ACTor: IGVC 2022 Self-Drive Design Report

Lawrence Technological University

Team Captain:	Giuseppe “Joe” DeRose	gderose@ltu.edu
Team Members:	Joseph Schulte	jschulte@ltu.edu
	Mark Kocherovsky	mkocherov@ltu.edu
	Adilur Choudhury	achoudhur@ltu.edu
Mentors	James Golding	jgolding@ltu.edu
	Justin Dombecki	jdombecki@ltu.edu
	Mitchell Pleune	mpleune@ltu.edu
Faculty Advisors	Nicholas Paul	npaul@ltu.edu
	ChanJin “CJ” Chung	cchung@ltu.edu

Faculty Advisor Statement

I, CJ Chung & Nicholas Paul, 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.

ChanJin Chung Nicholas Paul

Date: May 15, 2022

1. Introduction and Overview

A Polaris GEM e2 based Autonomous Campus Transport (ACTor) is an autonomous vehicle research platform developed by university students to research and develop new technologies for autonomous systems. The vehicle is equipped with Dataspeed's throttle-by-wire, brake-by-wire, steer-by-wire, and shift-by-wire system. Computers include an Intel NUC, a primary computer with GPU, and Raspberry PI 3s. The sensor suite used for perception consists of a Velodyne 16-Beam 3D LIDAR, Hokuyo URG 2D LIDAR, Piksi multi real-time kinematics GNSS, and front-facing Mako PoE camera. Some modules from the Robotic Tool Kit (RTK) software, provided by the US Army Ground Vehicle Systems Center (GVSC) were adapted to provide basic sensing capabilities. The ACTor vehicle is capable of performing IGVC self-drive tasks such as lane following, obstacle avoidance, waypoint navigation, and object detection. This report describes the autonomous system designed and the methodologies used for the system development and integration.

Major hardware and software changes from 2021 IGVC system [1] include: additional sensing with Hokuyo 2D LIDAR, updated GPS capabilities and improved YOLO models for object detection. In addition to hardware and software changes, the team increased their emphasis on virtual methods. In particular, the simulation capabilities first used in 2021 were extended with more capability and used to model a majority of the IGVC competition tasks.

2. Design Process

Methodology Based on Agile Development Concepts

Our team adopted the concept of agile development methodology to allow rapid adaptation to changing user requirements, focus on the most important tasks from the user's point of view, and promote rapid delivery of preliminary results and early detection of errors. The concept of the Agile methodology is shown in Figure 1. We focused on the following six principles out of the famous twelve agile principles for our development:

(1) Our highest priority is to meet user requirements through early and continuous delivery of working software, incrementally. (2) We are always ready to adapt to the changes of requirements. (3) Deliver products frequently with a shorter timescale. (4) Face-to-face interaction is the most efficient and effective method of conveying information within a development team. (5) Working released software is the primary measure of progress. (6) Simplicity, the art of maximizing the amount of work not done, is essential.

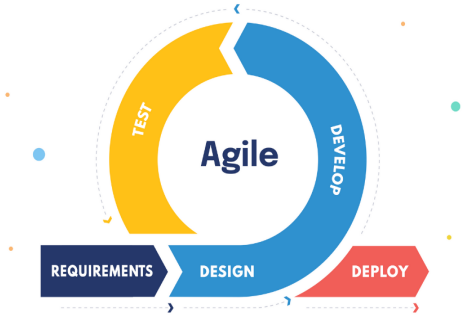

Figure 1: Agile development methodology

Team Organization

The team meets in-person and/or via Zoom at least once a week to set goals and discuss new innovations and technologies to develop for the vehicle using the agile development concept. These meetings also provide a way to collaborate in creating new ideas and divide up tasks for the upcoming week. Each member was delegated to specific tasks by comfortability and interest. The team is composed of computer science students (see table 1). Each student puts around 5 to 10 hours a week coding, testing, or in meetings to accomplish the tasks.

Name	Degree Program	Primary Responsibilities
Giuseppe “Joe” DeRose	M.S. Computer Science	Drive By Wire System Camera sensing and processing Lane Centering and object avoidance
Joseph Schulte	M. S. Computer Science	LiDAR, eStop; Sign, Pedestrian, Pothole, and Tire detection
Mark Kocherovsky	M.S. Computer Science	GPS, Parking, Web API
Adilur Choudhury	B.S. Computer Science	New team member; assisting team members

Table 1: Team members and roles

3. MECHANICAL SYSTEMS

ACTor is built on top of a modified Polaris Gem e2 provided by a joint sponsorship from MOBIS, DENSO, Realtime Technologies, and SoarTech. The base vehicle was provided by MOBIS and the drive by wire system was provided by Dataspeed Inc. The drive by wire system, outlined below, supports fully autonomous operation of the vehicle.

To ensure robustness of connections and components, computer and power delivery systems are mounted securely under the seats where they are difficult to kick. The camera is mounted with a 1/8in aluminum bracket kept as short as possible to provide sufficient rigidity. Components on the top of the vehicle are mounted to an 80/20 frame, and are both rigid and easy to remove for safe storage. Our VLP-16 lidar is mounted to an adjustable camera tripod to let us easily aim it. All components mounted outside the vehicle are waterproof and resistant to damage by rain.

Vehicle Abilities

The Polaris Gem 2 has a top speed of twenty miles per hour, and a range of approximately twenty miles. All sensors mounted outside the vehicle are weatherproof, and the doors and body of the Gem2 are sealed. The vehicle has been tested to go up a 30% grade without difficulty, and the suspension has been kept stock to the Gem 2 design.

