University of Toronto

UTRA Autonomous Rover Team (ART): Espresso

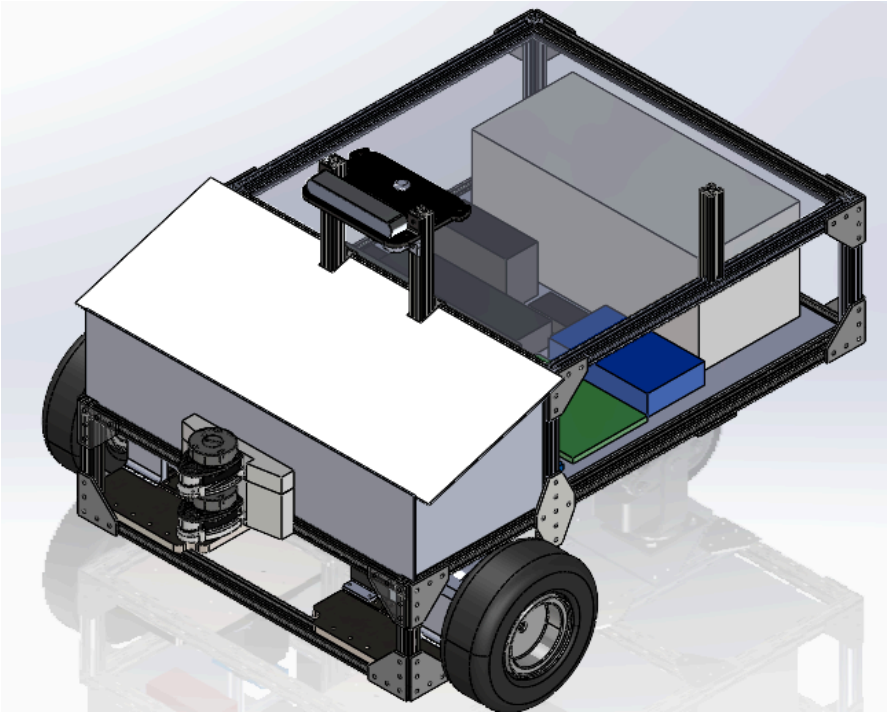

Figure 1: Cad model of Espresso

May 15, 2024

ART IGVC Competition Team

Name	Email	Name	Email
Ghamr Saeed	ghamr.saeed@mail.utoronto.ca	Tracy Sun	tracy.sun@mail.utoronto.ca
Tiger Luo	tigerluo03@gmail.com	Aditya Raval	aditya.raval@mail.utoronto.ca
Leon Lee	leonl.lee@mail.utoronto.ca	Ivy Tan	tianqin.tan@mail.utoronto.ca
Sujit Peramanu	sujit.peramanu@mail.utoronto.ca	Yifei Zhou	fei.zhou@mail.utoronto.ca
Gaurav Gaur	gaurav.gaur@mail.utoronto.ca	Eshan Sankar	eshan.sankar@mail.utoronto.ca
Grace Huan Liu	gracehuan.liu@mail.utoronto.ca	Preet Mistry	preet.mistry@mail.utoronto.ca
Danelle D'Souza	danelle.dsouza@mail.utoronto.ca		

I, Professor Colic, certify that the design and engineering of the vehicle (Espresso) by the current student team has been significant and equivalent to what might be awarded credit in a senior design course.

Professor's Signature: _____

Prof. Sinisa Colic
Assistant Professor, Teaching Stream, Mechanical Engineering colic@mie.utoronto.ca
Tel: 416-978-5435

1. Conduct of Design Process, Team Identification and Team Organization

1.1. Introduction

The University of Toronto Robotics Association's (UTRA) Autonomous Rover Team (ART) was founded in 2008 by a group of students with a passion for robotics. The team has since grown to fill the shoes of an introductory robotics-designed team at the University of Toronto. Our team is relatively large, we started the year with around 80 members and currently have upward of 30 active members on a weekly basis. Our team's focus is to give students a platform to learn robotics concepts while aiming for a completed project.

1.2. Organization

Team Leadership			
Ghamr Saeed	Project Manager	**Ivy Tan**	ROS Lead
Leon Lee	Project Manager	**Jerry Hu**	CV Lead
Yifei Zhou	Embedded Lead	**Jennifer Zhang**	CV Lead
Tiger Luo	Embedded Lead	**Alvina Yang**	Marketing Director
Karanjit Gandhi	Mechanical Lead	**Eshan Sankar**	Marketing Director
Giacomo Cristiano	Mechanical Lead		

ART is divided into 4 subteams, Mechanical, Embedded / Electrical, Computer Vision (CV), and Robot Operating System (ROS). Team leads are responsible for onboarding general members, leading work sessions for their respective subteam, and ensuring that tasks are completed adequately and in a timely manner.

1.3.Design Assumptions and Design Process

Members were split into small groups within their subteam, each working on a specialized task until that task is finished at which point they are reassigned to different groups. During the fall, the team was doing research and design work mostly, and during the winter, the team implemented the designs they were working on.

Some design considerations and assumptions our team employs:

- Our already tight budget has been reduced even more this year. Despite this, we opted to create an entirely new chassis. We also bought a highly capable computer to overcome hardware limitations. While this has allowed us to tackle a more challenging and interesting design process and boost the capability of our models, it has also been fairly difficult to balance the demand of this year's plans with the budget.
- We are limited in members that are licensed to use our school's machine shop, and even more limited in availability to use it. As such, designs are influenced heavily by a small subset of tools all members can use. We avoid lathes, mills, CNCs, and welding wherever possible.
- Our team is entirely extra-curricular. We lose a lot of members during midterm and final exam season, and have further member loss after the school year ends.
- We assumed that the competition course, requirements, and rules have remained majorly unchanged.
- The main purpose of our team is education, so we elected to employ a large number of students knowing well that many of them are in their first year of university and have done no robotics or programming before. Hence, a lot of onboarding is necessary.

