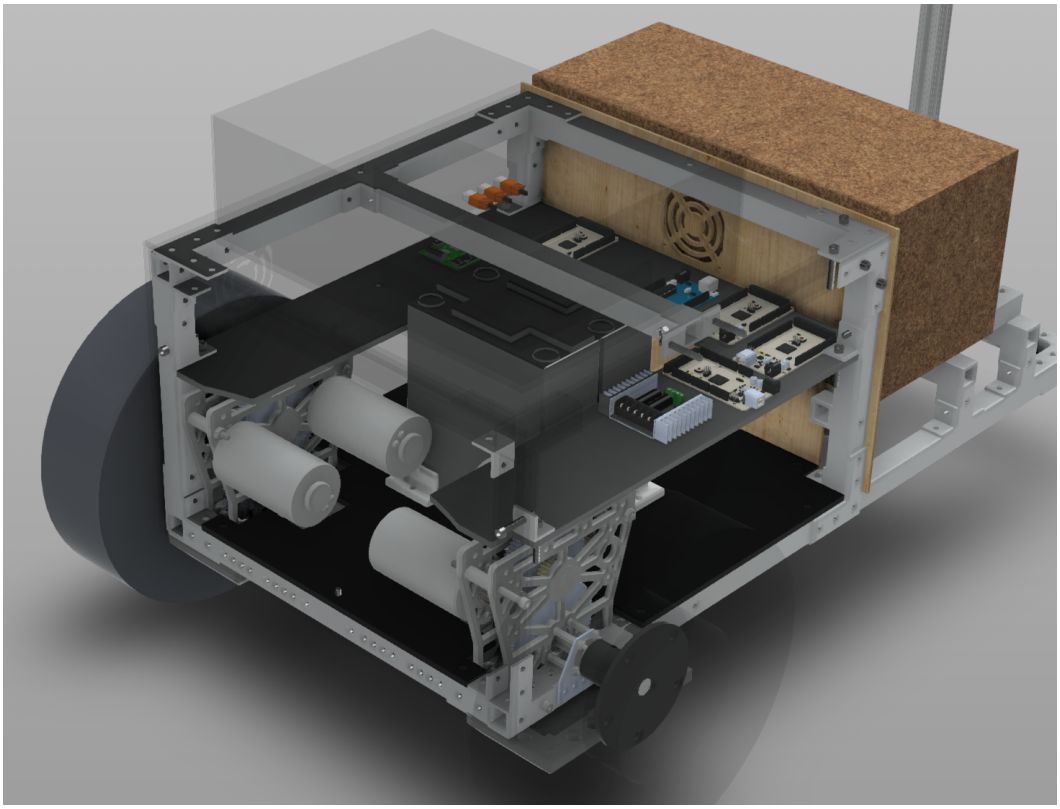
Kohei Design Report

University of Michigan - Ann Arbor



Submitted: May 15, 2022

Team Captain: Hersh Vakharia | hershv@umich.edu

Faculty Advisor: Xiaoxiao Du | xiaodu@umich.edu

Damen Provost | provostd@umich.edu

Statement of Integrity: Provided Separately

Team Roster

Name	Email	Leadership Position
Aakash Bharat	aakashvb@umich.edu	
Aarya Kulshrestha	akulshre@umich.edu	
Alan Teng	thtalan@umich.edu	Business Lead
Albert Hung	azhung@umich.edu	
Alexander Cole	aljcole@umich.edu	
Alison Brei	breia@umich.edu	
Aman Kushwaha	amankush@umich.edu	
Andrew Varghese	andvarg@umich.edu	Computer Vision Lead
Arianna Kerkmaz	akerkmaz@umich.edu	
Ashwin Saxena	ashwinsa@umich.edu	Controls Lead
Benjamin Rossano	brossano@umich.edu	
Benton Edmondson	benton@umich.edu	
Christian Foreman	cjforema@umich.edu	Operations Director
Colin Yoon	colyoon@umich.edu	Sensors Lead
Daniel Chayes	dachayes@umich.edu	
Daniel Fichtl	fichtld@umich.edu	
Daniel Garan	garadan@umich.edu	
David Welch	dswelch@umich.edu	
Dhruv Parmar	dparmar@umich.edu	
Drew Boughton	drbought@umich.edu	
Hersh Vakharia	hershv@umich.edu	Team Lead/President
Himan Yerrakalva	yerrak@umich.edu	
Jahnavi Bhatt	jahnavib@umich.edu	
Jose Diaz	diazjose@umich.edu	Embedded Systems Lead
Kohei Nishiyama	kohein@umich.edu	Platform Lead
Krishna Dihora	kdihora@umich.edu	
Megan Dzbanski	mdzbansk@umich.edu	
Rajiv Bharadwaj	rajivbh@umich.edu	
Ryan Hou	rlhou@umich.edu	
Saksham Sadh	saksham@umich.edu	
Zijun Ning	joshning@umich.edu	
Ziyang Ning	zyning@umich.edu	

Team Overview

Introduction

Kohei, Japanese for “peaceful sailing”, as well as the name of the team’s platform lead, is the University of Michigan - Ann Arbor’s Autonomous Robotic Vehicle Team’s (UMARV) submission for the 2022 Intelligent Ground Vehicle Competition. Founded in the fall of 2016, UMARV is an interdisciplinary student-led project team that aims to provide its members with a hands-on introduction to autonomous robotics. We believe that hands-on education complements classroom learning, and any student can learn through robotics, regardless of their background.