X-By-Wire

The Polaris Gem 2 vehicle was updated with Dataspeed drive-by-wire hardware and the Dataspeed ADAS Development Vehicle Kit software. The software kit enables the vehicle to be driven using native or controlled interfaces with ROS. For example, the user may drive the vehicle using native accelerator/brake pedal, steering wheel angle/torque and gear selection commands. Or, another option is to use a Dataspeed provided Universal Lateral/Longitudinal Controller (ULC). The ULC option allows users to provide linear and angular velocity targets for the vehicle and the controller modulates the accelerator/brake, steering and gear selection systems to achieve the desired response. ULC also provides a number of parameters to adjust the characteristics of the vehicle behavior, for example, targets for linear and angular acceleration for the controller behavior. Currently, our team is using the Dataspeed default parameters which targets a velocity dependent linear acceleration limit (0.9 - 1.2 m/s^2 for our operating conditions) and a constant deceleration target of 1.5 m/s^2. An example of the linear velocity control for the GEM vehicle is provided later in this section.

The primary ROS interface to the Dataspeed ULC is a geometery_msgs/twist message with additional parameters for modifying various control parameters. This simple interface allowed the team to easily integrate the ULC into the vehicle environment. Once the integration was completed, a series of twist commands were used to direct the vehicle acceleration, deceleration, yaw rate and gear selection (Forward or Drive). Figure 2 demonstrates the linear velocity control for the GEM vehicle using the default

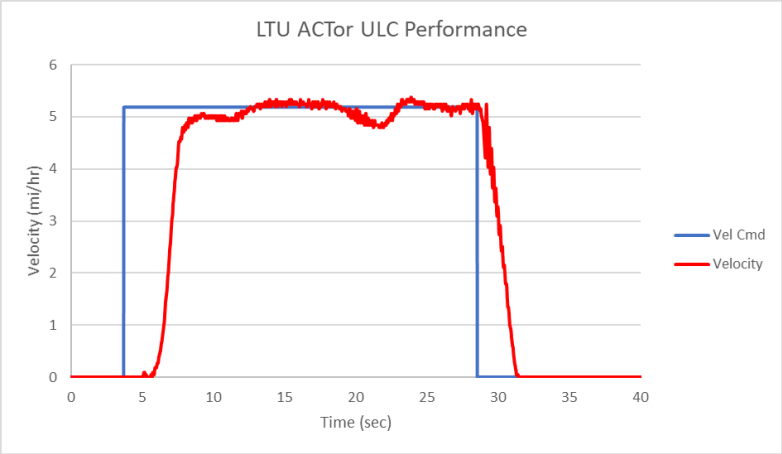

Figure 2: Dataspeed ULC linear velocity control example

acceleration limits for a 5.2 mi/hr Unit Speed Limit test. Our team has found that that use of tight and loose tracking results in smooth acceleration behavior with very little overshoot.

4. ELECTRICAL SYSTEMS

Computing Components

The main computer is an Intel NUC running Linux Ubuntu. This computer is responsible for coordinating all of the other components of the vehicle, and hosts our route scripting tools. It is also responsible for object detection and sensor fusion to use the camera and LiDAR for high-accuracy real-time detection and 3D positioning of pedestrians and obstacles. There is a secondary computer with a powerful desktop CPU and GPU (Ryzen R7-2700, GeForce GTX 1070 Ti) meant for heavier deep-learning models, such as our alternative lane-following system. Most of the computation happens on the Intel NUC. Raspberry PI 3s are used for hardware e-stop, remote e-stop, safety and status lights, and an LED panel.

Sensory Components

The system primarily uses a single Allied Vision Mako G-319C [2] power-over-Ethernet camera; it provides high image quality and enough field of view to capture the lanes while retaining enough resolution to detect signs. The 6mm 1stVision LE-MV3-0618-1 lens [3] at 1.8 full stops provides a 50 degrees field of view.

The car is equipped with a Velodyne VLP-16 "Puck" LIDAR [4] donated by Veoneer. The Puck is a small, compact 16 channel lidar, weighing less than two pounds. Its capture range is 360 degrees horizontally and 15 degrees vertically with a radius of 100 meters. It is able to output 300,000 points per second across its 16 channels. These data points represent points in space some distance from the lidar source. It also has a Hokuyo URG-04LX-UG01 [5] mounted on the front. The URG is a small two-dimensional lidar with four meters of range.

The vehicle utilizes a Piksi Multi GNSS Module [6] to gather GPS and compass data for navigation. The module has centimeter-accurate positioning and fast update times. The fast and reliable data makes it well suited for a moving vehicle, where great distances will be covered over a small amount of time. The GPS module is used alongside a second Piksi used as a base station providing Real Time Kinematic (RTK), resulting in more accurate GPS information. Additional GPS Rover on the vehicle was installed to enable vehicle heading information.

Connecting Components

All systems of the car have been connected through either Ethernet or CAN buses instead of USB, as seen in Figure 3. Ethernet is preferable because of the superior robustness that it provides over USB. The main exception to this is the connection to the drive-by-wire (DBW) system, which is reliant on a single USB connection. This vehicle will automatically come to an emergency stop if it detects a malfunction of this connection.

The Vehicles E-Stop system is split into two parts. The first part is a closed hardware loop that runs through all the E-Stop buttons. This loop starts and terminates at a Raspberry Pi. Should any of the buttons be hit, the circuit will be open and the pin on the Pi will read low. The pi will then send a signal to the Intel Nuc through ethernet. The NUC will then safely stop the vehicle.