Our design decisions process starts with identifying an issue, then the relevant subteams will produce candidate solutions that are brought to a leads' meeting. At this meeting, candidate solutions are proposed and other subteam leads are able to comment on them, specifically thinking how this design will affect

each individual subteam. If a candidate solution appears to satisfy every lead's constraints, a team member is assigned to the design and a prototype is made. In hardware systems this typically means a CAD model, in software, a simulation. If the design appears to work, the member in charge follows through while working with leads to implement a final version of the design.

2. System Architecture of the Vehicle

2.1. Significant Mechanical, Power, and Electronic Components

- Electronic: Laptop, Raspberry Pi, LIDAR, stereo camera, 300W motors, and motor controllers. ROS runs on the laptop, communicating with the Raspberry Pi over SSH to control the motors. LIDAR and the stereo camera connect to the laptop via USB.
- Power: LiFePO4 batteries and 24V regulators.
- Mechanical: K353a tires, swivel caster, and 6061 aluminum extrusions (T-slots) for the frame.

2.2. Safety Devices

Manual safety devices involve manual and remote shutoff power switches, with remote shut off possessing a range of greater than 20m, and manual shutoff placed 2ft above the ground. Automatic safety devices include fuses, heat sensors, current sensors, and a ROS shutoff that automatically turn off power when abnormality occurs without the need for personal intervention. The ROS shutoff takes into consideration sensors measuring current and voltage at the motor controllers, sensors measuring temperature within the rover enclosure, and the state of the ROS stack. If any unusual or extreme conditions are sensed (ie, overheating), or error messages are received, an automatic shutdown is initiated on the drive circuit, disengaging motors, causing it to come to a smooth stop. Once the issue is resolved, the system can be restarted.

2.3. Significant Software Modules

We leverage Robot Operating System (ROS) to facilitate sensor communication and rover behavior. Significant software components include using the robot_localization package to filter odometry sensor data from the Hall Effect Sensor, IMU and GPS. Obstacle Detection relies on a 2D LiDAR for physical obstacles (i.e. barrels), while the lanes are tracked using a ZED stereo camera combined with classical and deep learning algorithms. Mapping is handled using the SLAM package 'cartographer', which takes in obstacle and odometry data and outputs a map and transform frames. Finally, our navigation is handled by the Navigation Stack, which uses the move_base package to interface with the pathplanning package navfn and generate and trajectories and associated movement commands.

The major tasks of computer vision perception are lane and pothole detection, which are difficult to detect using other sensors. We still provide alternative computer vision algorithms to detect other obstacles like barrels, but it is not our major focus. To encounter various situations and unexpected failures, we developed both classical and deep learning methods for lane and pothole detection. The classical approach mainly utilized feature detection and image filtering techniques, which requires less computing power compared to deep learning methods. As for the deep learning approach, we mainly took use of the state-of-the-art object detection model, YOLOv8, which turns out to be a robust, accurate, and relatively light weight neural network model.

2.4. How these Items are Connected and Interact in the Vehicle

ROS receives sensor data which is provided to the laptop either through direct usb communication with the sensors or through the Raspberry Pi. In turn, the Pi is connected to the motor controllers through Arduino microcontrollers. Embedded takes a twist command from ROS and sends it to the motor controllers

through the Raspberry Pi, CV provides simulated obstacles that are added to the ROS cost map, both ROS and CV pipelines are handled by the main computer. Sensors are placed strategically such that they perform as well as they can (e.g. lidar is placed at the tip of the rover, IMU at the top to avoid the magnetic fields created by the rover, etc.) and some are mounted on with custom 3d printed mounts. Circuits have been designed and implemented to preserve the integrity of any communication that does not happen directly through usb by means of electrical isolation and filtering. The chassis is designed to house components securely using velcro. The chassis provides easy access to all components while providing a significant amount of waterproofing. During testing, assembly files of the designed chassis were used to simulate the performance of the Rover more accurately in Gazebo.

3. Effective Innovation in Vehicle Design

3.1. Detection of Ramp using Two LiDAR Units

Ramp detection was done via two LiDAR units to detect a difference in distance from the robot's point of view. The expectation is that a ramp is the only obstacle that will generate a large enough distance difference to be detected by our software, given that the LiDARs are at two different heights. Thus, the cost of an expensive 3D LiDAR unit can be avoided by using two cheaper 2D units. This is described in greater detail in section 6.2.4.

3.2. Ramp Navigation using Series of Waypoints

We developed an operating mode to traverse the ramp in a controlled manner. When the ramp is within visible range of the rover, the regular navigation is interrupted and enters ramp navigation. We ensure the rover moves straight by setting a series of waypoints spaced along the ramp, thus limiting the undesired path variability (i.e. big turns) that could force the rover to fall off. Discussed in more detail in section 6.6.

3.3. Galvanic Isolation

Through extensive testing, the motor controllers were found to produce unstable feedback and have significant voltage rail shifting due to poorly designed internals. We developed a method of galvanic isolation using optocouplers. to minimize the impact this has on the rest of the electronics suite. The rover was designed such that the only points of contact between the motor controller circuit and the control circuit are digital signals. To galvanically isolate the two circuits, optocouplers are used to safely and quickly transport these digital signals, while also heavily reducing noise.

3.4. Fast and robust classical lane detection method

From previous competition experience, we found that the performance of a deep learning model can be heavily impacted when running in parallel with other tasks. In last year's competition, the inference speed dropped by a lot compared to the testing we did. Therefore, we developed a backup algorithm that utilizes no deep learning module and only depends on traditional CV algorithms like feature detection and image filtering.

