IGVC 2024 Self-Drive Design Report

Lawrence Technological University

Team: ACTor

***Date**: May 15, 2024*

Team Captain	Devson Butani	dbutani@ltu.edu
Members	Ryan Kaddis	rkaddis@ltu.edu
	Milan Jostes	mjostes@ltu.edu
	Vipul Prajapati	vprajapat@ltu.edu
Assoc. Members	Sean Kill	skill@ltu.edu
	Travis Bowman	tbowman@ltu.edu
	Aaron Wisneski	awisneski@ltu.edu
Faculty Advisors	Nicholas Paul	npaul@ltu.edu
	Justin Dombecki	jdombecki@ltu.edu
	Giuseppe “Joe” DeRose	gderose@ltu.edu
	ChanJin “CJ” Chung	cchung@ltu.edu

Faculty Advisor Statement

I, CJ Chung, Nicholas Paul, Joe DeRose, and Justin Dombecki of the Department of Math and Computer Science at Lawrence Technological University, certify that the design and development on the ACTor research platform by the individuals on the design team is significant and is either for-credit or equivalent to what might be awarded credit in a senior design course.

1. Conduct Of Design Process, Team Identification And Team Organization

Team Introduction and Organization

Team ACTor consists of undergrad students and graduate students. Associate team members worked on this IGVC project in a regular lecture class, Deep Learning, as a term project in the class. After the team was officially formed on January 9th, 2024, we have a team meeting every Tuesday evening to check the progress and discuss issues and problems. Each member was delegated to specific tasks in the beginning based on prior experiences and interest. Weekly team meetings also provide a way to collaborate in creating new ideas and divide up detailed sub tasks for the upcoming week. In-person interaction is an efficient and effective method for sharing information within a team. Roughly, each team member puts in about 5 hours a week for coding, testing, documenting, maintaining, code reviews, and meetings. See Table 1.

Name	Degree Program	Primary Responsibilities	Hours
Devson Butani	M.S. Computer Science	Team lead, redesigning software architecture, updating car and sensor hardware, maintaining source code, and analyzing new feature viability	400
Ryan Kaddis	B.S. Computer Science	Simulator development; Turns, Tire & pothole detection and avoidance	250
Milan Jostes	B.S. Computer Science	Unit Testing, Performance Assessment, Lane Detection, Lane Centering & Keeping	200
Vipul Prajapati	M.S. Computer Sci.	eStop, Turns, & Parking	140
Sean Kill	M.S. Computer Science	Associate team member; Collecting data, annotation, training models	100
Travis Bowman	B.S. Robotics Eng & B.S. Computer Sci.	Associate team member; Collecting data, annotation, training models	90
Aaron Wisneski	B.S. Computer Science	Associate team member; Collecting data, annotation, training models	42

Table 1: Team members and roles

Design Assumptions and design process

Instead of using traditional waterfall process models, our development philosophy is based on the idea of agile development. We try to meet IGVC requirements through early, continuous, frequent, and incremental testing of the working system of which the critical component is the software. We fully understand the project progress is measured by working software. Simplicity is essential too - we are always targeting to test the most important functions while avoiding tasks that do not contribute significantly to the end goal of the planned delivery/testing. However, a serious problem is the Michigan weather. It is too cold to test the vehicle system physically outside till full spring since our vehicle does not have a heater. Additionally, we cannot test the vehicle at night because it is too dark outside. Considering those problems, we set our development strategies as the following:

1. We actively use simulators to develop initial code
2. Test the initial code using the vehicle indoors as much as possible until the midterm evaluation period at the end of February
3. After the spring break and when summer time begins on March 10th (it is not dark around 7pm after summertime), we use our small outdoor course on campus to continue to develop and test our system
4. We try to have as much experience as possible by developing the system in different environments so as to be fully experienced to different environments for the unknown real IGVC self-drive competition 2024 course

Figure 1 below shows our team member onboarding strategy and validation timeline using multiple test environments.

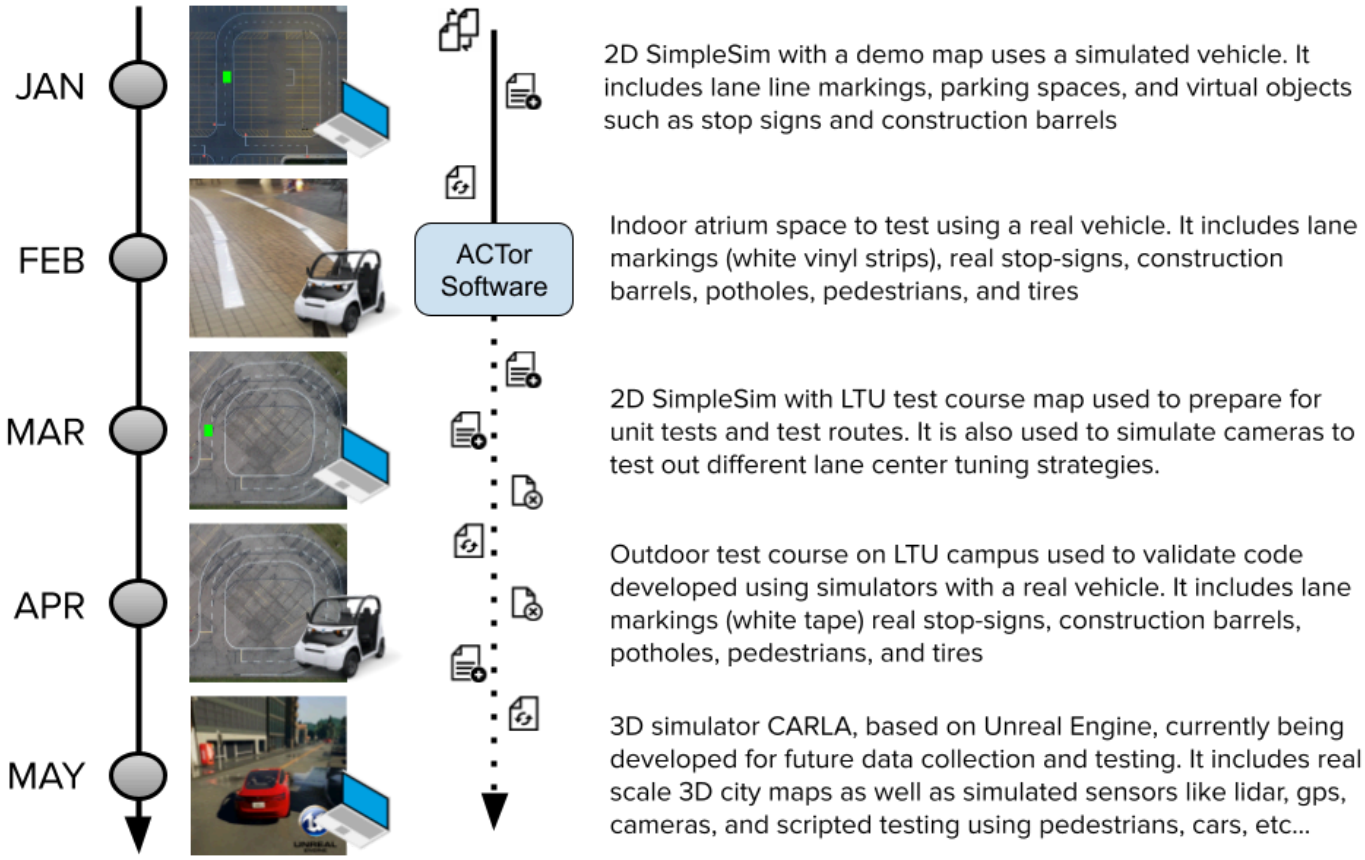


Figure 1: Team member onboarding strategy and validation timeline using multiple test environments

2. System architecture of our vehicle

ACTor (*Autonomous Campus Transport*) is an autonomous vehicle research platform built on a Polaris GEM e2 base jointly sponsored by MOBIS, DENSO, Veoneer, Realtime Technologies, Dataspeed, and SoarTech. The vehicle is equipped with Dataspeed drive-by-wire system, two vision sensors, one 3D Lidar. two 2D Lidars, and a RTK GNSS; all connected via ethernet and USB to a primary laptop with discrete GPU and a Raspberry PI 3. Major hardware changes from 2023 IGVC [1] include addition of a wide angle WDR (Wide Dynamic Range) camera, updated LED display controller, and active emergency braking. Figure 2 shows a high level component interface diagram.

For safety, ACTor's emergency stop system monitors multiple layers of hardware and software checks, and allows activation via physical BRBs (Big Red Buttons), radio switch (2.5GHz) or using the web UI over the local WiFi network (2.5/5GHz) which uses our new Python-based self-drive software architecture.