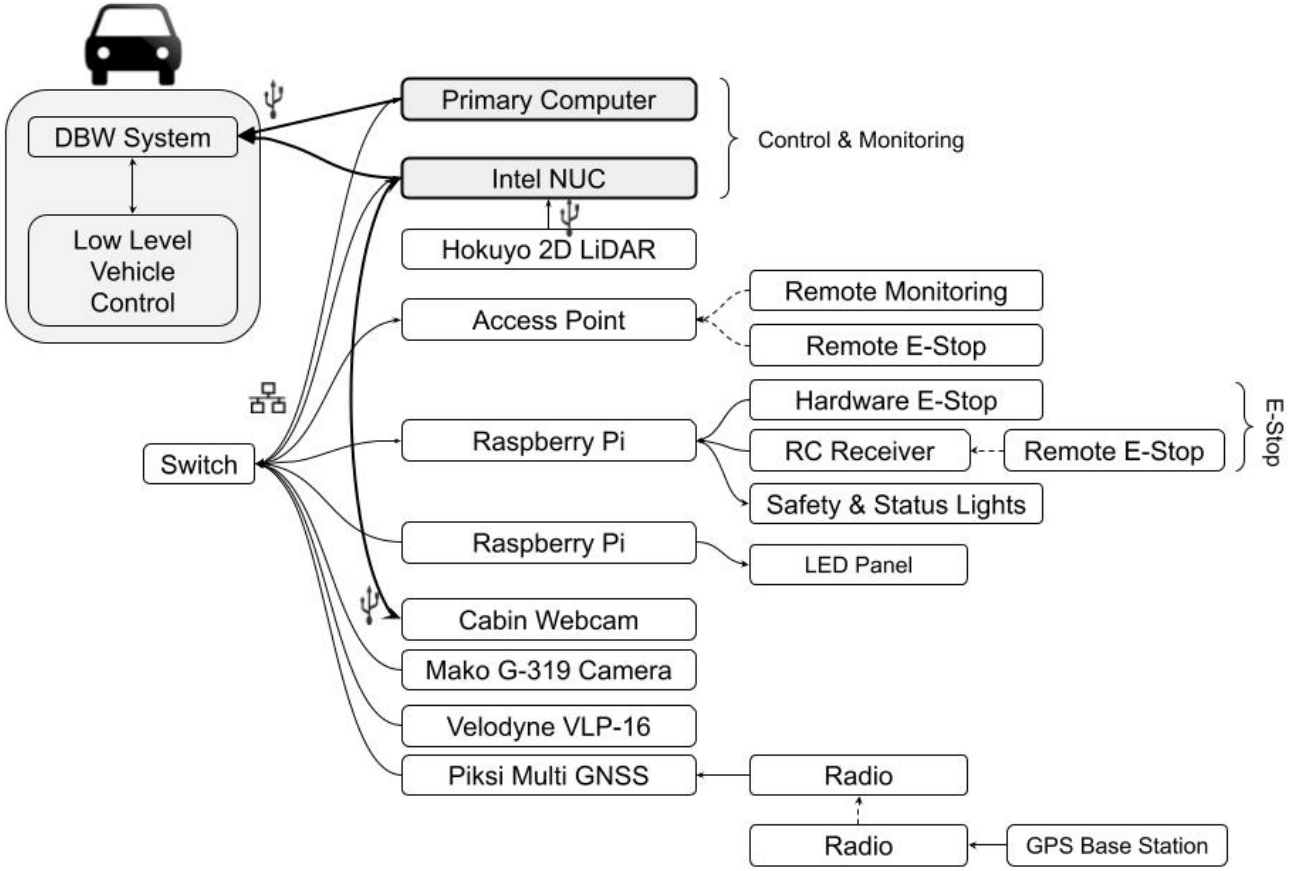

Figure 3: All sensors except cabin camera and control nodes are connected to a local network through Ethernet via a switch or CAN via a CAN Bus. Solid connections indicate physical cable connections. Dashed connections indicate wireless communication.

Power Connections

All electrical energy on the vehicle is supplied by four deep cycle absorbent glass mat batteries connected in series. A single 12V connection is taken from one battery to supply power to the control systems. All low power 12V devices are run off of a large power delivery panel, which can individually enable/disable connections through a touch screen. A 1000W inverter is also mounted and connected directly to the 12V supply connection for easy charging of laptop batteries. This is primarily for ease of use, and is very useful while developing. The switch has four power over ethernet connections, currently used to supply power to the E-Stop computer (Raspberry Pi), and vision camera (Mako G-319C). The described power connections are shown in Figure 4.

The majority of energy is spent accelerating the vehicle, and relatively little power is spent powering control components. A full charge will take six to eight hours, and can travel twenty miles. The inverter is >90% efficient, and represents a low power loss compared to the GPU computer.

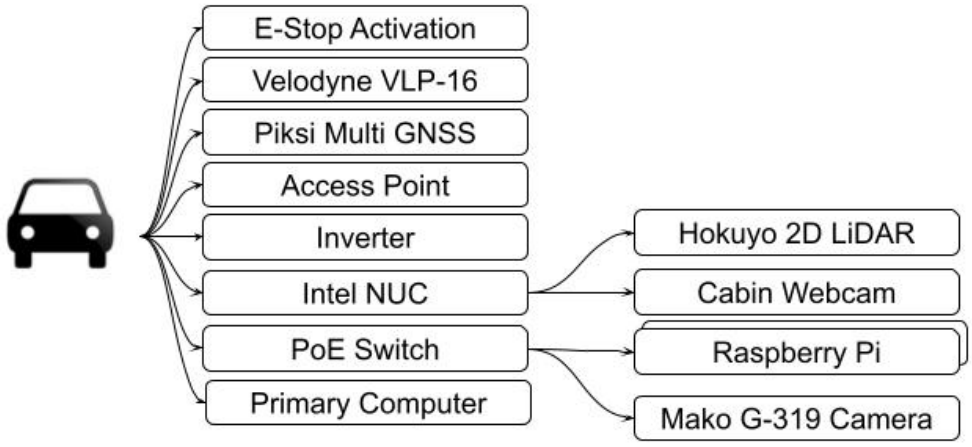

Figure 4: Vehicle, sensor, and components power distribution

Cost of Hardware Components

Item	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
Intel NUC Mini PC kit NUC7i5BNH Core i5	$543.00
Additional NUC (as a backup)	$543.00
Velodyne VLP-16 “PUCK” 3D LiDAR, 16 beams	$7,999.00
Hokuyo URG-04LX-UG01 LiDAR	$975
Swift GPS, Piksi Multi GNSS	$1,644.56
Additional rover module and antenna to get GPS heading info	$896.00
Mako PoE Camera	$1,031.00
Main Computer with GPU	$1,800.00
Miscellaneous items including Lenses, wireless e-stop, LED lights, on-board touch screen monitor, bluetooth keyboard, cabin camera, RPIs, inverters, and LED panel system	$2,300.00
Total	$67,731.56

Table 2: Estimated cost of ACTor vehicle

5. SOFTWARE SYSTEMS

Requirements and Design Goals

ACTor's software is designed and implemented with several requirements in mind. First, the software should follow the design principles set in place by Robot Operating System (ROS). That is, the software should be as distributed and modular as possible [7]. Second, the software should be able to be developed and modified

quickly. Since ACTor is primarily a research vehicle, implementing new ideas or research projects should be simple. Finally, the software should be built in such a way that its inputs and outputs are interchangeable. This enables the software to be quickly tested and allows for smooth implementation of new hardware and software. In addition, we introduce the executable RTK in an independent Docker container using different Linux OS and ROS versions to integrate with other ACTor packages and external nodes using Linux 20.04 and ROS Noetic.