4. Mechanical Design

4.1. Overview

Espresso was created to address multiple integral design flaws that our previous rover, Caffeine, possessed. The chassis had some of the major ones such as difficult access to components, bending in the middle under the weight of the batteries, being one solid unit that cannot be disassembled, and

relatively aggressive forward tilt of the chassis which caused the lidar and camera to point towards the ground, and bad waterproofing that made it difficult to access the circuit.

The new rover needed to be stronger so as to not bend at the weight of the components or the added competition weight, it needed to provide better access to serviceable components, and needed to sit flat with no tilt. All while following the weight and size requirements.

4.2. Description of Significant Mechanical Components

4.2.1. Chassis

The chassis consists of a 3 ft by 2ft ladder frame with an upper subassembly that is designed to house the payload, and all electronics excluding sensors (i.e LIDARS & stereo camera) and safety equipment (i.e safety lights, & e-stop). The lower sub assembly comprises the drive end.

4.2.2. Drive system

The rover relies on a differential drive system conforming to a unicycle model. The system consists of 2 larger front wheels, each driven by a motor, and a third smaller back caster wheel.

4.3. Descriptions on Frame Structure, Housing, and Structure Design

4.3.1. Frame Structure

To improve on the previous design, we decided that the chassis should be a box rather than a sheet of metal. This would give us the ability to store our components while also being able to isolate them from weather effects. The chassis is made with 6061 T-slots that are connected together in a “diagonal figure 8” shape (Figure 2). This shape choice balances the larger size of the motorized wheels with the smaller size of the caster wheels to keep the rover as level as possible. To account for buckling under heavy weight, we added 1 support on each side of the chassis that diagonally connects the lower and upper boxes. We performed ANSYS static structural load simulations (Figure 3) that verified the frame’s capability to handle weights much larger than needed without bending.

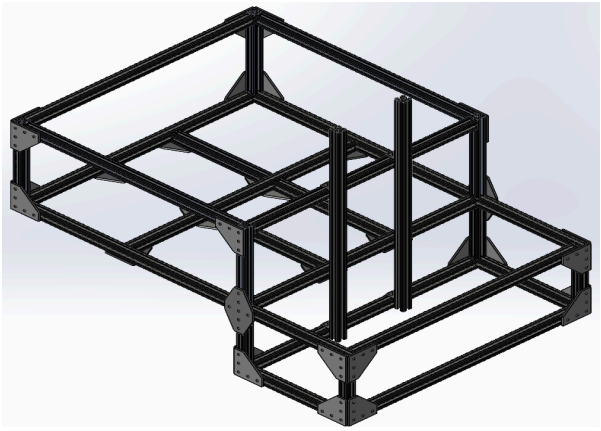

Figure 2: CAD model of the frame as it would be assembled in real life.

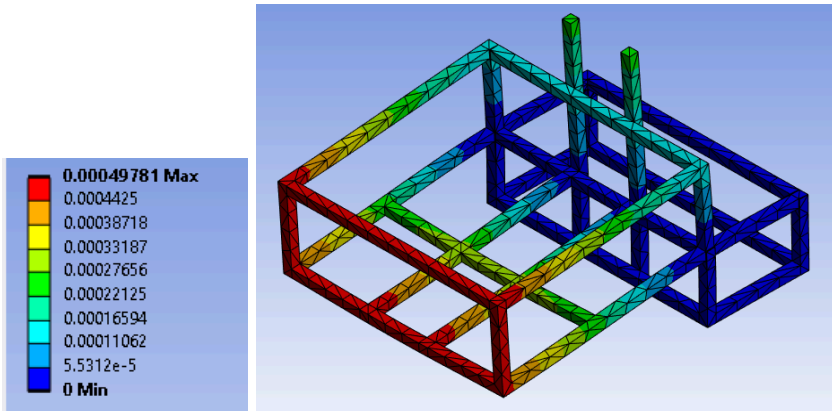

Figure 3 : Ansys static load simulation on the body of the rover.

4.3.2.Housing

To house components, we decided to go with plywood, acrylic, and 3D printed PETG as our materials of choice. Acrylic covers the outer layers of the chassis with the bottom layer of the upper box is made of plywood. To house the motors, we machined 2 plates of ⅜" 6061 aluminum and high infill 3D printed PETG motor mounts. Brass heat set inserts were leveraged to allow reliable assembly and disassembly of the mounts. The aluminum plates are mounted to the bottom box of the frame with spacers in between to make sure that the rover is level. Since the laptop requires very frequent access, it is housed in its own unit attached to the acrylic plate on the upper box of the rover.

4.4.Drive System

Although research was put into the drive system this year, we decided to mostly inherit it from the previous rover since it worked well thus far, though we moved the caster wheels to the back instead of the front. The motors are ATO 300 W brushless DC motors. They come equipped with a 16:1 planetary gear system. This allows for a maximum speed output of 188 rpm and a maximum torque of 14.4 Nm. Our drive system is comprised of the motors, which are connected to 2 front wheels, and the back caster wheel (Figure 4).

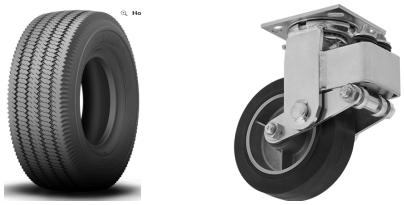

Figure 4: Pictures of the front driven wheels (on the left) and caster wheel (on the right)

Drive Wheel Specifications:

SIZE	PLY RATING	RIM WIDTH (IN.)	O.D. (IN)	S.W. (IN.)	MAX LOAD (LB)	PSI
4.10/3.50-5	4PR	5x3	11.8	3.9	395	50

4.4 Suspension

The decision was made to eliminate external suspension entirely, opting instead to use the intrinsic suspension from the wheels of the new rover. In the context of the ramp and rough terrain and based on the specs of the wheel, reducing the internal pressure (psi) by roughly 10% from its standard pressure.