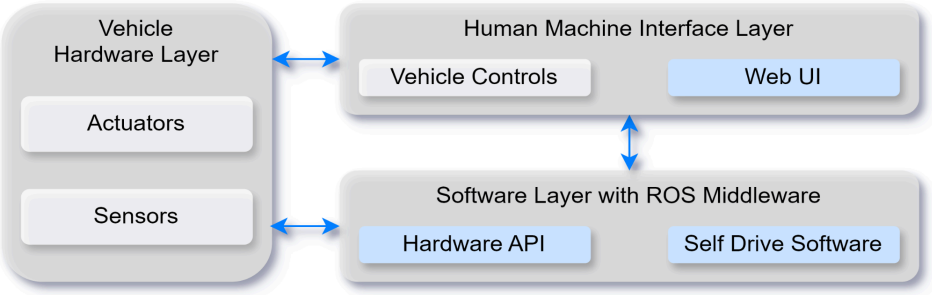


Figure 2: High Level Component Interface

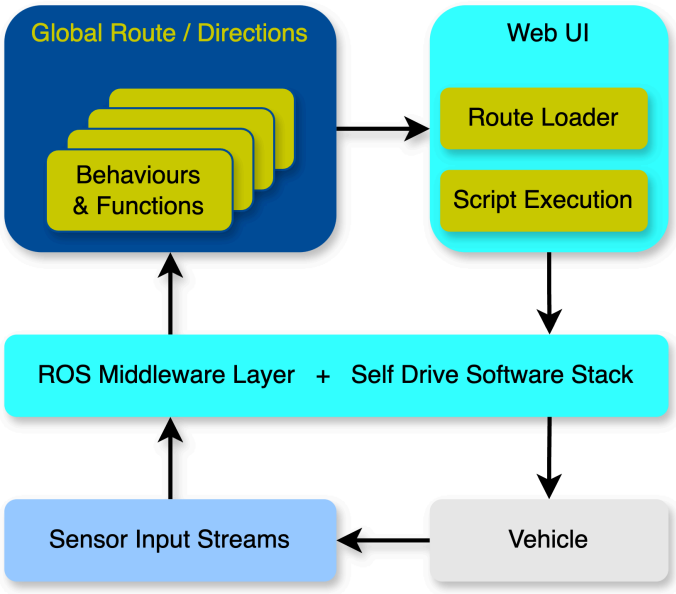


Figure 3: High Level Software Architecture

For this year's IGVC self-drive challenge, our software architecture has been completely re-designed (see Figure 3) compared to the 2023 version. Instead of using pre-compiled Lua scripting language that needs to be re-loaded every time a change is made, we now leverage Python's interpreted nature as well as multiprocessing capabilities to specify the top level vehicle behaviors for route generation.

Our lower level utility software has been re-written in Python to enable updated vehicle control strategies, modularize packaging to streamline CI/CD, and efficiently onboard new team members at different academic levels.

ACTor now uses a new Web UI accessible over the local network; developed using Nice GUI and Python to enable simplicity and offer real-time configuration.

3. Effective Innovations In Actor Design

- Moved from using pre-compiled behaviors to Just-In-Time compiled interpreted behaviors for route generation. Seeing how last year's software needed extra steps between making route changes to actually executing them on the test course and the use of LUA scripting (a separate language/syntax that members needed to learn), we chose to redesign the system using Python and effectively remove the entire process. Now, members can simply make alterations to the code and hit run on the web UI after reloading the script.
- Vehicle control strategy is now directly using closed-loop road angle (angle that the wheels point to) instead of the conventional variable 'yaw rate' for steering for geometric reproduction and repeatability of vehicle path in different vehicles and environmental conditions. One of the reasons is that this breaks down the function's core interpretation and in the future, this will simplify training neural networks that could work on simulated vehicles (especially in CARLA) or even vehicles of different makes or models.
- New Web UI brings open customization, real-time vehicle status and enables the interpreted nature of Python to play routes for a simplified field (or simulation) testing experience. The UI is flexible on what gets displayed and users can select their data of interest to view remotely in the vicinity of the vehicle's local network. This was more of a necessity than anything else as the car only seats two, removing other members from contributing their informed perspectives while on the test course.
- Our new YOLOV8 model for object detection uses a unified approach and OCR (Optical Character Recognition) for reduction of sign detection false positives. Halfway through the training process, we realized that all these separate neural networks were being run to detect separate objects of interest from the same camera feed. By combining all of the training datasets, we developed a unified model to improve the efficiency of our detection and classification. The unified model allows for a simpler approach that is more VRAM and power efficient than the previous methodologies without sacrificing detection accuracy.
- We added hardware based E-Stop to direct hardware braking. The braking aspect was moved directly to the brake actuator via the DBW firmware. This was inspired by the fact that the previous strategy was to send a 0 speed goal for any emergency situations. Even though it worked every time, we needed the ability to tune the amount of braking. Moving this functionality to the DBW gives more safety than before because it will trigger E-stop if something goes wrong at a hardware level. It now applies constant brake pressure as saved in embedded memory in the firmware. For example if the main laptop or RPi crashes, or a wire becomes disconnected, the E-Stop automatically gets activated without the need for any software.

4. Description Of Mechanical Design

ACTor platform uses a drive by wire system developed and installed by Dataspeed Inc; to enable maximum safety as well as complete vehicle control at all times. With this, our self-drive hardware and software stack acts as an overlay to the vehicle giving absolute override to the driver when needed.

Our hardware philosophy is in line with modularity and cross compatibility, meaning our software can be used on any vehicle as long as the required hardware is present and configured; as evident in our second ACTor vehicle already being used for computer vision and deep learning research at LTU. Even though both vehicles use different hardware configurations, our component mounting strategy and accessible electronics bay allow researchers to plug-in their own hardware with ease.

Since this vehicle's primary use is research, safety and ease of use are higher priority than aesthetics. Hence, the entire dashboard has been replaced with a hinged mounting board allowing for easy access to all non-permanent electronics and connections such as Ethernet, USB or low voltage power cables. This allows for fast reconfiguration of sensors or painless debugging; the working area remains within easy reach and offers complete visibility of each and every component.

As seen in Figure 4, ACTor is pre-equipped with self-drive enabling sensors; all of the external mounted sensors (roof rack, front/rear bumper, etc.) are rated IP67 or similar. The Polaris Gem e2 base being used has weather-sealed doors and windows installed as options from the factory. These sensors are connected to the in-vehicle components through either a weather-resistant conduit running through the passenger door frame or from under body panels. This allows for fast and easy access to individual cables as well as keeps them tucked away for safety and a robust connection.