Architecture and Design Strategies

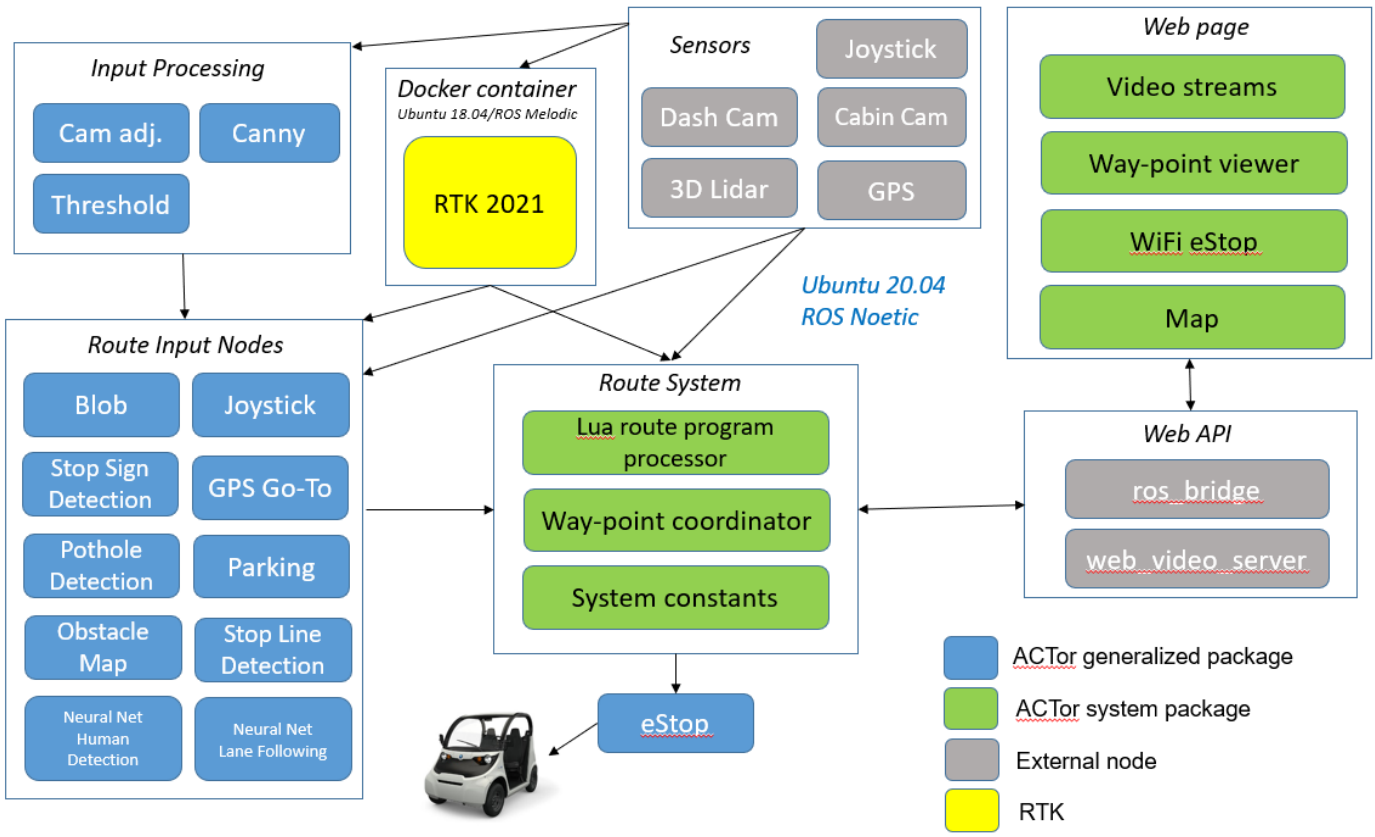

Figure 5: Basic data flow through core packages

The system is divided into several independent but connected packages: sensors, input processing, route input, route system, web API, and RTK in a Docker container. Figure 5 shows a high level overview of what is contained within each of these packages.

Sensors

The sensors package is a collection of sensor driver publisher nodes and configuration files. Each sensor, such as the GPS, lidar, or camera has its own node(s) that are responsible for converting data into usable formats and publishing information onto the ROS network. The system provides abstraction of data by placing the responsibility of reading and converting the data in the input packages' hands, which provide clean data structures to the nodes that control the vehicle's logic. By separating the sensor package into multiple nodes, the system becomes very maintainable, giving the ability to add and remove sensors easily. We have also installed a Hokuyo URG two-dimensional LIDAR that provides a LaserScan of the detected area, which is used instead of our Velodyne LIDAR for certain tasks where the Velodyne unit would not be able to make the necessary detections.

Input Processing

The input processing package cleans and prepares the data for the route input package. Currently the nodes within this package are designed to take the camera data received from the camera node. The package is responsible for applying white balance, gaussian blur, and other OpenCV algorithms to make the camera data usable for the route input package.

Route Input

The route input package takes the sensor data either directly from the sensors or from the image processing package and transforms it into usable data to be fed into our route system. The route input package is important as it provides as much information to the route system as possible, but allows the route system to subscribe and unsubscribe from each node. Each node within the route input package will only process sensor data if the route system is subscribed to it. Many of the nodes are computationally heavy so it is important to make sure they are running while the route system needs them.