Although the team was founded in 2016, this year has been a fresh start for the team. Due to COVID-19 and remote-schooling, the team has been unable to develop a vehicle to compete in the Intelligent Ground Vehicle Competition for the past two years. Additionally, the vast majority of the team graduated during the pandemic, causing this year’s team to be composed almost entirely of new members. Therefore, we decided to focus on rebuilding each subsystem of the vehicle to allow new members to have a comprehensive understanding of how the vehicle works.

UMARV is supported by both campus and corporate sponsors. The team is supported by the University of Michigan Robotics Institute. Our corporate sponsors include Ford, Bose, Aptiv, Northrop Grumman, Raytheon, and Underground Printing.

Team Organization

Our team is organized into six sub-teams: Business, Computer Vision, Controls, Embedded Systems, Platform, and Sensors. The Business sub-team handles sponsor relations, as well as the media/marketing side of the team. The Computer Vision sub-team develops the lane and pothole detection system using computer vision techniques. The Controls subteam develops the path planning, control systems, and simulation systems. The Embedded Systems sub-team develops the low-level programming, which includes motor control, safety features, the status indicator light, and the power delivery on the robot. The Platform sub-team designs and builds the robot chassis to fit within the given design requirements. The Sensors sub-team configures the robot’s sensors to compute odometry and Simultaneous Localization and Mapping (SLAM) to produce a map of the surrounding environment. Sub-teams frequently worked together to integrate their work into the final robot, and members were free to move between subteams.

The leadership team consists of the Team Lead, Operations Director, Engineering Director, as well as leads for each sub-team. The Team Lead, Operations Director, and Engineering Director provide the general direction and strategy for the team, and the sub-team leads focus on the technical development of their respective subsystems.

Design Process and Assumptions

Due to the team being new and inexperienced, we opted to dedicate the fall 2021 semester to onboarding, research, and design. Each sub-team underwent a multi-week onboarding project that aimed to educate members on fundamental concepts for that sub-team. The rest of the semester was dedicated to research, and each sub-team formulated a plan for how to approach the development of their subsystem for the following semester. The following semester, the team started the development of the actual platform.

Our team utilized an agile-based project tracking system called Zenhub. Zenhub integrates directly with GitHub, which is the version control system we use. The team utilized 2-week sprints for tasks that contributed to a longer 1-2 month goal, called an epic. Utilizing Zenhub allows for assignable, labelable tasks, and creates a lasting record of the development of our vehicle.

Some assumptions made during the design process include:

- Wheel slippage is negligible
- Only splash-proofing was considered when water-proofing the robot

Vehicle Design Innovations

The exterior panels for the GOAT use a quick attach system to increase serviceability. In response to pains with battery swapping and charging, we designed 3D-printed brackets to allow faster replacement of batteries through the GOAT's side panels instead of having to remove the top cover and disconnect all of our compute hardware.

Description of Mechanical Design

Overview

The objective of the mechanical system is to provide the chassis, payload support, and a maneuvering method for our robot, Kohei. The team used Solidworks for designing the platform and various machines for manufacturing including a horizontal bandsaw and laser cutter. CAD was used to create a model for parts of the platform system and to combine the parts for assembly. Figure 1 shows the CAD model of Kohei in Autodesk Fusion 360. A screenshot from Autodesk was used as it would show the systems and structure of Kohei more clearly than in Solidworks.