4.5 Weatherproofing

Weatherproofing involves shielding the chassis and its internal components from possible rainfall. A large acrylic sheet was cut into rectangular panels that were then fitted in the grooves of the T-slots. 3D printed spacers were incorporated to ease access into the channels for installation purposes.

4.6 Electrical Integration

Highly visible trailer lights at the top of the tower act as the safety lights for Espresso. These lights were selected to comply with competition rules turning on when the rover is in teleoperation mode and flashing in autonomous mode. Printed PLA mounts for the lidar, camera, and GPS were used for their cost. We attach our electrical components to the wooden layer using velcro which guarantees the stability and modularity of components.

5. Electronics and Power Design

5.1. Overview

A high-level overview of the electronics systems and components is shown below in (*Figure 5*).

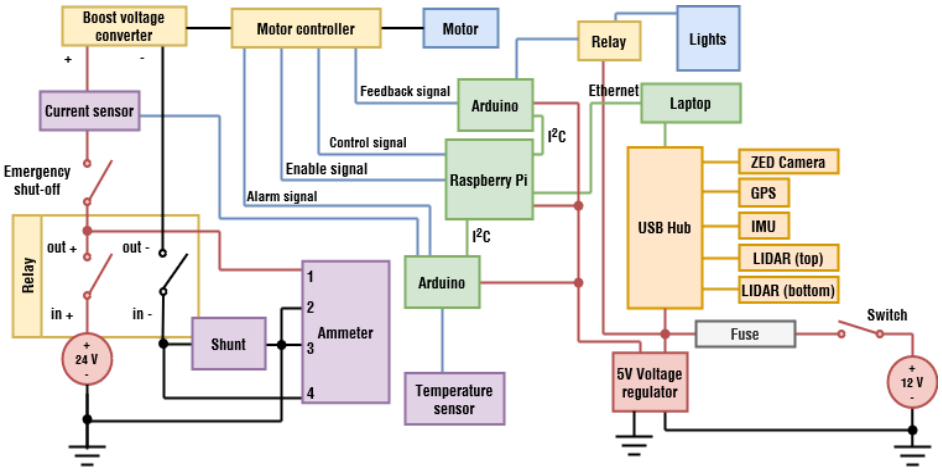

Figure 5: High-level overview of the electronics systems. There are two sets of motors, but only one is displayed for brevity.

The new Espresso electronic suite now consists of two circuits, the first is a signal/sense circuit which consists of the laptop, microcontrollers, sensors, and a 12V battery. The second is a motor drive circuit consisting of the motor controllers, 24V batteries, and many safety/regulating features. The primary improvement over last year is the implementation of a star-style grounding net, a complete galvanic isolation between the two circuits, and a soft start switch, allowing for much better stability of power rails and signal integrity.

5.2. Description of the significant power and electronic components

Espresso still does a majority of its computation and heavy sensor data processing via the laptop and its USB-connected sensors. The components are listed as below:

Laptop	i7-12650H, 16GB DDR5, RTX3070
Lidar (x2)	RPLidar A1
IMU	PhidgetSpatial 3/3/3 Precision 1044_1

Stereo Camera	Zed Stereo Camera
GNSS	Columbus P-7 Pro

The power distribution system consists of components used to switch on and off the power (with soft start), regulate voltage, and provide safe emergency stoppage.

Soft Start Switch	IRF9540, PC817
Batteries (x2)	12V 8AH 8 Ampacity LiFePO4
Voltage Regulator	Boost converter, 24V 1500W
Remote Relay	30A antenna remote switch
Mechanical E-Stop	10A push button
Motor Controllers	ATO-BLD750

The signal circuit interacts directly with several low level sensors meant to assess the current safety state of the rover and communicate its assessment with ROS.

Microcomputer	Raspberry pi 3 model B+
Microcontrollers	Arduino Mega (x2), ESP-32
Temperature Sensor	LM60CIZ
Current Sensor	ACS712 Hall Effect sensor
Optocouplers (x6)	PC817

5.3.Power distribution system capacity, max. run time, recharge time, safety

The 300W motors are controlled using ATO-BLD750 controllers, powered and regulated to 24V by two 12V lithium batteries. These batteries are 8AH with a maximum current draw and intake of 8A. The batteries thus have a max run time of 8 to 1 hour, depending on speed limitations, and have a recharge time of 1.6 hours. The primary consideration of the motor drive circuit (*Figure 6)* is the limiting of a large inrush current during startup, which causes the Battery Management System (BMS) that is internal to the batteries to shut them off. This is mitigated through the usage of a soft start switch, which slows the inrush enough for the voltage rails to stabilize, and the BMS to not fail.