The route input package contains our lane centering algorithm ("blob"), obstacle maps for avoidance and emergency monitoring, stop sign, one way sign, stop line, and pothole detection algorithms, as well as GPS follow and parking maneuvers. Each of these nodes can be combined together to accomplish the necessary tasks.

Route System & Lua Scripting

The vehicle is entered into IGVC Spec2/Self-Drive each year, and in order to make competition go smoothly, we decided to try to remove the long compile times when tweaking our code. To do this, we structured our navigation around a script parser, enabling us to make most tweaks only to the scripts instead of the C++ code.

The "Router" node outputs a Twist movement command. It will either elect to forward a twist topic from another specialized navigation node or send a chosen constant topic. The router continuously runs a selected Lua script that contains all the logic of the navigation logic to make these decisions. This script has Lua functions that read/publish ROS std_msgs, specify what topic to forward, decide what constant twist command to send, activate the e-stop, get the local obstacles, check the current waypoint target, talk to RTK components [12]. Figure 6 shows how the Route Server is implemented and how Lua scripts are executed. Our system is centered around the router, its primary goal is to feed the route script as much data as possible.

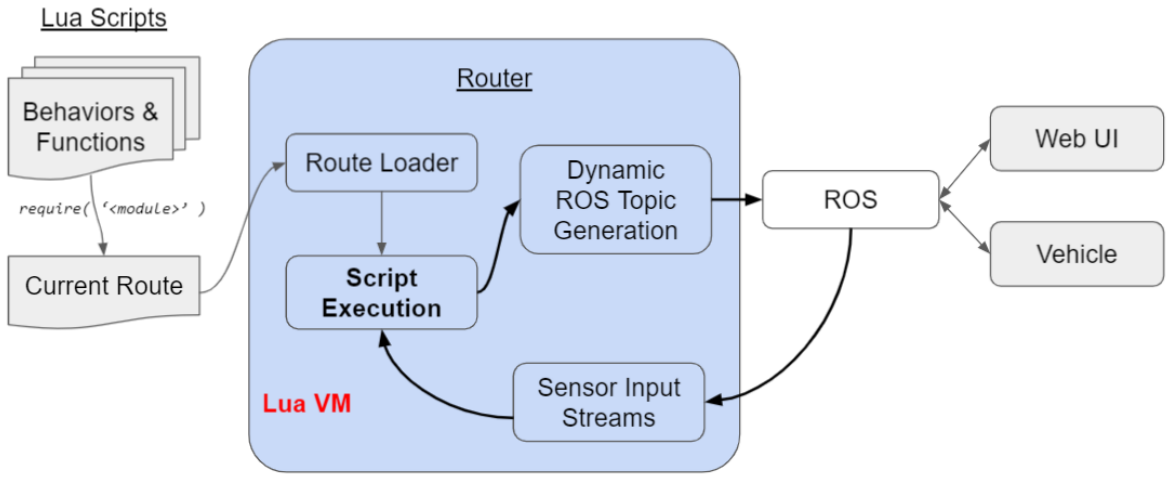

Figure 6: Implementation of Router Server

Web API

The ACTor vehicle receives routing data and displays useful diagnostic information through a custom-designed webpage hosted on the main computer. The website is written in JavaScript using the React and Node frameworks along with Bootstrap CSS. By using these frameworks, new components can be added to the server without having to modify the entire page or write complex css. The website displays a live feed of the "blob" node's output, allowing viewers to see where the lane centering algorithm is trying to steer the car. The route system takes input from the web server.

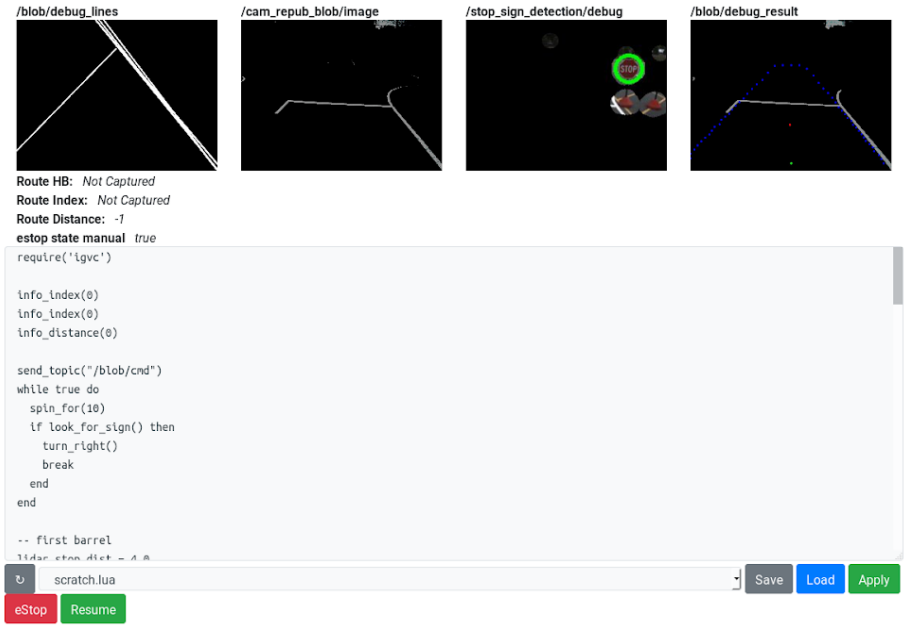

Figure 7: Web interface example

The input consists of an editable text field containing the current route the car is going to take. This allows rapid development of new routes and the ability to easily edit older routes. Figure 7 shows an example of the web interface.