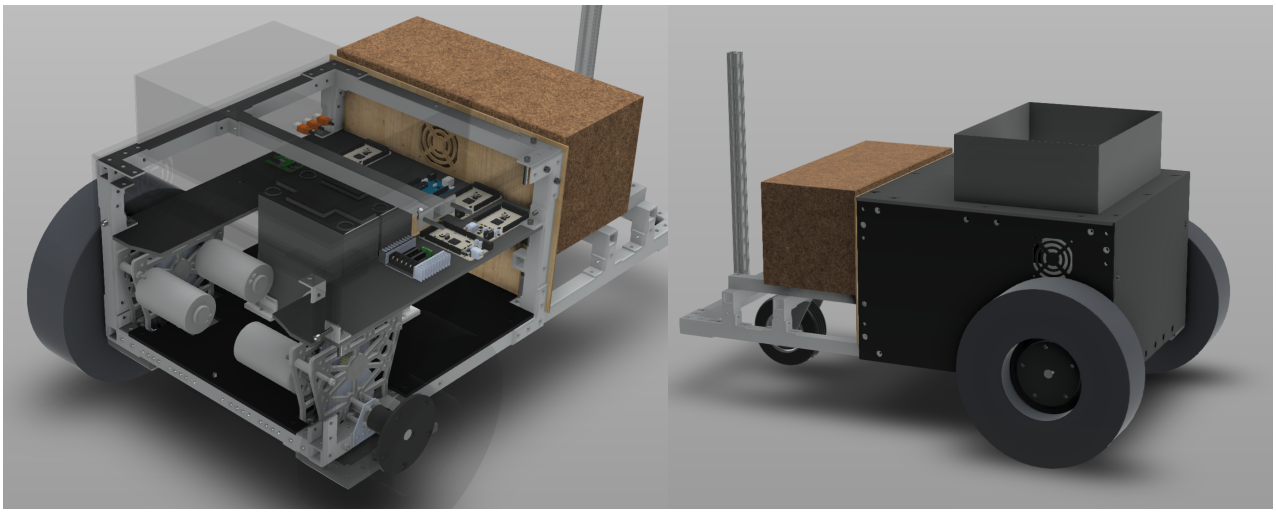


Fig 1: The structure and chassis of Kohei in Autodesk Fusion 360

Decision on frame structure, housing, structure design

Designed with portability, serviceability, and longevity in mind, the chassis is constructed from 1-inch square aluminum tubing in order to minimize overall system weight and maintain rigidity. The dimensions are within the competition requirements at 28 inches wide by 36 inches long and well below the height limit. The chassis is secured by custom-made mending plates and attachment brackets and secured with one common 1/4 inch thread hex bolt and nut to improve serviceability. The brackets and mending plates can be easily manufactured and the hex bolt ensures that no stripping will occur. Having a common bolt and nut type makes the entire system easier to service. Kohei features a two-wheel drivetrain with a third free caster wheel to provide a balance between power, stability, and maneuverability. Since the front wheels offer a majority of the support, the design is forward-heavy. Inside the main chassis of the robot, there are two gearboxes with motors, encoders, batteries, and a shelf for the embedded system components. The batteries are placed in the interior of the robot, behind the drivetrain to act as a counterweight, in combination with the payload. We used 3D-printed brackets to hold the batteries firm against the chassis. Above the main structure, there is an IP64-rated splashproof box containing the primary and secondary computers as well as the LiDar and a stereo camera attachments. At the rear of the robot rises a safety pole which features LED safety light and a physical emergency stop button, as stipulated by the rules requirement.

Suspension

Kohei lacks suspension in its structure as the platform team decided that it would create unnecessary weight on the vehicle. The team thought that suspension would not be needed as the competition would be on concrete, which is relatively flat and does not have many bumps. Although there would be slopes and inclines on the course, the shock would not be too large on the robot as the change in slope is not immediate. Also, the caster wheel has proven to be

useful in balancing out the robot on rough terrain. However, including suspension does improve the stability of the robot while in motion, so it is a possible future improvement.

Weatherproofing

The crucial components of the robot are weatherproofed since they are inside boxes in the front part of the robot. The black main structure is built from metal frames with plastic covering on all six sides, connected to the frames by velcros and bolts. This ensures that under any weather conditions, the systems inside would be protected. The IP64-rated splashproof box on top of the main structure contains the system related to the LiDar. By placing the system inside a secure box, the system can be connected to LiDar while being weatherproofed. The safety stop and weight holder can be outside the box because they would not be damaged by weather conditions.

Electronic and Power Design

Overview

The electronic and power design was implemented by the Embedded Systems subteam. Kohei was designed from the ground up as we designed a whole new electronic system since the last competition. This allowed the team to update the vehicle with O-Drive motor controllers with onboard PID tuning and new DC brushless motors. Fig. 3 shows a diagram of the electrical and power system.

Power Distribution System

The power system is composed of two 12-volt batteries connected in series to give the system 24-volts with a 70A circuit breaker being used to connect and disconnect the entire vehicle. Having two standard car batteries powering the vehicle allows great flexibility in swapping batteries quickly with a standard size. The batteries power the 24-volt power rail that then gives power to all the other electronic components. Since most components require 12-volts, a step-down transformer connects the 24-volt power rail to the 12-volt power rail which then powers the rest of the vehicle.

Electronics Suite Description

NVIDIA Jetson TX2

The NVIDIA Jetson provides a high-speed discrete GPU suitable for real-time image processing and pairs particularly well with the ZED camera, as Stereolabs maintains an SDK specifically for the Jetson TX2. The Jetson provides exceptional performance considering its power consumption, form factor, and price. In addition, the included development board has integrated HDMI, Ethernet, USB, and WiFi to speed up development.

Intel NUC

The Intel NUC provides a laptop CPU in a compact package without requiring the expertise of a dedicated embedded system. The i5 processor was selected after analyzing the expected computational power required to run navigational subroutines, such as path planning. The NUC provides all of the amenities to be expected from a high-end ultrabook without the peripherals not required for autonomous operation.

NETGEAR Switch

Strong long network connectivity was required to capitalize on the ROS' distributed system capabilities. The ZED camera can output 3D video at up to 2K 15FPS, so high bandwidth communication was a must. Having an Ethernet switch allows us to quickly link in new devices like laptops. Combining this modular system with the nature of ROS allows for quick visualization by connecting to the robot's local network with an Ethernet cable.

Velodyne VLP-16

Unexpectedly low obstacles in previous years' competition necessitated that we act quickly to install and integrate the Velodyne VLP-16 3D LiDAR; our previous LiDAR only scanned in a plane. The Velodyne boasts significantly increased range, accuracy, and weatherproofing over the RPLIDAR we used last year and has become an integral part of our sensor systems.