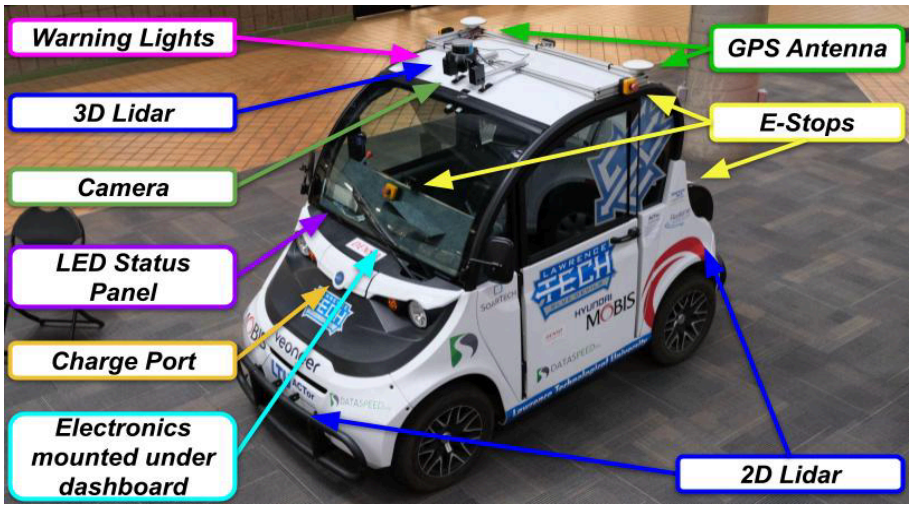


Figure 4: ACTor hardware suite

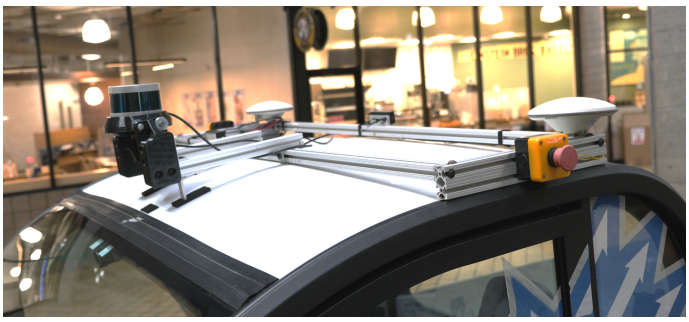


Figure 5: Extrusion based Roof Rack

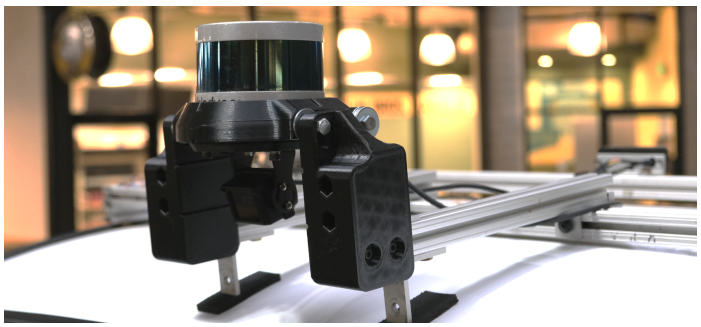


Figure 6: LiDAR and Camera Mount

As shown in Figures 5 and 6, ACTor relies on T-slot aluminum extrusions and 3D printing for most of the mounting needs. For our targeted research use cases, minimizing ideation to implementation time is very important. By using a simple extrusion roof rack, we have fastened the prototyping process of different LiDAR mounts as well as testing various vision sensors. The roof rack allows changing sensors and quickly prototyping their respective mounts. One of the latest updates include an adjustable LiDAR and camera mount that reduces error in relational data transforms used for perception of the environment. Table 2 shows the cost of the ACTor project for IGVC2024.

Item Specifications	Price
Polaris GEM e2 vehicle with various options such as doors and trunk	$15,000.00
New Polaris GEM ADAS Systems (Drive-By-Wire systems by Dataspeed) including installation fee	$35,000.00
Velodyne VLP-16 “PUCK” 3D LiDAR, 16 beams	$7,999.00
Hokuyo UTM-30LX 2D LiDAR	$6,500
Hokuyo URG-04LX-UG01 LiDAR	$975
Swift GPS, Piksi Multi GNSS	$1,644.56
Swift GPS, Additional rover module and antenna to get GPS heading info	$896.00
Mako G-319 PoE Camera with 6mm vari-focal lens	$1,031.00
E-Con Systems RouteCAM CU22 IP67 - Outdoor Lowlight GigE HDR Camera	$379.00
MSI Gaming Laptop, Intel 8-Core i7-11800H, 16GB RAM, 512GB SSD, GeForce RTX 3050 Ti 4GB	$1,258.99
Miscellaneous items including lenses & filters, e-stop switches, wireless e-stop, LED strobe lights, cabin camera, RPIs, inverters, switches, router, mounting rack, and LED panel, USB HDR cam, etc.	$3,000.00
Total	$73,683.55

Table 2: Cost of ACTor vehicle and hardware

5. Description Of Electrical And Power Design

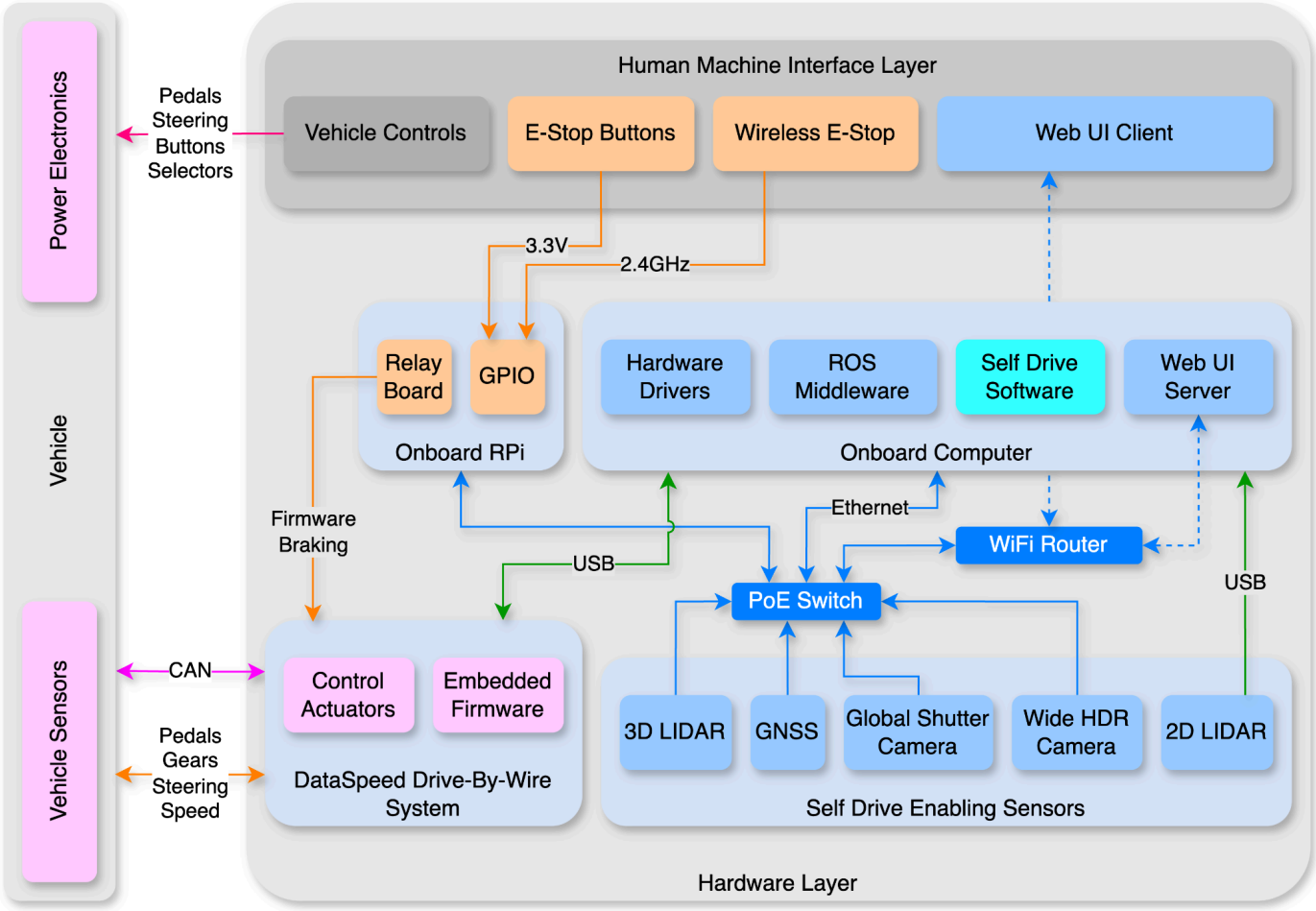


Figure 7: Electronics suite - with their interfaces and system layout

The factory-installed batteries take 6-8 hours to charge fully and provide 20 miles of range (while using all autonomous electronics). The DataSpeed Power Distribution System protects components from overload and offers control of all the circuits via a touchscreen interface or the CAN bus. Some of the power efficiency losses come from the 1kW inverter (92% eff.) and DC to DC converters (92-96% eff.) adding upto 800W that supply a range of voltages to individual components. Using separate converters allows adequate power as well as fusing. Figure 7 shows ACTor’s electronics suite. Figure 8 describes the power distribution strategy.