Robotics Technology Kernel (RTK)

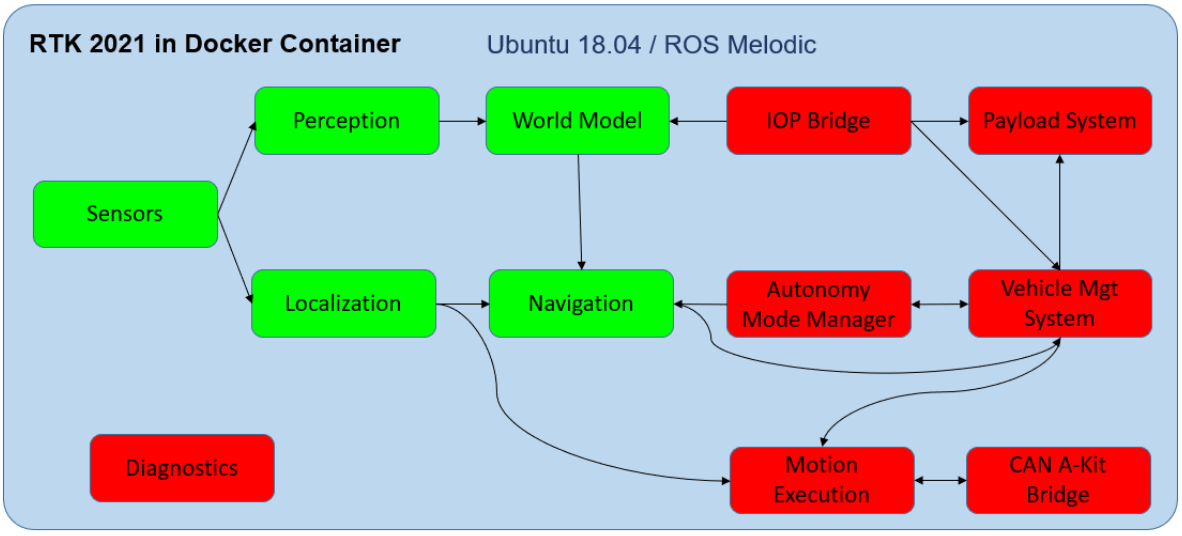

Figure 8: Packages of the RTK system in a Docker Container

In order to integrate RTK 21 that uses old Ubuntu 18.04 and Melodic ROS with our computers using newer Ubuntu 20.4 and Noetic ROS, we introduced an innovative concept of using Docker containers. RTK packages in a Docker container shown in green in Figure 8 are designed to utilize to get sensory information ACTor's Route Input nodes.

Lane Following

Lane detection and centering is combined into a single algorithm nicknamed, "blob," uses OpenCV to process color images to find lane boundaries. The algorithm begins by running one of a few methods of edge detection on the image, which are interchangeable at run-time. These include Canny edge detection (grayscale or color), adaptive threshold, and the Sobel operator. Most of the time the Canny (color) method is used. Next, a Hough transform is run on the edges to detect lines. Only lines within forty-five degrees of vertical are accepted. This is done to avoid detecting horizontal edges from stop lines and zebra-stripes. These lines are then extended and drawn on their own image for the final "blob" processing.

Lastly, a point C_V (see Figure 9) is chosen to be just above the center of the front bumper. Twenty to one-hundred probe lines are sent out in a fan at even intervals between left and right above horizontal. When these probes find a pixel that has been filled by a Hough line, the distance and angle are recorded. If a probe does not find a line, the edge of the image is used. Each probe is then modeled as a spring to push or pull on the initial point toward the center of the lane C_L. The horizontal component of this force is used as steering input for the vehicle. The nominal distance and force of the modeled springs is tuned for the vehicle.

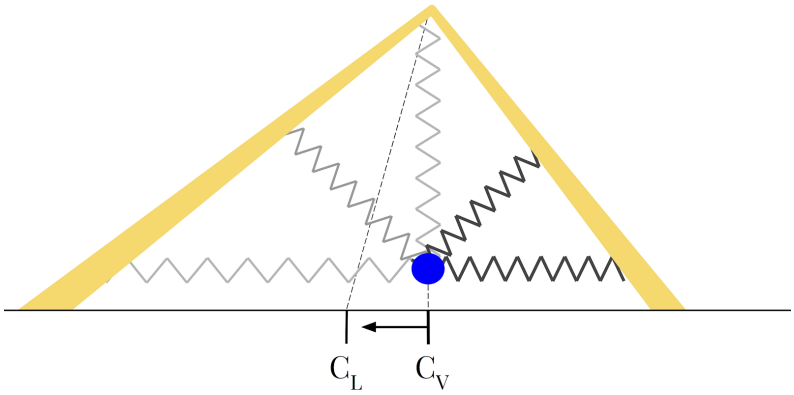

Figure 9: The "blob" algorithm uses simulated springs to push the vehicle's center C_V toward the center of the lane C_L. Thicker lines represent more compression and therefore more influence on the direction the point will move.

Obstacle Detection and Resolution

The obstacle avoidance package currently uses input from the VLP-16. Using functionality built into the TARDEC RTK system, ground and obstacle pointclouds are generated from the VLP-16's input.

The current implementation of the obstacle avoidance algorithm checks regions defined by parameters as shown in Figure 10. These regions are published to the route system, which determines the action to be taken. 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 scenario. The obstacle detection node allows for detection of objects in arbitrarily defined spaces. This allows different events to occur based on which detection region the object is in.

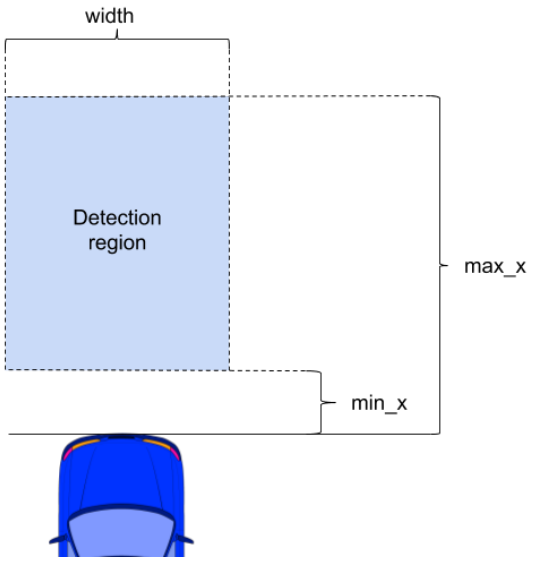

Figure 10: How to define detection region

Pedestrian Detection

For human detection, we use an efficient pre-trained deep neural network architecture called YOLO [13] which is freely available based on common datasets. Using this model, it can quickly and accurately identify any person in its vision, and we fuse this information with LiDAR data for highly accurate 3D positioning. This is sufficient to determine their location and enable simple interactions, like stopping and avoidance. We primarily use YOLO v5 Nano, a version of the architecture small enough to run on the NUC's low-power CPU but powerful enough for reliable, stable, and accurate detection. Figure 11 shows the detection of people on a test course.