Stereolabs ZED Camera

The primary draw of the ZED camera is its low cost and high depth-sensing range. Compared to other RGBD solutions, the ZED camera offers much higher depth cloud resolution through software processing of the stereo images. Unlike systems that rely on infrared light, like the Kinect, the ZED retains nominal functionality in direct sunlight. The Stereolabs development team has provided a rich SDK with ROS integration included, speeding up deployment cycles by reducing hardware and embedded development time.

Phidgets IMU

The Phidgets ecosystem has a strong draw due to its well-documented and maintained C library and ROS integration. Another draw is the relatively low cost of the Phidgets system, but this results in less accurate sensor readings with higher noise.

Garmin GPS 18x USB

The previous years' GPS was connected via an RS-232 serial port and required significant jury-rigging to get powered as well -- it was designed to be powered from a 12V cigarette lighter socket. The 18x USB offers similar functionality, but with a more developer-friendly USB interface. The 18x is designed for automotive applications and as such comes weatherproofed, a significant factor in our decision to keep using the same model.

Arduino Mega

A simple hardware/software layer was required to interface between our ROS layer and the serial interface of the ODrive motor controllers. Using an Arduino Mega allows us to process ROS messages on a lower-level device, allowing the maximum abstraction of the drivetrain to the ROS stack. In addition, the ODrive's manufacturer provides and maintains an Arduino library to interface with the velocity controls of many motor controllers connected over serial, speeding up development and reducing testing time.

O-Drive Motor Controllers

The ODrive motor controller offers tight integration with velocity commands, having built-in PID position and velocity control. In addition, a wide variety of customization and diagnostic options are a significant quality-of-life boost while interfacing with hardware. For example, the motor controllers monitor battery voltage and motor current and will limit performance characteristics as necessary to stay within user-specified operating parameters. ODrives are also able to add a software velocity and current limiter that adds another level of safety for the vehicle.

Neo Brushless Motors

The Neo motors were chosen to drive the vehicle due to their onboard hall effect encoders and their torque at the desired RPM which gives around 5 mph with the gear ratios. It was necessary to change to brushless motors since the ODrive motor controllers are only compatible with brushless motors. This gave the opportunity to upgrade the motors and include the encoders into the motors instead of having to attach them ourselves. This reduced complexity and a point of failure of the vehicle.

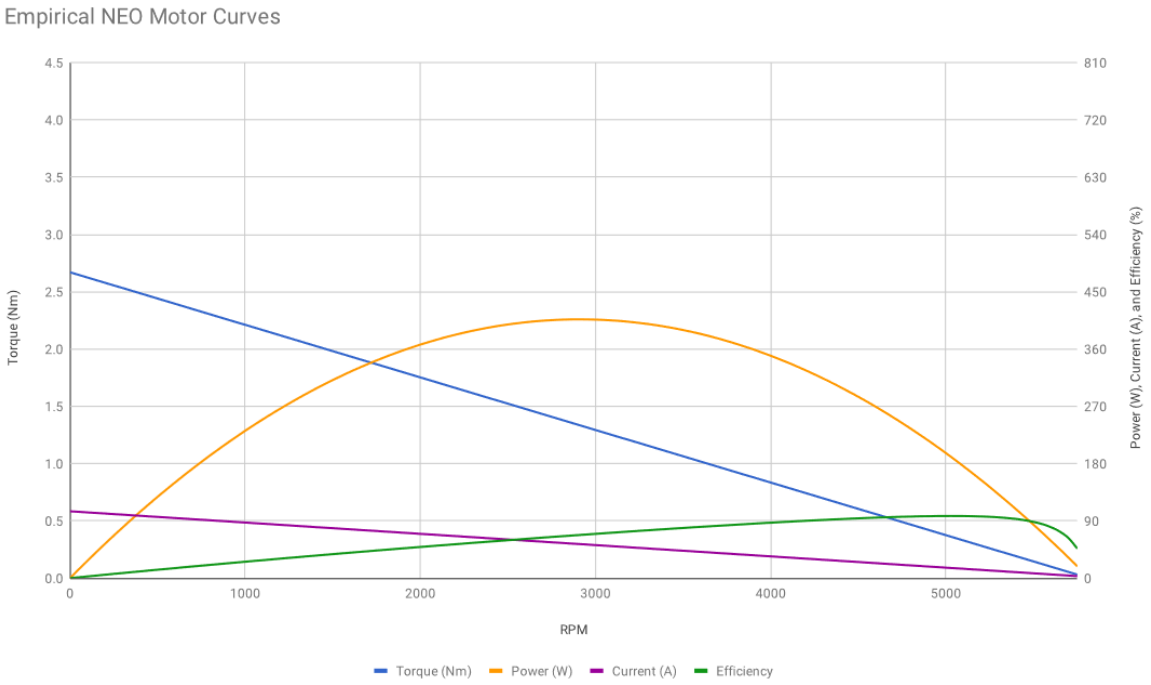


Fig. 2: Torque and Efficiency Curves for the Neo Brushless Motors

Safety Devices and Integration

Being able to operate our robot safely is a key part of the competition. When enabling the robot, main power from the batteries is enabled by flipping a circuit breaker mounted on the outside of the bot, easily seen and accessible by anyone. When the robot is turned on, power is supplied to a status light, showing its current state. In an effort to make the robot more modular, the Platform subteam designed 3-D printed mounts for the batteries that can be easily attached and removed.