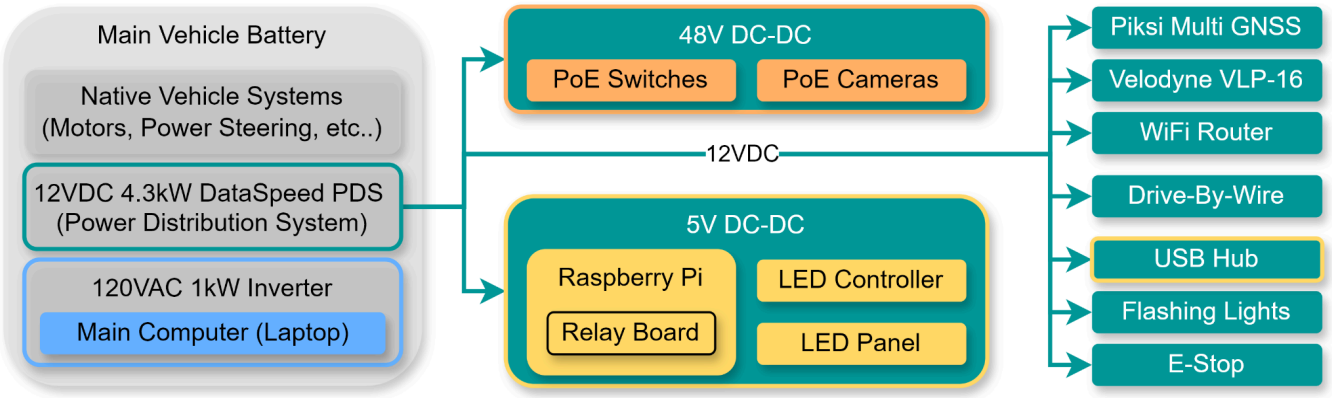


Figure 8: Power Distribution Strategy

The vehicle's components are controlled by a laptop (Intel 8-Core i7-11800H, 16GB RAM, 512GB SSD, GeForce RTX 3050 Ti 4GB VRAM) that runs Ubuntu 20.04. This laptop also runs our route scripting tools and performs object detection and sensor fusion using the camera and LiDAR. These sensors enable high-accuracy real-time detection and 3D positioning of pedestrians and obstacles. The laptop's discrete GPU allows us to use deep-learning models such as our alternative lane-following system and Yolo v8 based detection algorithms. We also use Raspberry Pi 3B+ for hardware e-stop, remote e-stop, safety monitoring and status lights, and an LED panel to display status information outside the vehicle.

The Polaris Gem e2's motor controller limits top speed to 35 mph and is able to provide a range of 20 miles. However, while under autonomous mode we enforce lower speed limits using the Dataspeed Inc ADAS Development Vehicle Kit. This drive-by-wire (DBW) kit allows the vehicle to be driven using native controls (driver operation) or electrically actuated hardware via ROS.

Their firmware modulates the accelerator/brake pedals, steering wheel and gear selection hardware to achieve the desired linear and angular velocity targets for the vehicle. Moreover, direct steering angle targets can be provided to control maneuvers. We use the default firmware parameters, a velocity dependent linear acceleration limit of 0.9 - 1.2 m/s2, and a constant deceleration target of 1.5 m/s2. Figure 9 shows the linear velocity response with the default acceleration limits for the 1.5 mph speed limit test.

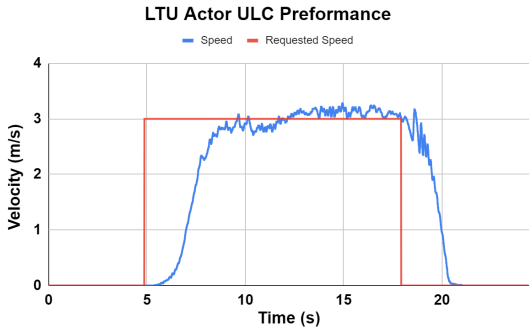


Figure 9: DBW control responsiveness

Our vision system consists of an Arducam IMX291 camera mounted under the 3D LiDAR on the roof. It captures 1080p images using 2.9um pixels with Sony Starvis sensor technology and a 120 degrees field of view. This allows us to capture the lanes and clearly detect road signs up to 30 feet away. The 72dB of Dynamic Range enables preserving color accuracy and sharpness in varying lighting conditions.

A Velodyne VLP-16 "Puck" LIDAR [4] donated by Veoneer provides 360 x 15 degrees of field of view with a radius of 100 meters. It outputs 300,000 points per second across its 16 channels. We also use Hokuyo URG-04LX-UG01 [5], a small two-dimensional lidar with 4 meters of range, mounted on the front and the back. It helps us detect immediate obstacles of significant size like tires, cones and curbs. These 3 lidars also assist us during parking operations.

To navigate, the vehicle uses two Piksi Multi Modules [6], which are RTK GNSS receivers that can access multiple bands and constellations. These modules provide position and heading data with centimeter-level accuracy and high update rates. This is ideal for a moving vehicle that needs to cover large distances in a short time.

We use Ethernet connections for most of the components to ensure reliability and ease of debugging in a noisy electrical environment.This networked architecture also allows remote access for testing and monitoring purposes. The only components that do not use Ethernet are the 2D lidars, the cabin webcam, and the USB CAN interface to the DBW system. The vehicle will automatically activate an emergency stop if it detects any failure in the DBW connection.

The emergency stop system consists of two parts. The first part is a loop circuit that connects all the E-stop buttons to a Raspberry Pi. Any button push will cause the circuit to break, allowing Dataspeed DBW hardware to

safely stop the car via direct braking with the aid of a relay attached to the Raspberry Pi. The second part is a heartbeat mechanism that requires constant communication between the main computer and the Pi to enable any interaction with the DBW. Additionally, the Pi controls warning lights on top of the vehicle that flash when the vehicle is in autonomous mode. This dual safety system ensures that all critical components are functioning properly before allowing autonomous driving.

6. Description of software system

Since the primary function of ACTor is to enable research and development in self-driving, our software architecture requires easy onboarding for new students, and a seamless integration of new ideas. Using the distributed and modular software design principles of the Robot Operating System (ROS), our software design enables quick development cycles by allowing its inputs and outputs to be interchangeable. With these specifications in mind, our software stack is designed to be quickly tested and allows for smooth implementation with ever-changing hardware and software components.

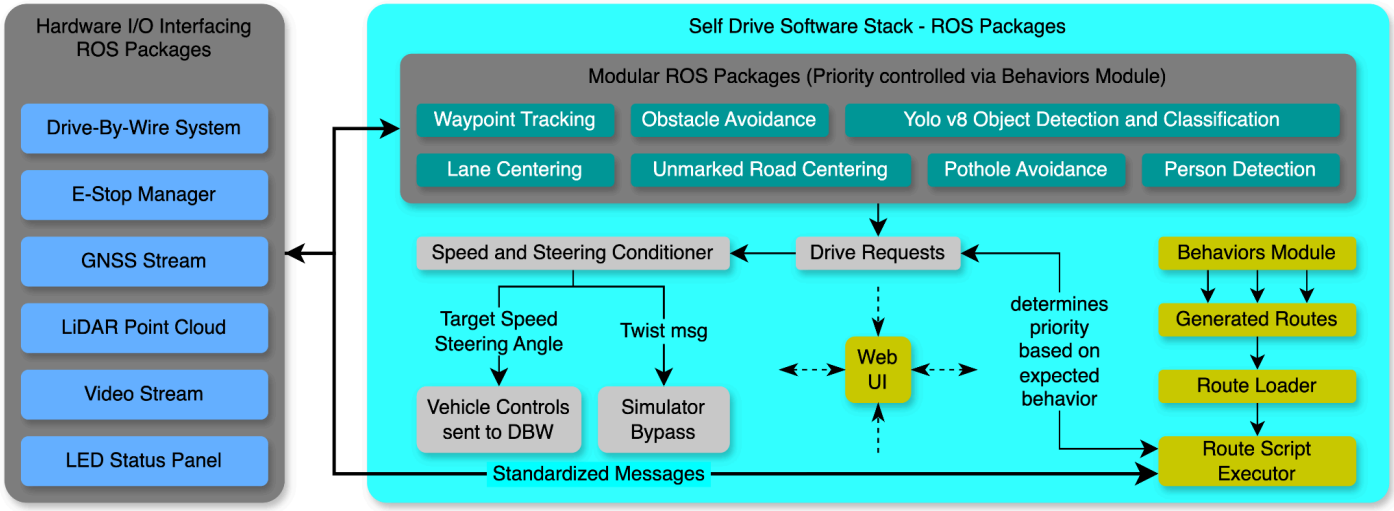


Figure 10: Description of Software System

Our system architecture (Shown in Figure 10) is distributed across various packages that use ROS as the middleware to communicate with our self-drive software stack; packages include sensor drivers, I/O processing, perception modules, core control, status updates, and web user interface.

Perception and Localization

Our hardware interfacing sensor packages contain publisher nodes and configuration files for each sensor (e.g. GPS, LiDAR, camera) that take in raw values from sensors and convert them to ROS messages, creating higher level input abstractions for the vehicle navigation system. This enables fast and easy integration of new sensor hardware or software as the ROS message types and topics are standardized for each sensor type. For example, the 2D and 3D LiDARs use three separate nodes to gather raw data. The long-range 3D surround LiDAR (Velodyne Puck mounted on the roof) outputs a standardized PointCloud message with its frame of reference relative to the vehicle's baselink. Front and rear short-range 2D LiDARs (Hokuyo URGs mounted on the bumpers) used for close range avoidance and parking outputs a standardized LaserScan message also with its own frame of reference relative to the vehicle's baselink. Using these reference frames allows the self-drive software stack to fuse the LiDAR data together when required for tasks that need long and short range clearance data.