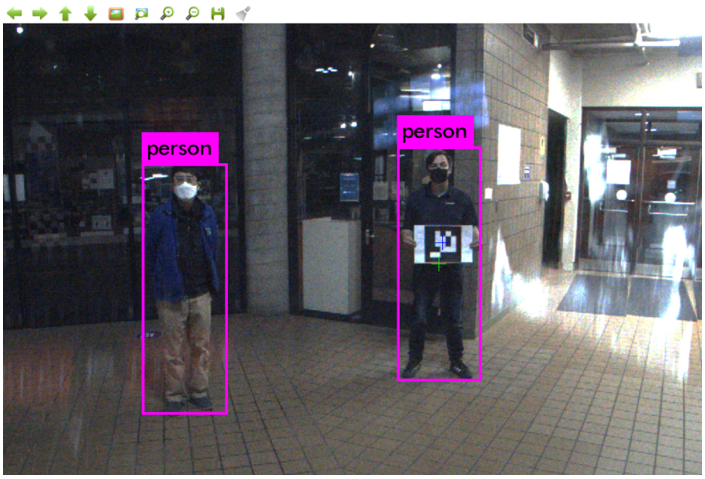

Figure 11: An example of detecting people

Sign Detection

The sign detection algorithm uses color filtering along with HAAR classifiers in order to detect signs based on shape and coloring. Thousands of images were used to train the HAAR classifiers used. The algorithm allows for fast recognition of signs in varying environments. Figure 12 shows examples of detecting a stop sign on campus.

Figure 12: An example of detecting a stop sign

Pothole and Tire Detection

Pothole detection is accomplished through color filtering to isolate the pothole in the frame. The pothole – and everything else within the color range – is depicted as a white mask, and the rest of the image is shown as a black background. If the mask covers a sufficient portion of the frame region, then the ACTor will execute an avoidance maneuver.

Tire detection is accomplished with a small forward-facing 2D LiDAR (See Figure 13) mounted low to the ground. The low visual contrast and low profile of a tire relative to the ground make it difficult to detect through vision or the top-mounted 3D LiDAR, but front and rear low 2D LiDARs that scan parallel to the ground are extremely effective at detecting such obstacles.

Figure 13. 2D LIDAR

GPS updates

Our Piksi Multi GNSS has two antennas used for higher precision. Our ROS nodes return coordinate (latitude and longitude) info in Earth Centered Earth Fixed format, together with Inertial Measurement Units and Heading Info (Northeast Down). Our system also has an optional base station that can be used for better coordinate certainty. We use GPS data for distance calculations and route fixtures. For example, the merging task uses a waypoint to know when to begin the turn onto the correct lane.

Parking

IGVC requires three parking tasks: Pull In, Pull Out, and Parallel Park. The Pull In task demands that the ACTor drive straight down an avenue and neatly turn into a parking spot from the furthest lane. This is accomplished using distance commands based on GPS heading info. The Pull Out task demands that the ACTor reverse out of the aforementioned parking spot, turn onto the given avenue, and reverse until close to a barrel. The movement is achieved using distance commands, but barrel detection is achieved using a backwards-facing 2D LIDAR (update). The Parallel Parking task requires that the ACTor successfully parallel park without crossing certain boundaries, which represent real-life barriers and objects. This too is accomplished using distance commands.

6. LED Panel System

The LED Panel System is called ROS River which will display information about the ROS system to the user via a LED display (See Figure 14). ROS River runs on a Raspberry PI 3B and is connected via ethernet to the main ROS core. The display used for the project can be scaled to any size using the WS2812B LEDs. Once powered, ROS River will begin to run the application which subscribes to certain ROS topics to determine what to display. One such topic is “display/text” which when published will display the data directly. The application also has the capability to run in an “auto” mode which will determine what to display based on information from the topic “rosout”. This mode is useful to determine errors within the ROS core.

The purpose of ROS core is to communicate information about the vehicle’s current state/objective to individuals outside the vehicle itself. With the information displayed over the ROS River system, individuals will be able to determine the current state of the vehicle to ensure understanding of objectives and issues. Without the display only the individuals inside the vehicle would know of issues and the current tasks.

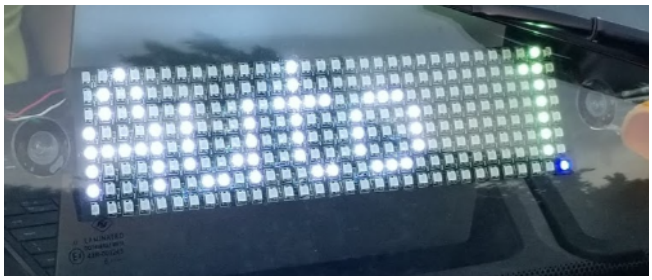

Figure 14: LED Panel system “River”

7. SAFETY

Safety Considerations

There are several safety measures taken into account with ACTor’s hardware and software revolving around an external “emergency monitor” (EM). The EM is a Raspberry Pi connected via the ROS network. The ACTor software is unable to directly control the vehicle and must do so via the emergency monitor (see Figure 15). The software sends a motion command to the EM which validates max speed, max turn radius, etc. and then forwards the command to the vehicle via the “host” node. Along with motion validation, the EM also monitors the lidar node and E-Stop subsystem for emergency signals.

