	4	health monitor - testing	5	100	Chaotic
Software - Vision							
Too much glare from sun	2	sun	3	internal filtering (ZED), angling (moving camera), deep learning -- too much effort, hood over camera (sunglasses), variable ND filters	1	6	Complex
Moisture on Camera lens	5	rain / fog	3	enclosures / ingress protection	4	60	Complicated
Video feed gets cut	5	USB cable falls out	2	fasteners , DUCT TAPE	3	30	Complicated
Vibrations from Camera	2	footage is not stable -- ground is unstable	5	gimble	4	40	Obvious
Run out of Memory	2	iterative processes cause memory to leak/ run out (Specfically RAM)	2	skip frames (min frames we are processing) , YOLOP, zed may have a controllable frame rate, refactor code	8	32	Obvious
Camera Overheats	3	sun / not likely	1	cover/ fan?	6	18	Chaotic
Node fails within the pipeline	4	nodes interconnected - if one fails all fails	3	feedback to healthmonitor - something that says where the failure happened	3	36	Complex
Compute overheats	2	temperature/ spike in temperature	1	throttle CPU down / cooling fans	6	12	Complex
Software - Localization / GPS							
Waypoint file corrupt	5	human entering data - waypoint entering data human error	5	write protect files	8	200	Complicated
File Incorrectly name	3	file named by human - separate files for multiple directions - if wrong file gps is going	4	isolate competition files (UTF) test , two names reserved for course, test	7	84	Obvious
NTRIP Failure	4	our connection fails - state servers fails	4	uncorrected data	6	96	Chaotic
Duplicate TF Publishers	4	two pieces of software telling end to do different things	5	testing ahead of time for launch files - supress one over another to show	5	100	Complex
Incorrect Launch file tree	3	launch files calling launch files (inside vs outside) - not consistant	3	using health monitor listing nodes to show which is being run	5	45	Complicated
Low GPS Hz	3	publishing rate not fast enough - correction is not done on software side	4	slow down the robot - fixing software	4	48	Complicated
Bad Lane to Laser	3	didn't account for field of view or items in view - bad transform -- false positives (aim)	4	doing more testing (outdoor) ---	5	60	Complex
Not Crossing finish lane	2	gps is at the start line	4	waypoint server starting after it started - - bias waypoint by X servers	4	32	Obvious
Robot seeing self as obstacle	3	360 lidar can see structure of the robot	8	dynaimc reconfigure -- permanent	2	48	Obvious
Trajectory planner --> robot hits obstacles	3	because of trajectory planning params or calcualations	3	better testing	3	27	Complicated
Extreme LIDAR conditions	6	rain))	1	go to 2D LIDAR (backup system)	1	6	Chaotic

Appendix B: Power Requirement Calculations

Main Battery (Drive + Auxiliaries):

340 watts (9.43 amps) typical required

Drive:

Each motor 7 Amp max, 3 amp typical.

Control board consumes 1-2 amps

Combined max 15 amps, 7 amps typical

12V auxiliaries:

Velodyne 8 watts

Cooling fans 4x 2.4w each max = 10w total

Beacon 30 watts @12v

Monitor 30 watts @12

Total 78 watts/.9 converter efficiency = 87 watts = 2.4 amps @ 36v

5V Auxiliaries:

GPS .5 watt

Arduino, relays combined .5 watt

Total 1 watt / . 9 converter efficiency = 1.1 watts = .03amps @ 36v

Total:

7 + 2.4 + .03 = 9.43 typical amps required.

Limiting discharge of our 20 AH main battery to 50%, we can expect 10AH/9.43A = 64 minutes of runtime per charge.

Laptop Power Requirements:

Max consumption 230 watts. 160 watts typical.

Battery duration 2.65 hours typical. 1.8 hours minimum.

Internal battery 60 watt hour

Auxiliary battery = 36v*20AH = 720 WH

Max discharge 50%: 720 * .5 = 360 WH

360 + 60 = 420 total watt hours available

420WH/230W = 1.8 hours

420WH/160W = 2.65 hours