To ensure that no safety issues arise during a run, a wired E-Stop, wireless E-Stop, and speed limiters are integrated into the robot using the Arduino microcontroller and O-Drive motor controllers. The wireless E-Stop we selected has a range of 250 feet, allowing the robot to be safely stopped from anywhere. The large red E-Stop button is attached directly to the motor controllers and the wireless E-Stop is connected to a digital I/O pin on the Arduino. Triggering the wireless E-Stop will cause the microcontroller to send a signal to the shutdown pin on the motor controllers. The O-Drive motor controllers are also capable of having a velocity limit, which we have set to 5 mph.

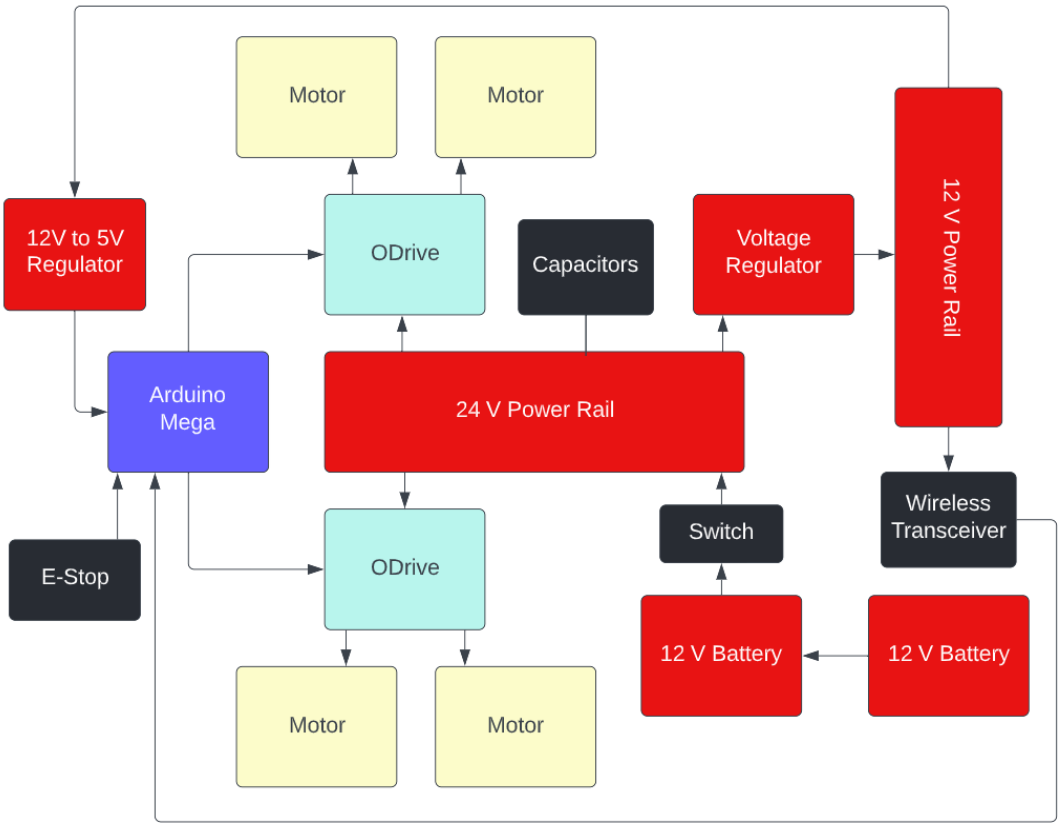


Fig. 3: Electronics and Power diagram

Software Strategy and Mapping Techniques

Overview

All of the robot's software is powered by the Robot Operating System (ROS) running on a base Ubuntu 20.04 installation. In line with our modular design philosophy, ROS was selected as the robot's operating system due to its extensive modularity, community support, and power features. ROS is a distributed networking and communications library allowing multiple devices to work together. A ROS computation graph is divided into discrete nodes that can publish and subscribe messages to build a network of information. Nodes communicate with each other over TCP, allowing them to connect to nodes on other computers through our Ethernet switch. This system facilitates the communication between different processes and enables the team to work on independent tasks; each software subteam can develop nodes entirely separately from the others.

The goal of the robot is to navigate through a series of waypoints while avoiding obstacles identified with data from the onboard sensors. The navigation system receives an occupancy grid, LiDAR data, odometry data (pose and twist estimates), and coordinate transform data to build navigational cost maps that the robot can plan a path through. Goals are generated by transforming the given GPS coordinates into the robot's world frame and updating the broadcast goal as each set of coordinates is reached. Fig. 4 shows the software architecture of the robot.