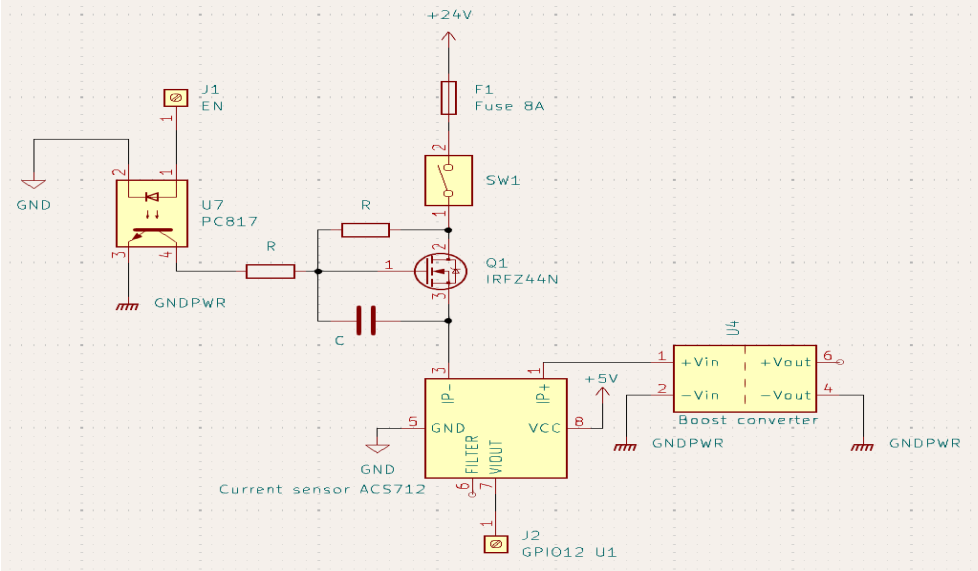

Figure 6: Motor drive circuit

5.4. Electronics suite, including computers, sensors, motor controllers

The motors contain Hall-effect sensors that generate a square-wave signal at a frequency roughly proportional to the rotational speed of the motors. These feedback signals are used to estimate the rover's velocity and displacement. However, the signals are very delicate and prone to noise, so we use optocouplers and an RC filter to galvanically isolate and denoise the signals.

The signal circuit includes the laptop, a Raspberry Pi, an ESP32, and two Arduinos. The laptop handles a majority of ROS computation and navigation, and is connected to the rest of the circuit through the raspberry pi, communicating through SSH on the ROS URI. The Raspberry Pi is the main point of contact between the ROS localization and navigation stack and any physical hardware, communicating with the ESP32 through the MicroROS serial client and with ROS through the network. It is also responsible for sending motor control signals, emergency shutoff signals, and startup signals. The Arduinos are dedicated to reading the motor feedback signals and send the processed information to the ESP32 through SPI communication. The ESP32 serves as a midpoint to send ROS messages, while also reading signals from the current, temperature, and voltage sensors.

5.5. Mechanical and wireless ESTOP systems

The manual mechanical switch is a latching large red button that matches competition constraints. It is a high side power switch connecting the batteries to the voltage regulators, which upon pressing cuts off power and causes the motors to come to a complete stop. The wireless ESTOP is a remote relay that is identical in function and is in series to the manual switch, however it uses an antenna to allow for emergency shutoff from over 20m away.

6. Software Strategy and Mapping Technique

6.1. Overview

Our software team is broken into two distinct parts: CV and ROS. CV processes camera data to generate a representation for lane markings and potholes. ROS uses sensor data to localize, map, and navigate the environment.

In general, we use the Hall Effect Sensor, IMU and GPS to compute odometry, as well as a 2D LiDAR and ZED stereo camera for obstacle detection. These two sources of information are used to generate a single map using a SLAM package, combining the available environment data. The path planning is coordinated with the Navigation Stack using the move_base package, which uses the map to compute the shortest path to the goal while avoiding obstacles. This periodically outputs a TwistStamped message that is sent to the electrical integration modules for actuation.

This for a modular system resembling a microservices architecture, with each module (ie. odometry, navigation, lane detection) capable of being developed without changing other modules (Figure 7).

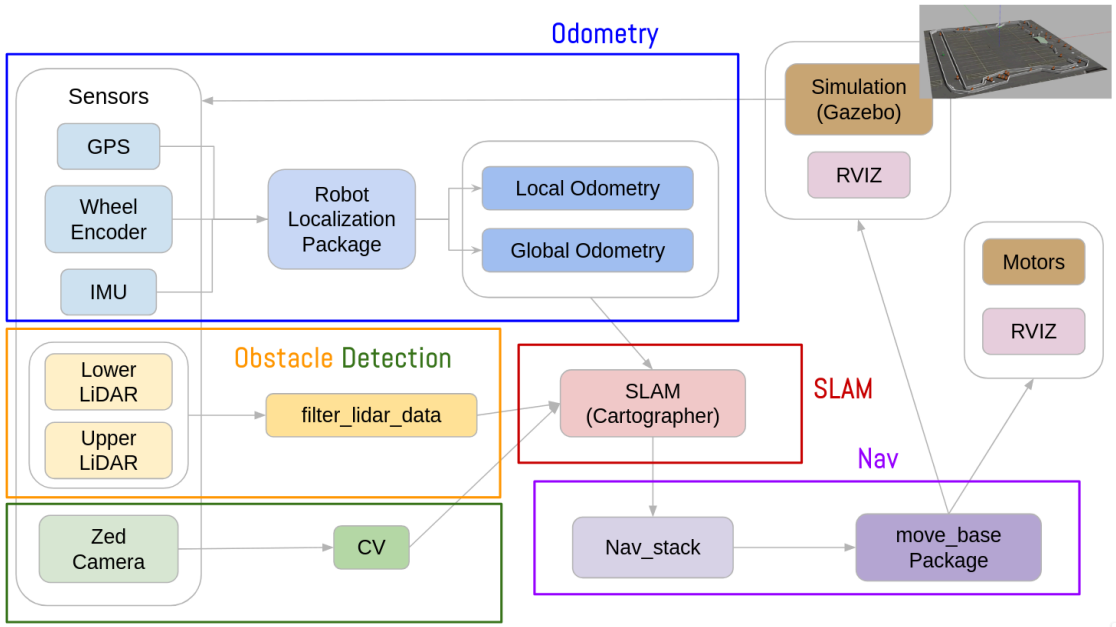

Figure 7: General Software Pipeline (Electrical Integration Ommited)

6.2. Obstacle Detection and Avoidance

6.2.1. Lane Marking Detection

To encounter various situations and also have backup options, we implemented both classical and deep learning approaches for lane detection.

Our classical approach (Figure 8) mainly relied on feature detection and filtering techniques. First, the image is converted to grayscale and gaussian blurred. Then, Canny edge detector is used to extract the lane mark edges. The obtained edge map is further filtered by only picking the white contour to get an accurate lane detection mask. We only run detection up to 70% of the frame height to reduce noise. We process 8 frames at once as they are very similar. As a result, we reached a 25ms inference speed and near perfect detection, therefore a fast, robust, and accurate algorithm.