Similarly, using a two-antenna Piksi Multi GNSS configuration, ACTor is capable of high precision position and heading accuracy using SBAS. With an optional base station (used outside of IGVC), a RTK fix can be achieved for centimeter level accuracy as well as EKF heading corrections. Our ROS nodes use positional coordinates (lat/long in ECEF format), inertial measurement units (IMU) and heading info (NED - north east down). This GPS data is then used to calculate the distance and angle from static waypoints. For example, during the merging behavior, the route includes a set of waypoints departing from the current lane to the directed lane; the calculated target steering angle from the waypoint node takes over priority control for a moment to depart the vehicle from one lane to another and then lane centering takes over priority again.

Obstacle Detection with 2D & 3D LiDAR

The obstacle avoidance ROS package uses input from the VLP-16. Using built in functionality, ground and obstacle PointClouds are generated from the VLP-16's input. For close proximity detections, the Hokuyo 2D LiDARs pitch in with their laser scans. The current implementation of the obstacle avoidance algorithm checks arbitrarily predefined regions around the car as shown in Figure 11. These regions not only report if any object is present but also calculate their closest distance from the vehicle, altogether this helps the behavior executor to determine the action to be taken. For example, if an obstacle is within an emergency region, the vehicle will halt. If it is far ahead in the road, it may execute an avoidance maneuver or halt depending on the route and outputs from other obstacle classification nodes.

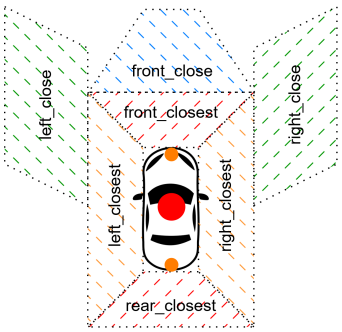


Figure 11: Regions of Interest

Object Detection and Classification

The previous year's implementation for object detection and classification used separate YOLOv8 models for tire, pedestrian, and sign detection, as well as traditional computer vision techniques for pothole detection. The separate models used approximately 4GB of VRAM, which we wanted to reduce. We thought that creating one model to detect all of our object classes would be the best way to do this. This was possible because null images for each of the classes were the same, so the classes were compatible for training a single model to detect all of them.

We developed a unified model that was trained on the previous datasets for tires and signs, along with newly created datasets for the simulated potholes and pedestrians. The new dataset totals around 10,000 images, with 4543 stop signs, 2313 pedestrians, 1323 potholes, 1867 tires, and 975 null images. After using data augmentation the number of images increases to 20,589 images. The unified model performed better than the previous implementation of individual models for each class.

We load the model into memory when object detection is called via a published message, the number of detections for each class is published. The biggest bounding boxes for each of the corresponding classes are published and used to determine the distance from an object.

Optical Character Recognition (OCR) is also implemented with EasyOCR to reduce false positives in stop sign detection. Vest detection is achieved using a simple orange mask when a pedestrian is detected. The unified model in combination with OCR uses approximately 1.6GB of VRAM compared to the approximately 4GB of VRAM the previous individual model implementation used.

When the camera detects the tire or simulated pothole with a high confidence level (see Figure 12) and is within a close range from the vehicle, the lane changing program will be triggered. Lane changing uses dead reckoning to turn for a certain amount of time and reactivates the lane following algorithm afterwards.

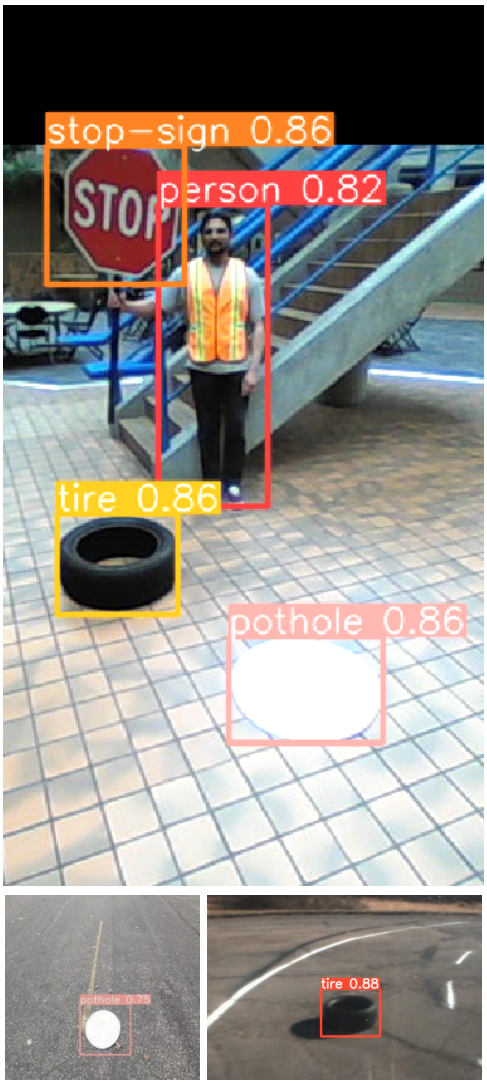


Figure 12: YOLOv8 Object Detection

Lane Following

Lane following involves two primary tasks: lane detection and maintaining a position in the center of the lane. There were three different algorithms developed, however the 'blob' algorithm was chosen for the final process. Figure 13 illustrates the steps involved in the lane detection algorithm.

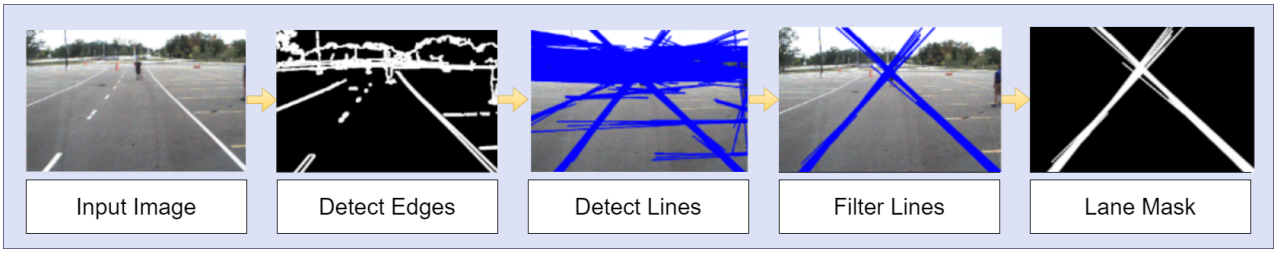


Figure 13: Lane detection filtering process

The process begins with the application of OpenCV functions, such as Canny edge detection, blurring, and dilation, to transform the initial image and accentuate its edges. Subsequently, a Hough transform is then applied to these edges to identify lines. The detected lines are filtered, where only those at approximately 45 degrees to the vertical are retained. These lines are then extended to create the final lane mask called the 'blob' lane mask. For lane centering, this mask is used to generate dynamic forces or springs that help guide the vehicle toward the lane's center. In Figure 14, a series of probes extend from the vehicle's front center (marked as a blue dot) to detect the Hough lines, and form springs at these intersection points. The force exerted by each spring, which varies with its length, contributes to positioning the vehicle within the lane.

Blob Lane Detection

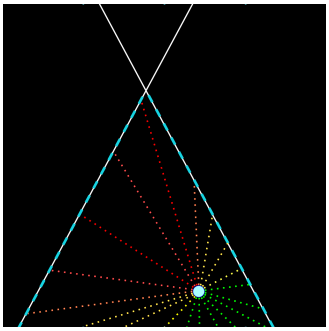


Figure 14: Springs form where rays from the vehicle intersect with a Hough line.

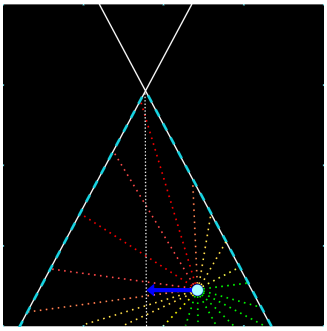


Figure 15: Springs with the shorter lengths push the vehicle's center toward the lane's center, while springs with longer lengths pull.

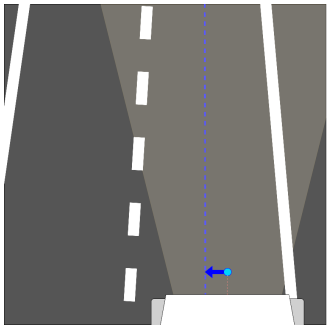


Figure 16: Lane centering reduces the distance between the front center of the vehicle and the center of the lane.