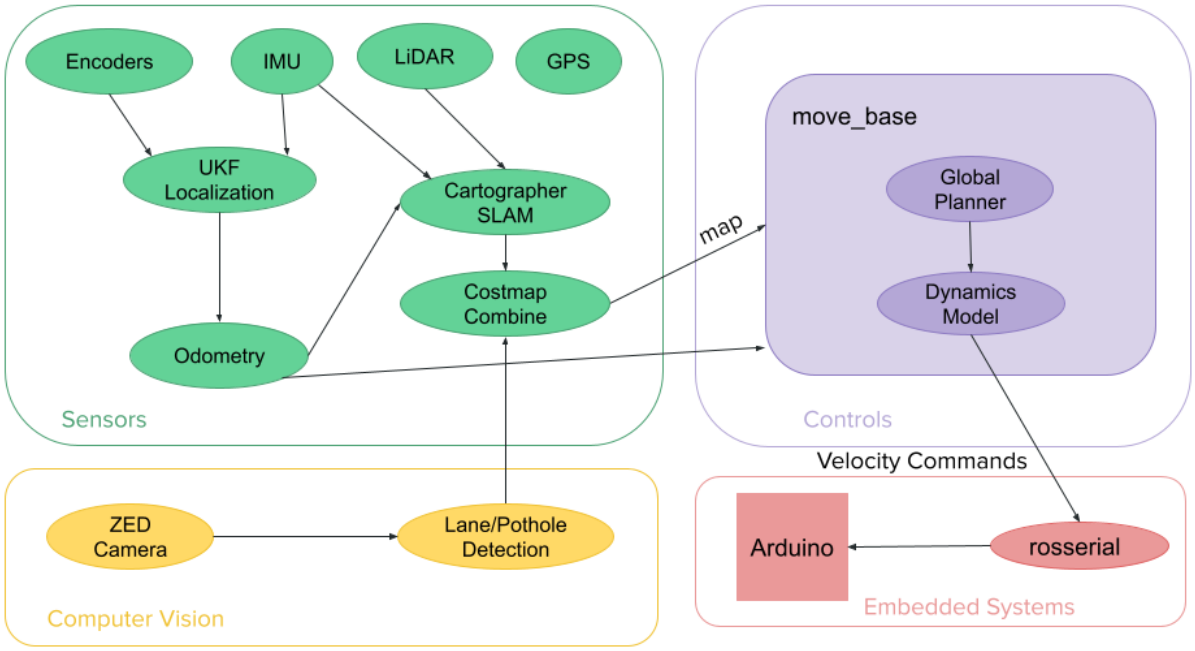


Fig. 4: Software architecture diagram of Kohei

Obstacle Detection and Avoidance

Larger obstacles are identified by our 3D spinning LiDAR, the Velodyne VLP-16. The VLP-16 boasts a 100-meter range, and data is organized in a point cloud format. This point cloud can be provided directly to our SLAM system to compute a global occupancy grid containing the surrounding environment.

For the robot to be able to detect 2D obstacles such as white taped lanes or white solid potholes, we use our Stereolabs ZED camera feed. Our Nvidia Jetson handles this task and creates an occupancy grid with the information required to see these obstacles. The approach we take consists of first converting the image into a bird's eye view using existing computer vision techniques. Subsequently, we perform a white line detection and a blob detection algorithm in order to extract the lanes and potholes from the image. The white line detection consists of a gaussian blur, canny edge detection, and Hough line transform. We fine-tuned thresholds based on test data we gathered. For the potholes we did a Hough Circles Transform on the original image in order to find random blobs. Additionally, we did a color threshold so that we can avoid seeing lines/blobs that are not white. Fig. 5 shows screenshots of the different steps our algorithm goes through. In order: original image, perspective transformed image, white-line/pothole detected image, occupancy grid.

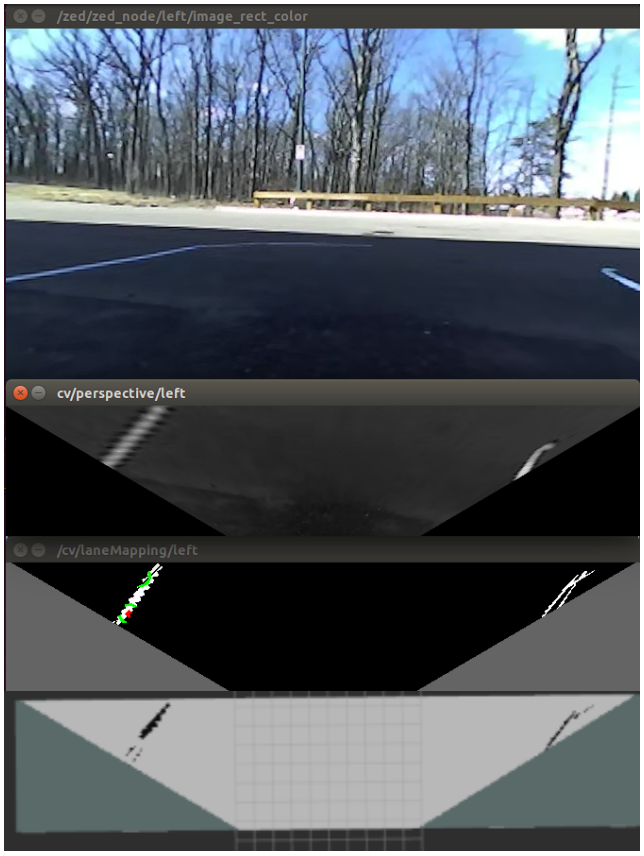


Fig. 5: Different steps of the lane detection algorithm

Map Generation

We utilize a sophisticated pose-graph Simultaneous Localization and Mapping (SLAM) solution called Google Cartographer. Cartographer offers a robust and highly configurable solution that permits us high confidence in the quality of generated maps, especially in noisy environments. Cartographer also has great potential for scaling, as its sparse representation of the environment allows for significantly more space to be mapped before running into memory usage constraints.