Our deep learning approach (Figure 9) mainly utilizes the YOLOv8 object detection model. A dataset of 1174 pictures from last year's competition track is manually labeled in the segmentation mask format. The results are more accurate near the camera than further away. To further increase the inference speed, we tried parameter pruning. Testing showed that pruning 5% of the parameters gives an inference speed of 9ms with roughly the same accuracy.

Figure 8: Lane detection result from our classical approach (blue lines)

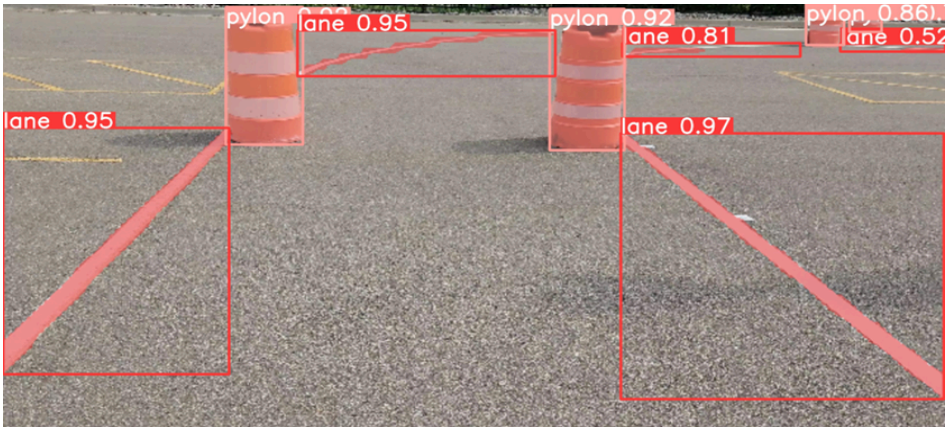

Figure 9: Lane and barrel detection result from YOLOv8

6.2.2. Pothole Detection

We tackle pothole detection (Figure 10) using the same algorithms described in 6.2.1.

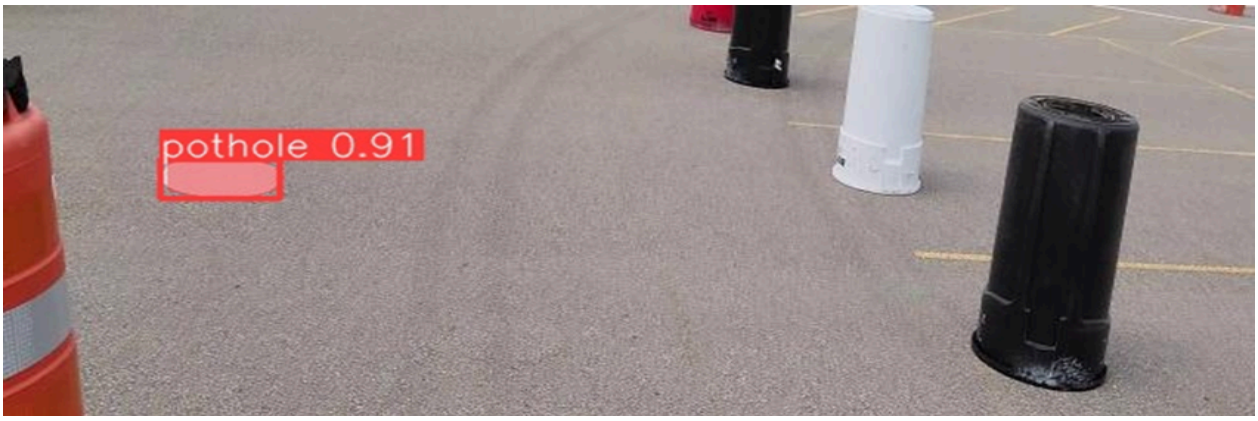

Figure 10: Pothole detection result from YOLOv8

6.2.3. CV Integration

Since our detection results in the format of a segmentation mask, the integration of all our detection is pretty straightforward. All we have to do is label the lanes and potholes using different numbers and bitwise OR all the detection masks and we have a map containing all detections. Then we would utilize the depth reading from the ZED camera and image_geometry library to convert the 2d map into a 3d map.

6.2.4. Physical Obstacle Detection

To detect physical obstacles like barrels and barriers, we rely on a Slamtec RPLIDAR A1M8 for its high accuracy and speed. We considered using 3D lidars, but they are beyond our budget, and computer vision is less accurate. Due to the relative sparsity of the data, it was also computationally feasible for us to avoid filtering the point cloud, so we directly add points detected by our lidar to the map. This assumes that anything the lidar detects is an obstacle. We handle ramp detection separately to avoid recognizing it as an obstacle. This reduced computational expense, as we do not run point cloud classification.

6.3. Map Generation

The map is generated using the SLAM package 'Cartographer' [1], which provides loop-closure that aids in returning the rover to its starting position. Mapping works by successively building submaps (local SLAM) using sensor data and connecting them consistently with background threads for loop closure (global SLAM). Each submap is built using obstacle data obtained from lane/pothole detection (camera) and LiDAR, with new submaps being placed according to odometry information.

To detect the ramp we mount a second LiDAR above the first and measure distance difference resulting from the incline of the ramp and publish a new LaserScan message that removes the ramp from our detection data so the rover does not avoid the ramp. We computed the optimal distance threshold to trigger filtering using the fact that the ramp doesn't exceed a certain incline and trial and error.

6.4. Goal Selection and Path Planning

To set goals, we enter the GPS coordinates into an ordered json file, with heading (North vs South, in case we must complete the course backwards) and laps to complete. Then, we send the transformed pose to a move_base goal. Move_base handles the path generation based on the costmap_2d, which reads directly from the Cartographer map (Figure 11). Afterwards, we query our GPS to see if we reached the point.