The lane centering algorithm aims to minimize the distance between the vehicle's front center (Blue Dot) and the lane's center (Dotted Line). The springs work to equalize this distance through the application of forces, which serve to reposition the vehicle's center towards the midpoint of the lane. The horizontal force components from these springs serve as steering inputs for the vehicle. Figures 15 and 16 contextualize these mechanics with respect to the vehicle positioned within a lane and within the 'blob' node, respectively.

Mapping and Navigation

Our current approach to self driving is to stay as close to human behavior as possible. We've also shifted from a common ROS approach of using rates of directional velocities (linear/yaw) to using direct steering and pedal values to control the vehicle. This strategy allows us to operate the vehicle in any environment (outside of IGVC) with efficient compute power as well as streamline data collection (from real vehicles or CARLA simulator) for future AI based self-driving research. For navigation, we are strictly using real-time local scene data (3D point clouds and wide angle images) to handle features such as lane centering, obstacle avoidance and other detections. Global scene information (GPS coordinates or map) is only used for getting a general direction to make a route to follow and trigger steps in the route; for example, turn right at the intersection once passed these coordinates. In the event lane lines are not available, we can momentarily fallback to using GPS coordinates and head in the direction of the next waypoint. The accuracy of GPS-only navigation is environment dependent (weather, tall buildings, bridges, etc...) hence not reliable on its own; we rely on real-time vision and point cloud data for local control while the GNSS provides corrections and/or rough directions toward the next waypoint. As soon as lane centering is confident on its detection again, the priority is switched back. This is all described by Figure 17.

One of our research projects include DeepSteer, a convolutional and recurrent neural network based driving

approach on roads without lane markings. [11] This method was developed using our second ACTor vehicle and can be effectively enabled in our self-drive software stack; however, it is not required particularly for IGVC use cases. Traditional scene mapping techniques to generate self-drive operational design domains are avoided to maintain simplicity, compute and data efficiency, and pave the way for future development of end-to-end neural network based self-driving.

This will eventually help lead our self-drive software stack towards being able to drive in any environment using any vehicle as long as it is configured to our specifications.

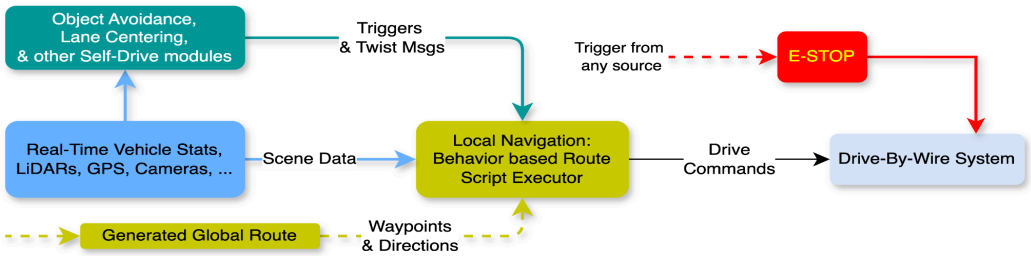


Figure 17: Navigation Design

Our simplified route system can be easily visualized as a switchboard operator following a set of directions, controlling inputs from various environment perceptions to output certain vehicle behaviors. The behaviors are made to directly influence vehicle controls; for example, lane centering directly adjusts the steering wheel angle while the front LiDAR checks for obstacles and adjusts the speed accordingly. Behaviors like stopping at the intersection are simple priority overrides, from the LiDAR checks to stopping at the sign, that control the speed and stop for the allotted time. Similarly, the perception packages (detection and avoidance) can override both steering and speed control if needed for behaviors like stopping for a pedestrian or navigating around a tire. These overrides can either trigger the next step in the planned route (like turning right after stopping at the intersection) or continue their current waypoint trajectory once they are resolved (like stopping for a pedestrian).

Using Python as the base language, our route inputs are compiled right before execution allowing for easy debugging or making changes to routes on the fly, and continue running without needing to wait. This is one of the biggest time-saving changes we have spearheaded this year. Cutting down on debugging time while testing in the field has immediate benefits of fastening the development timeline. This combined with our new Web UI is used to select routes, view real-time vehicle controls, debugging information from any ROS topics, and also provide an emergency stop button.

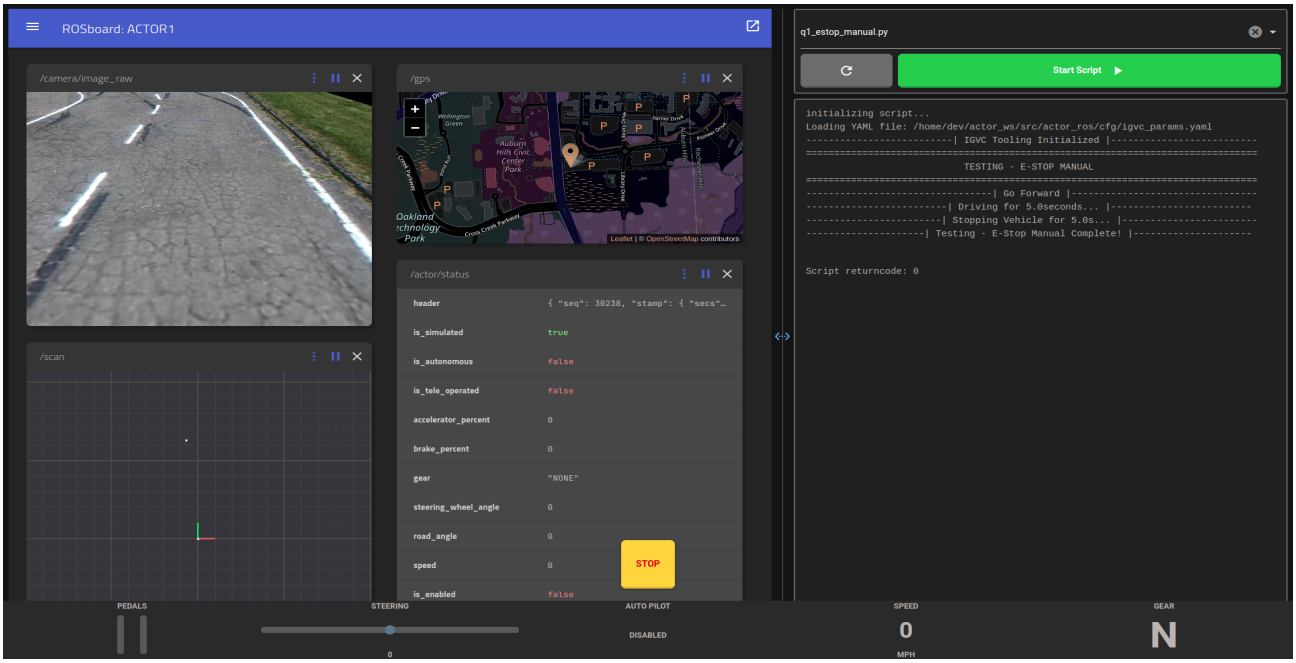


Figure 18: Web interface

The UI (Shown in Figure 18) is designed using NiceGUI, a python library that enables beginner friendly web development; allows any team member to learn and make changes in a short amount of time. Moreover, unlike Angular, Express.js and other commonly used web frameworks, having the entire project in a single Python file allows for easy CI/CD, version control and debugging.

Parking

IGVC involves three specific parking tasks: Pull In, Pull Out, and Parallel Park. The Pull In task requires the ACTor to drive straight down a lane and then smoothly turn into a parking spot from the furthest lane. This is accomplished using distance commands based on GPS heading info. The Pull Out involves the ACTor pulling out of the aforementioned parking spot, turning onto the given outside avenue, and should stop close to a barrel. The movement is achieved using distance commands, but barrel detection is achieved using a 2D LIDAR. The Parallel Parking task requires that the ACTor successfully parallel park without crossing certain boundaries, which represent real-life barriers and objects. This is accomplished using distance commands as well.

Status Display via LED Panel System

As an additional option, we've also developed an outward facing LED Panel System (Figure 19) to display real-time vehicle status information specifically for individuals on the test course. Since the vehicle only seats two, other team members or spectators are often left out on the status reports while the self-drive mode is active. The LED Panel System also helps people outside contribute their perspective to debug errors or correct vehicle behaviors (not immediately apparent to the passengers) at a single glance without needing any connected devices or cellular based telemetry services.

A QuinLED-Dig-Quad (ESP32 based Ethernet LED Controller) enables web-based LED control using the WLED firmware API; also allows scaling panels to any size using flexible WS2812B LED Matrices. These panels directly interface with one of our ROS packages (github.com/Aeolus96/wled_bridge) to display scrolling text or images.