Cartographer integrates into ROS and provides an occupancy grid containing the obstacles identified in the point cloud data from the LiDAR. Following the Cartographer occupancy grid generation, we combine the occupancy grid computed using the stereo camera to have a single map that describes the location of the surrounding obstacles, and where the robot is in that environment.

Software Strategy and Path Planning

Sensor fusion between the IMU and wheel encoders is accomplished through an Unscented Kalman Filter, which is more forgiving than an Extended Kalman Filter when it comes to calibrating the sensor odometry. The GPS was chosen to be left out of odometric sensor fusion due to its non-continuous nature, which testing revealed significantly reduced the accuracy of pose estimates.

The final costmap from lane/pothole detection and SLAM is continuously provided to our global planner, which uses a repeated A* algorithm for path planning. After the repeated A* algorithm was implemented, we needed to make it a plugin for ROS. In order to get the planning algorithm to work with move_base, we followed the sample GlobalPlanner class written for move_base and structured our code similar to that one.

Goal Selection and Path Generation

The GPS waypoints are transformed into the robot's world frame to simplify path planning. Given a global costmap, a local target is found to move the robot toward the nearest GPS waypoint. In order to calculate this local target, the goal was to find a straight line to the GPS waypoint and use the intersection point between that line and the border of the cost map as a target position. In the end, we will receive a cost map from move_base, perform our search algorithm to find a path, and send a set of nodes back to move_base for the command velocities to be performed.

Additional Creative Concepts

At the start of the semester, we set out to implement the D* Lite algorithm as our global planner, since members of the club had independently ran tests & simulations comparing the Repeated A* algorithm vs D* Lite algorithm in maps of various difficulties. The D* Lite algorithm turned out

to run much quicker compared to the Repeated A* algorithm mainly because of the ability to keep adding new information to a single global map. After working on implementing D* Lite for a month, we realized that the way move_base was structured, did not integrate well with the D* Lite algorithm since it does not allow for dynamic replanning, where the map is always added to. Instead, move_base assumes a new map is always being sent to the planner. Because of the team's unfamiliarity with move_base, we decided that we would much rather change the algorithm rather than the existing framework.

Failure modes, failure points and resolutions

Vehicle Failure Modes

In case of SLAM scan matching algorithm failure, the newest odometry information is used to estimate the current pose of the robot. SLAM nodes are updated using forward projection according to the optimal solution for the pose graph.

The lane detection computer vision system may fail under certain lighting conditions. There have been cases where the algorithm incorrectly identifies lights as lanes. This can be dealt with by tuning the detection parameters based on the lighting conditions the robot is running in.

Vehicle Failure Points

Possible mechanical failure points are from the velcros. It is very unlikely that there would be a material failure because the frames of the robot are made of aluminum and the only significant stress comes from the weight of the different systems, which are light. The bolt attachments are also secure as there is almost no shear force acting on the bolt while the robot is running. The velcros connect the covering and the frame of the vehicle. Although there may not be any stress on the velcros, their connection may become loosened and fail as a result. In order to resolve this issue, the team will bring extra velcros so that replacing them can be easily done.

Failure Prevention Strategy

The general troubleshooting process for hardware and software failures is as follows. 1) Check that the status lights are lit and indicate nominal operation. 2) Check that connector cables are securely attached. 3) Verify that software nodes are running and messages are being transmitted. 4) Run ROS troubleshooting like roswtf, rqt_graph, and view_frames to verify that the node and message graphs are properly set up.

Testing

Due to the limited amount of resources, the team was not able to test the sturdiness of the mechanical components of the robot in the real world. However, the team ensured that the mechanical component would not fail by searching for materials that had enough strength to not

fail while running the robot. As evidence, the team had run the robot numerous times, but the mechanical component had not failed or had any signs of failure once.

Simulations employed

Simulations in Virtual Environment

The robot was simulated in Gazebo. This software was chosen given its integration with ROS is well documented, it is highly configurable, and it is open source. The sensor's hardware was simulated as accurately as possible using Gazebo plugins, which provided all the information necessary for the control stack to run. The information was published on different ROS topics to which our control subscribed to and used to transmit velocity commands back into Gazebo. On top of this, RViz was used to visualize the information that was going into the navigation stack which made debugging easier.

For the simulation environment, we primarily used Blender initially to arrange the different physical obstacles (cones, barriers, etc) and then ported them to Gazebo since team members were more familiar with Blender, and the port to Gazebo did not cause any problems.