For path planning, we rely on 4 key packages, move_base [2], costmap_2d [3], navfn and base_local_planner [4]. Move_base serves as our high-level interface, and is used to coordinate the costmap and path planning algorithm into movement commands. The costmap_2d package, reading from the SLAM map, creates a cost-based representation. The packages navfn and base_local_planner use this costmap to compute a global and local path that avoids obstacles (higher cost). As new obstacles are detected, they are added to the SLAM map, which updates the costmap and subsequently the path plan.

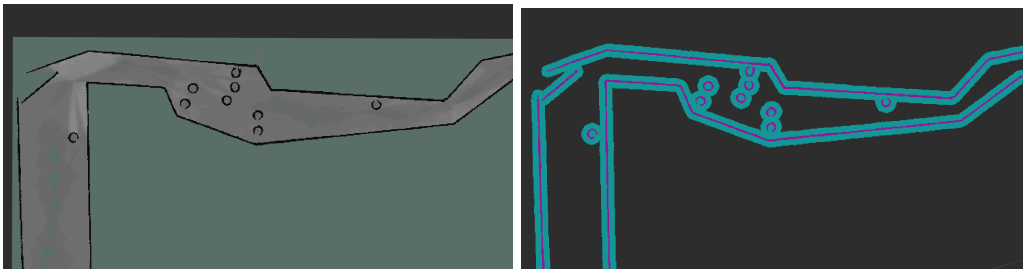

Figure 11: Map generated using Cartographer (left) vs costmap generated by costmap_2d (right)

6.5. Odometry Sensors

To track the robot's current position, odometry is generated by fusing data from the Hall sensors, IMU, GPS data and/or visual odometry using the robot_localization package[5]. Visual odometry is generated using a stereo odometry node from the rtabmap_odom package, using ZED camera data [6]. The ekf nodes implement extended Kalman filter state estimation to output global and local odometry[5]. Prior to fusing GPS data, it is entered into the navsat_transform_node with an initial orientation and heading given

by the IMU to generate a transform from UTM (Universal Transverse Mercator) coordinates into cartesian [5].

6.6. Ramp Navigation Mode

With the combined effect of the sloped surface and the strategy of removing ramp detection from the LiDAR data, the ramp has become difficult to traverse without finer control. Realizing that relying on the given waypoints would not be enough to ensure the planned trajectory results in a straight line across the ramp (and not dangerously near the edge where it could fall off), we decided to implement a ramp navigation mode that interrupts the regular path planning to safely get across the ramp.

The dual LiDAR system described in section 6.3 is used to detect proximity to a ramp and trigger an interrupt. The navigation program then plans a waypoint at the center and perpendicular to the ramp entrance. Once the rover is in place, a series of waypoints at 0.5-meter increments are placed along the length of the ramp, guiding the rover to traverse the ramp without veering off. Once the ramp is completed, the navigation program returns to the original list of GPS waypoints and continues on the course.

This strategy was developed to address the following problems:

- Simply sending a ‘go straight’ command once at the ramp introduces a risk of running off if the rover becomes the slightest bit crooked
- Entering the ramp at an angle or close to the edge may result in rotational compensation on the ramp that may overshoot since the wheels are at different heights
- The small increments constrain the path trajectory to a straight line and path deviations are less likely to occur

7. Analysis of Complete Vehicle

7.1. Lessons Learned

Hardware teams learned to always design for stability and robustness, rather than complexity and fancy capabilities. While software is prone to failure from errors in logic or bugs, hardware is always more likely to fail due to random effects from the physical environment that often can’t be predicted. As such, it is much better to design for worst case scenarios and never assume known conditions or situations.

On the CV side of the software, extensive research should be done prior to starting to work. In last year’s competition, we didn’t do much research on the classical detection approaches and therefore developed a relatively less efficient classical algorithm compared to this year. At the beginning stage of development, we looked into various papers that researchers have proposed and successfully developed our classical CV algorithm this year.

7.2. Hardware Failures

Failures include odometry systems failing due to instability in the voltage rails. This would cause a large amount of sensor data to be rendered invalid, and make navigation very difficult. Numerous steps have been taken to ensure this doesn’t happen, but in testing it, the odometry systems will still occasionally randomly fail despite the same starting conditions. There is also a speed limitation due to the battery’s low ampacity, which heavily limits our safe speed and will likely cost a lot of time when completing the course.

7.3. Software Testing, Tracking, and Version Control

Testing primarily uses our simulated world to gauge performance both qualitatively and quantitatively. For mapping, we specifically looked at how long it took to generate a map, its accuracy compared to the gazebo world, and how well the obstacle data in the SLAM map transferred to the costmap. We also

assess the quality of the lane transformation by visualizing in RVIZ and comparing the gazebo world for lane width, straightness, and distance from the rover.

To test navigation, we mainly changed the spawn point of our robot and tested to see if we could complete a particular task. For example, to test no man's land navigation, we changed the spawn point to be at the first waypoint, then we tried seeing if we could reach the last waypoint using our GPS navigation pipeline. For navigation, as we didn't have CV integrated for the majority of our testing, we relied on having physical walls in the place of the lanes. This lets us test our navigation performance independent of CV's performance. We tested consistency by running the auto-navigation repeatedly to identify problem spots that stalled or resulted in failure.

To assess the accuracy of the local, global, and visual odometry, a ground truth plugin (p3d_base_controller) was added to the Gazebo simulation [7]. This plugin gives the current coordinates of the robot in the world frame and was then transformed into the same frame that the robot odometry uses. Acting as a reliable odometry reading, we could compare the odometry generated by some combination of sensors (Hall effect, IMU, GPS) to the ground truth odometry, with deviation indicating drift and inaccuracies. We could identify which movements each sensor was more accurate in to fine-tune the filtering parameters and reduce overall drift. Bug tracking and version control were done using Git/Github, which served to organize individual issues and the branch/member dedicated to solving them. Pull requests are made once an issue has been evaluated to be sufficiently addressed.