Figure 19: LED Panel in the front

7. Cyber Security Analysis

The NIST Risk Management Framework (RMF) is a 7-step assessment of information security for data-intensive systems and organizations. The RMF consists of preparing an organization for security management, categorizing information based on impact, selecting the appropriate controls to protect the system, implementing said controls, assessing whether the controls are in place, authorizing the system to operate, and monitoring the controls for risk analysis. [12] These 7 steps are designed to support cybersecurity and information security programs, and are necessary to meet requirements for the Federal Information Security Modernization Act (FISMA). As we have implemented new security controls for our vehicle system, we have followed a similar process to that of the NIST RMF. Table 3 shows the most likely attacks, and responses to cyber security threats.

Local Network access is very important and if breached, the bad actor can interact with ACTor vehicle's software and running processes. To prevent this, we use WPA2 security managed by the Wi-Fi router and whitelist MAC addresses of known devices. This combination of LAN defense strategies protects against unknown devices over both wired and wireless connections.

2.4GHz and 5GHz signal jammers can be used to disable our wireless emergency stop as well as our web UI connected to external monitoring devices. For this reason, our entire self-drive hardware stack relies on wired connections to prevent malfunctions in critical vehicle operations. In case, a remote attack does take over control of the vehicle in autonomous mode, once the driver nudges the wheel or pedals, the firmware automatically disengages the DBW. Moreover, we use an active emergency stop system where it is required for a driver to enable self-drive mode physically from inside the vehicle.

Attack	Threat Level	Risk	Defense	Response
Local Network Breach	Medium	ROS Access, Web GUI Access	WPA2 Security, MAC Address Whitelisting	Driver takes control of vehicle, MAC Address Blacklisting
Source Code Tampering/Removal	Low	Unexpected Vehicle Behavior	Secured Device, Controlled Access	Recover code from version controlled online repository

Remote Connection to Main Computer	Medium	Process Tampering, File Tampering	Restricted SSH Access, Password Protection	Recover code, SSH Blacklisting
Remote Connection to E-Stop Monitor	Low	E-Stop Process Tampering	Restricted SSH Access, Password Protection	E-Stop turns on when the monitor is disabled or loses power
Wireless Signal Jamming	Low	Wireless Network Malfunction/Disable	Wired Connection for Critical Devices	Remote E-Stop may not work, Use exclusively wired connections

Table 3: Cyber Security Threats and their responses

8. Analysis of Complete Vehicle

During the design, maintenance and new hardware integration phase, we learned a few interesting things. In previous years, the team was chasing camera hardware with higher frame rates and a global shutter image sensor. This combination does make a lot of sense mathematically however, the color quality and contrast between pavement cracks and the painted lane markings was not constant. Cloud cover as well as time of day drastically affected the performance of lane centering. Using this as inspiration, we chose to try out multiple cameras with technologies like low latency (instead of high frame rate) and high gain HDR (instead of global shutter). One of the successes was the Sony Starvis line of image sensors. The image quality is now superb even in low light conditions where just the headlights of the vehicle are enough to differentiate between the road and the lane markings. Moreover, changing daylight does not affect the lane centering algorithm because the color accuracy and contrast has improved.

Handling hardware failures

Even though our ACTor hardware is built to be robust and survive multiple researchers poking around, we do come across hardware failures that may completely disable the vehicles functionality. Table 4 shows these core functionality breaking failures and how to mitigate them if they occur at competition.

Failure Points	**Resolution**
Camera is not found	Unplug and plug it back in, restart the laptop. Make sure the laptop charger is connected and the AC inverter is powered on
Laptop crashes	Hold the power button for 10s to shut it off. Then make sure the charger is connected and powered on. Turn it back on and re-launch the software
Ethernet connection	Make sure the cable is pushed in. Listen for an audible click as the cable latches in. Ping the IP of the device in question
Software Braking Failure	Restart the DBW and manually activate the E-Stop to check. If it does not work, check if the relay clicks when activated or DBW receives the brake instructions
Raspberry Pi Malfunction	If SSH does not work, remove the pi from the vehicle and diagnose it outside. Worst case, re-flash the SD card using the backup image and validate it again

Table 4: Hardware failures and how to fix them

Software Testing and Version Control

We exclusively use GitHub to keep various branches and commits to our code base. One of the benefits of using modular ROS packages is that each one can be handled by one or two team members enabling fast development outcomes and distributed workload. For tracking changes, we refer to the commit history as well as maintain highly commented code. This allows us to quickly identify why the code does what it does as well using git blame can allow us to directly reach out to the person responsible for that exact line of code. Since our rigorous testing strategy involves simulators as well as real vehicles, small bugs are ironed out instantly and large bugs like vehicle specific configurations can be handled in the field. Our web UI allows real-time access for all team members to monitor the vehicle while it is going around the course; enabling them to validate the functionality they worked on in conjunction with other packages.

After the weekly function testing is concluded, we backup the entire code base, identify the changes required, iron out smaller things like formatting and code comments, and send out a pull request. That is then reviewed by other team members and finally merged into the main branch. We also utilize the latest formatting, type checking and error identifying extensions in VS Code to increase productivity.

System-In-Loop Testing using Simulators

The team completed multiple simulation investigations using SimpleSim, a LTU developed ROS simulation package that supports kinematic models of Ackermann steer or differential steer robots, a pinhole camera model, an ideal LiDAR model, and GPS tracking model. The ground plane is supplied to the simulation environment as an image, and obstructions may be added to simulate approximated LiDAR objects.

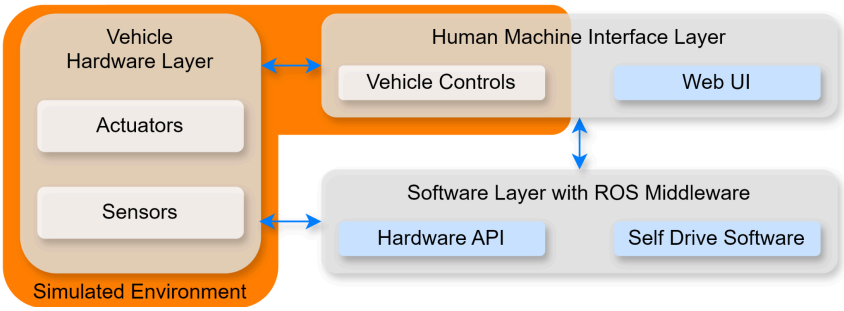


Figure 20: SIL Testing Overview

Obstructions were added to the simulation environment to represent barrels, pedestrians, and stop signs. Multiple software algorithms were tested by simulating the vehicle behavior on the LTU Lot H campus course model, see Figure 20. This was done to minimize the physical testing required to validate the system performance. The virtual environment was designed for seamless integration with the web interface used on the real vehicle. Configuration files were built for variations of the Lot H test course which reflect the IGVC qualifications and functions test requirements.

The software architecture is designed to be modular (refer to the Software Systems section), enabling individual or combined testing of functions. These nodes are subject to thorough integration testing both during development and in-field conditions. Additionally, the vehicle undergoes rigorous testing for each node. This process was conducted on the Parking Lot H Test Course (Figure 21) on the LTU Campus. The vehicle had a mechanical malfunction with its DBW system, after which all nodes were thoroughly tested to ensure their functionality.

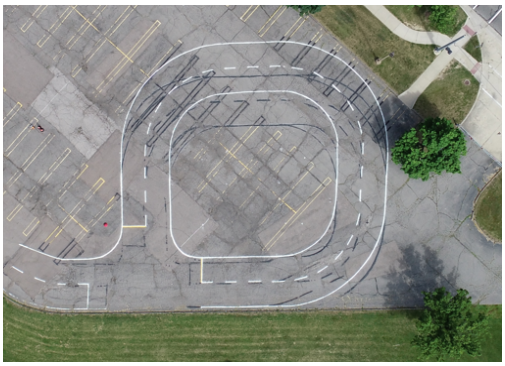


Figure 21: Test course at LTU

9. Initial Performance Assessments

How is ACTor performing to date?

The team is able to test many of the IGVC functions on our test course at LTU. Table 5 shows the status of each qualification, machine vision, traffic sign, intersection, parking, VRU, curved road and other tests.