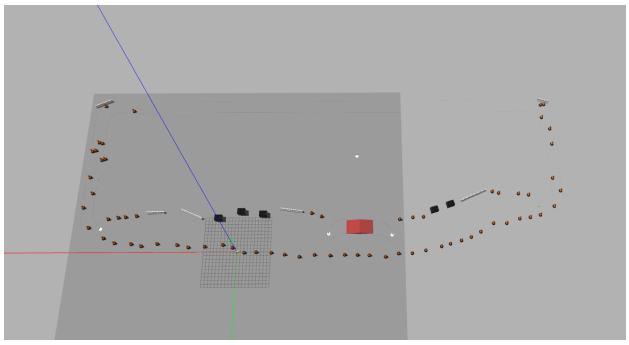


Fig 6: The whole map in Gazebo

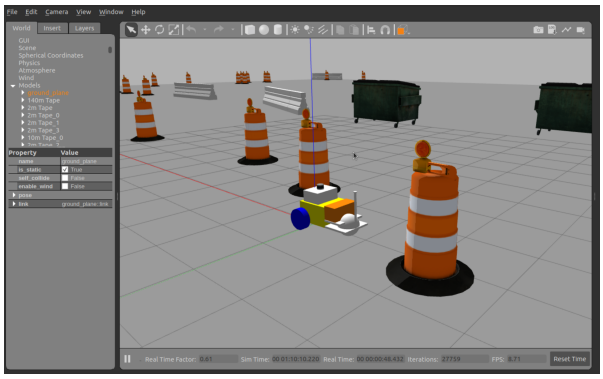


Fig 7: Zoomed in view of the environment

Theoretical Concepts in Simulations

Although we started out with the URDF (simulation model for the robot) from the previous year, there were a lot of problems with the actual simulation motion of the robot. First, it did not move correctly when passing in the velocity commands. We had to understand the structure of URDF files and understand how each attribute affected the simulation in order to fix the motion. After that, the center of mass of the robot was not working properly, causing the robot to often flip over when trying to drive it over a ramp. These issues were fixed through persistent and educated trial and error resulting in a very accurate simulation model.
After all the physical simulation problems were fixed, sensor simulators were added including LiDAR and IMU sensors.

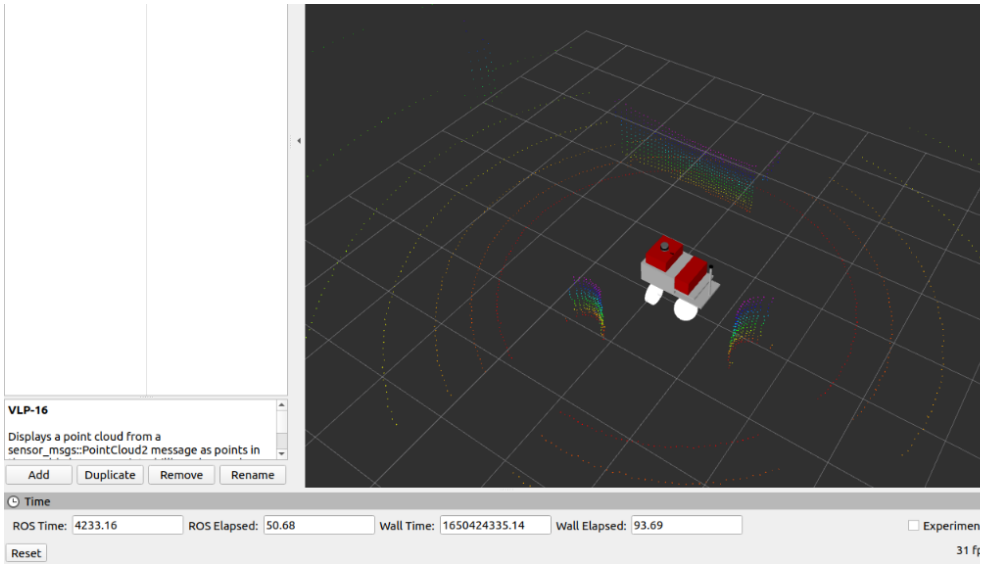


Fig 8: RViz of the LiDAR point cloud running on Gazebo simulation

Performance Testing To Date

Component Testing

Individual components on the vehicle have been tested and are working. Motor control is working and was tested via controller teleoperation. All sensors (wheel encoders, LiDAR, GPS, IMU, camera) are producing viable data. Both the mechanical and wireless emergency stops are fully functional.

Integration Testing

While the individual components are working, there have been some challenges with the final integration. Our odometry system has a significant amount of noise and drift, causing the map generated by SLAM to lose accuracy quickly. However, when odometry is stable, the resulting map produced by Cartographer is accurate. Additionally, there have been challenges with integrating global planning into move_base. D* Lite and A* planning algorithms have been developed and tested outside of ROS, but the team has not been able to successfully integrate global planning into the robot.

Initial Performance Assessments

Due to the challenges of integrating the various technical subsystems on the robot, initial performance assessments show that the vehicle struggles to autonomously navigate its surroundings. However, the team is in a good position to improve on this year's submission, so further development of the vehicle can improve its autonomous capabilities.