7.4. Virtual Simulation Testing Environment

We leveraged the Gazebo simulation, a standard practice employed by the ROS community. To create an accurate simulation, we first created a URDF (Unified Robot Description Format) model of our robot based on the final mechanical CAD assembly. After this was done, we simulated all of our sensors using Gazebo plugins with Gaussian noise to simulate sensor noise.

For the world, we created a model of the competition field in Gazebo. Our world allows us to test individual features without being blocked on other features being added. For example, we could test our navigation without the deep learning model since we had the walls instead. A good simulation also allowed us to test software updates independently from robot hardware which enabled our mechanical and embedded teams to work simultaneously on the robot.

Our simulation stack simulates all aspects of the auto-nav challenge. Sensor readings, obstacles, inclines, sensor noise, etc. We simulate all aspects of our mapping and navigation stacks.

We used Rviz to visualize the sensor data, lane detections, SLAM map, costmap, and path planning.

7.5. Physical Testing to Date

7.5.1. CV and ROS

The current CV lane detection and ROS integration as well as mapping was tested using 2 stripes of white tape approximately 5 ft apart and 7 ft in length. We stuck the tapes to the asphalt then we pointed the camera at them at different heights and tilts and holistically measured the inference. The results were promising with the lane projections appearing parallel to each other. This test was performed 2 times in different locations and under different lighting conditions. Full tests of the entire pipeline are scheduled for the next two weeks.

7.5.2. Embedded and Electrical

Physical embedded and electrical testing was continuously done throughout the year, each component was tested separately as they were slowly integrated, then the system would be tested with said component. Performance was evaluated and improved as needed. Examples are testing motors and motor

encoders by turning them on and measuring their max speed, testing ROS commands by controlling the rover through teleoperation on the ground, testing hall effect sensor accuracy by comparing its measurements to the frequency measured using an oscilloscope and the speed measured using a laser tachometer, etc. This way, most integral parts of the rover on the electrical and embedded sides were tested thoroughly.

8. Initial Performance Assessments

8.1. Mechanical Verification

The base frame or chassis was tested under standardized simulation conditions using ANSYS, where estimates of the weight of electronics and other components like the safety lights, camera, LiDAR etc., was taken and evenly distributed on the surface. The results helped us assess the maximum stress points and determine the electronics layout. As per the testing, the results verified that the rover chassis is capable of handling higher weights than our requirements without any buckling or bending. Stress testing was also done to confirm the load on the castor wheel and its ceramic mount by placing a standardized load block to investigate any critical areas of failure. The results of the load testing showed that the mount is secure and the castor wheel can function properly under the load of the components.

8.2. Electrical Stress Testing

The batteries have relatively small capacity and ampacity, however they are able to handle fast changes in speed thanks to acceleration limiters and voltage regulators. The batteries are also resistant to brownout, able to run the rover without tripping under voltage protections until reaching 15% capacity. However, during periods of intense load, the internal BMS is at risk of activating and shutting off the rover on its own. As such, it is not recommended to run the rover at speeds above 0.8m/s.

8.3. Lane and Pothole Detection

Our algorithm is mainly tested using videos of the track taken from previous years. The GPU used is Nvidia T4 in Google Colab. All detection accuracy mentioned is measuring the IOU of the detection and ground truth. In terms of our classical detection approach, we reach a near 100% detection accuracy in terms of IOU and an inference speed of 25ms. For our deep learning algorithm, the model reached 91.6% and 96% detection accuracy for lane and pothole respectively. The inference speeds are 12.8ms and 15.7ms for lane and pothole detection. Finally after doing 5% parameter pruning, we improved the inference speed to 9ms inference speed. Lane detection was tested physically to validate the algorithm with promising results as mentioned in section 7.5.1

8.4. Software Testing

As of now, testing of mapping and navigation has primarily been in simulation (mentioned in section 8.3) and have performed satisfactorily. Map generation of obstacles detected using the LiDAR are consistently accurate to the gazebo world, with delays of <1s between initial detection via sensor and its placement on the map. The lanes generated by the CV model are currently still in need of refinement. While the lanes appear on the map in clear lines, they are not perfectly incident to the real lines. We hypothesize that there are either inaccuracies in the depth conversion and will need further testing. The goal-setting and path-planning components of the rover also perform well. The previously described goal setting system in section 6.4 is able to accurately place a goal based on GPS input. The path planning also satisfactorily avoids all obstacles and with time, will reach the goal. The completion time is currently unideal and will be improved in the future. Finally, tests with the ramp navigation mode has successfully allowed the rover to traverse the ramp approximately 85% of the time, with the rover remaining within 10° of the length of the ramp. This comes at the cost of additional time taken and further optimization may be made.

References:

[1] “Cartographer Ros integration,” Cartographer ROS Integration - Cartographer ROS documentation, https://google-cartographer-ros.readthedocs.io/en/latest/.

[2] “move_base,” ros.org, http://wiki.ros.org/move_base [accessed May 15, 2024]

[3] “costmap_2d,” ros.org, http://wiki.ros.org/costmap_2d (accessed May 15, 2024).

[4] "base_planner," ros.org, http://wiki.ros.org/global_planner (accessed May 15, 2024)

[5] T. Moore, "robot_localization 2.7.4 documentation," 2016. [Online]. Available: http://docs.ros.org/en/noetic/api/robot_localization/html/index.html. [Accessed 15 May 2023].

[6] M. Labbe, "rtabmap_odom," Open Robotics, 19 April 2023. [Online]. Available: http://wiki.ros.org/rtabmap_odom#stereo_odometry. [Accessed 15 May 2023].

[7] Open Source Robotics Foundation, "Gazebo plugins in ROS," Open Source Robotics Foundation, 2014. [Online]. Available: https://classic.gazebosim.org/tutorials?tut=ros_gzplugins. [Accessed 15 May 2023].