Qualification Tests **Testing** - Q.1: E-Stop Manual **Testing** - Q.2: E-Stop Wireless **Complete** - Q.3: Lane Keeping (Go Straight) **Complete** - Q.4: Left Turn **Complete**- Q.2: Q.5: Right Turn *Machine Vision Tests* **Complete** - FI.1: White Line Detection **Complete** - FI.2: Static Pedestrian Detection (Vision) **Complete** - FI.3: Tire Detection *Traffic Sign Tests* **Complete**- FII.1: Stop Sign Detection *Intersection Tests* **Complete** - FIII.1: Lane Keeping **Complete** - FIII.2: Left Turn **Complete** - FIII.3: Right Turn	*Parking Tests* **Testing** - FIV.1: Parking. Pull Out **Testing** - FIV.2: Parking. Pull In **Testing** - FIV.3: Parking. Parallel *VRU Tests* **Complete**- FV.1: Unobstructed STATIC pedestrian detection **Testing**- FV.2: Obstructed DYNAMIC pedestrian detection **Complete**- FV.3: STATIC pedestrian detection. Lane changing **Complete** - FV.4: Obstacle detection. Lane change *Curve Road Tests* **Complete** - FVI.1: Lane Keeping **Complete**- FVI.2: Lane Changing *Other Tests* **Complete** - FVII.1: Pothole Detection **Complete** - FVII.2: Merging *Simple Main* **Testing**- Simple Main

Table 5: Status of each IGVC function on design report submission date. (Planning, Developing, Testing, Complete)

10. Unit Testing Results

The mandatory unit tests have been tested at the LTU test course. The following details the results of those tests:

Emergency Stop

Figure 22: Wired Emergency Stop Unit Test

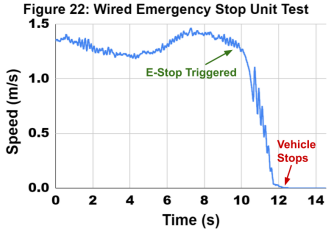
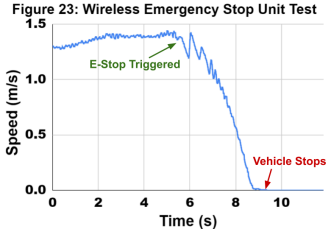


Figure 23: Wireless Emergency Stop Unit Test



Object Detection

Figure 24: Object Detection (Speed vs Time)

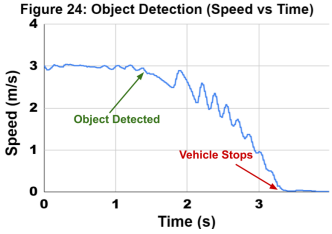
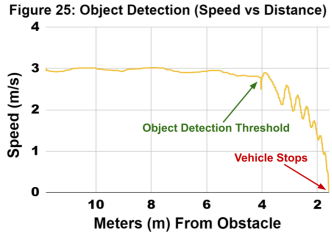


Figure 25: Object Detection (Speed vs Distance)



Speed Limit Tests

Figure 26: 3 mph (1.34 mps) Speed Limit Test

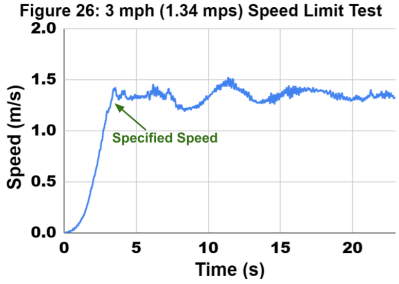


Figure 27: 5.2 mph (2.28 mps) Speed Limit Test

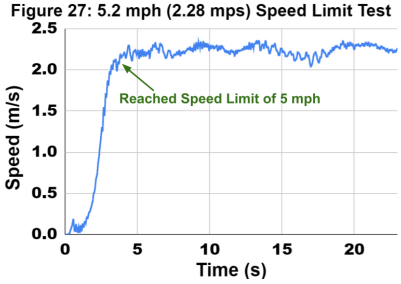
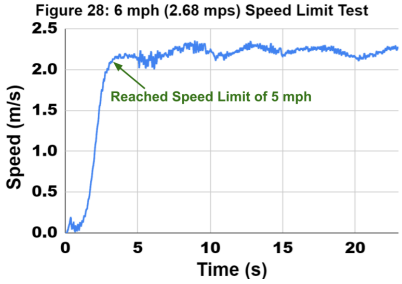


Figure 28: 6 mph (2.68 mps) Speed Limit Test



Lane Boundary Crossed

Figure 29: Right Lane Boundary Crossed

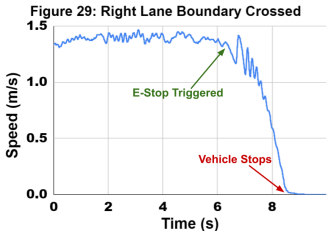


Figure 30: Right Lane Boundary Crossed Lateral

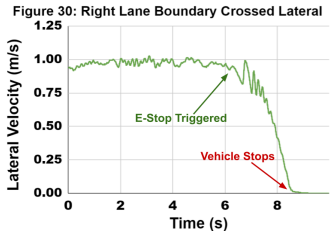


Figure 31: Left Lane Boundary Crossed

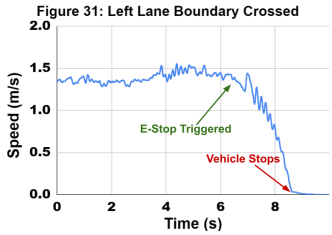
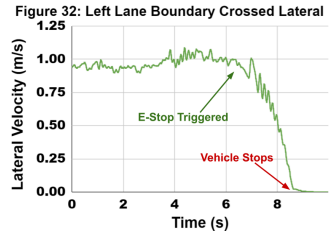


Figure 32: Left Lane Boundary Crossed Lateral



Reversing Speed Limit

Figure 33: 1 mph Reverse Speed Limit Unit Test

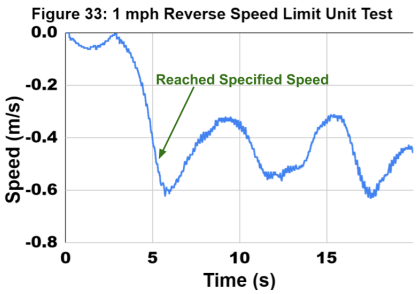


Figure 34: 2 mph Reverse Speed Limit Unit Test

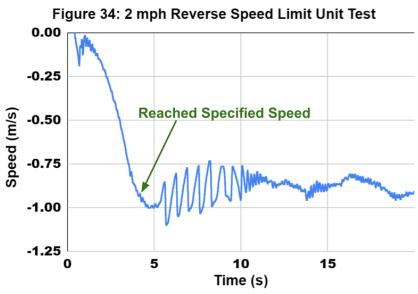
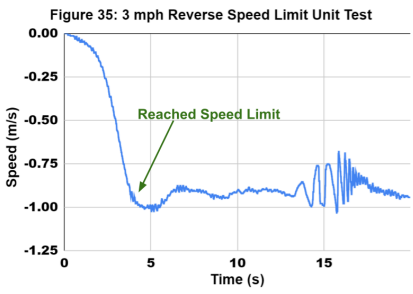


Figure 35: 3 mph Reverse Speed Limit Unit Test



References

[1] IGVC 2023 Self-Drive Design Report, http://www.igvc.org/design/2023/17.pdf (accessed 02-24-24)

[2] Mako g-319, accessed 05-15-31, https://www.edmundoptics.com/p/allied-vision-mako-g-319-1-18-inch-color-cmos-camera/33094

[3] 1st vision 1" 2 to 3 megapixel oem lens series, https://www.1stvision.com/lens/spec/1stVision/LE-MV3-0618-1, accessed 5-13-21

[4] Velodyne puck, https://velodynelidar.com/products/puck/, accessed 05-15-2023

[5] URG-04LX-UG01, https://hokuyo-usa.com/products/lidar-obstacle-detection/urg-04lx-ug01, accessed 05-15-23

[6] Swift navigation piksi multi gnss, https://www.swiftnav.com/piksi-multi, accessed 05-15-2023

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in ICRA workshop on open source software, vol.3, no. 3.2. Kobe, 2009, p. 5.

[8] Paul, N., Pleune, M., Chung, C., Faulkner, C., Warrick, B., Bleicher, S., A Practical, Modular, and Adaptable Autonomous Vehicle Research Platform, IEEE International Conference on Electro Information Technology 2018

[9] Mitchell Pleune, Nicholas Paul, Charles Faulkner, C. J. Chung, Specifying Route Behaviors of Self-Driving Vehicles in ROS Using Lua Scripting Language with Web Interface, 2020 IEEE International Conference on Electro/Information Technology

[10] Redmon, Joseph et al. "YOLOv3: An Incremental Improvement". arXiv. (2018)

[11] Kocherovsky, M., DeRose, G., Paul, N., Timmis, I., & Chung, C. J. (2024). Autonomous Vehicle Steering through Convolutional and Recurrent Deep Learning. In *Autonomous Vehicles and Systems* (pp. 83-111). River Publishers.

[12] NIST Risk Management Framework, https://csrc.nist.gov/Projects/risk-management, accessed May 14, 2024