If a signal is received, it issues a stop command immediately.

Additional precautions were taken to prevent EM and E-Stop failures. If the EM loses power, is disconnected from the network, or sends invalid data, the host node will send a blocking stop command within 200ms. Since the E-Stop requires an active external hardware signal, any malfunction of the E-Stop subsystem will also issue a stop command within 200ms. A safety light is also attached to the vehicle. The safety light will flash whenever the vehicle is in autonomous mode.

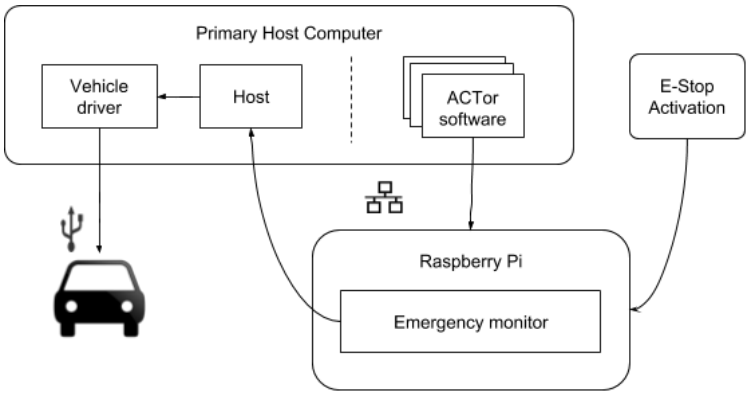

Figure 15: Emergency Monitor for Safety

eStop Performance

There is minimal latency between the beginning of an eStop event to the stopping of the vehicle. Figure 16 right shows the vehicle speed during an eStop event. In this example, the vehicle was traveling at 5 mi/hr and the system was given a configurable target deceleration of 1.5 m/s^2 deceleration. Please note that due to delays in the closed looped system, the average deceleration is approximately 0.75 m/s^2.

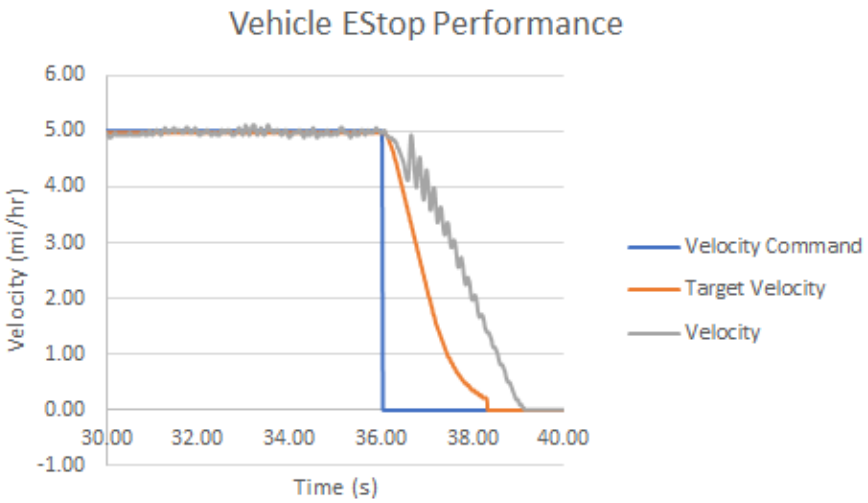

Figure 16 - The vehicle speed during an eStop event. The eStop was triggered at 36 seconds. The vehicle came to rest at approximately 39 seconds.

Failure Points

The following table 3 summarizes each failure point with risk level and description how to resolve.

Failure Point	Type	Risk	Resolution
Loss of Power	Hardware	Very Low	Autonomous mode is automatically deactivated and safety driver takes control. Primary computer (powered by battery) reports error.
Switch or Network Malfunction	Hardware	Low	Primary computer is unable to contact external safety monitor and issues an E-Stop command within 200ms.
Inverter Malfunction	Hardware	Low	Primary computer is unable to contact the external safety monitor due to loss of power and issues an E-Stop command within 200ms.
Raspberry Pi (Safety Monitor) malfunction	Hardware	Low	Primary computer detect irregularity in external safety monitor and issues E-Stop command within 200ms.
E-Stop malfunction	Hardware	Very Low	E-stop requires an active signal, if interrupted, the vehicle executes stop command within 200ms.
Camera Malfunction	Hardware	Low	Computer displays disconnection error. If needed, driver may E-Stop the vehicle.

Lidar Malfunction	Hardware	Low	Computer displays disconnection error. If needed, driver may E-Stop the vehicle.
GPS Malfunction	Hardware	Low	Computer displays disconnection error. If needed, driver may E-Stop the vehicle.
Camera is unable to determine lane lines	Software	Medium	Verify camera calibration before runs
A ROS node crashes	Software	Medium	Depending on which node crashes one of two things will happen: 1. Non-critical: The node is automatically relaunched by the system 2. Critical: The primary computer issues a stop command within 200ms
Invalid actions or routes are received.	Software	Low	Navigation immediately enters a paused state and halts route execution.

Table 3: Failure points and modes

8. SIMULATION AND PROTOTYPING

For this year, the team extended their two-dimensional simulation capability to enhance development and testing of the vehicle control algorithms. The simulation package, called Gazelle Sim, was developed to execute as a ROS node that may replace the vehicle to promote the development of sensing and control strategies in a virtual environment. In addition, the "2-dimensional" implementation requires low computational power when compared to 3-dimensional simulation environments. The Gazelle Sim package supports kinematic models of akermann steer or differential steer robots, a pinhole camera model, an ideal lidar model and simulated GPS tracking. The ground plane is supplied to the simulation environment as an image and circular and rectangular obstructions may be added to simulate lidar directions. Figure 17 is a snapshot of Gazelle Sim being used to test an sensing and control algorithm to complete the IGVC events.

As in previous years, virtual simulation capability assisted in minimizing the amount of physical testing required to validate the system performance.

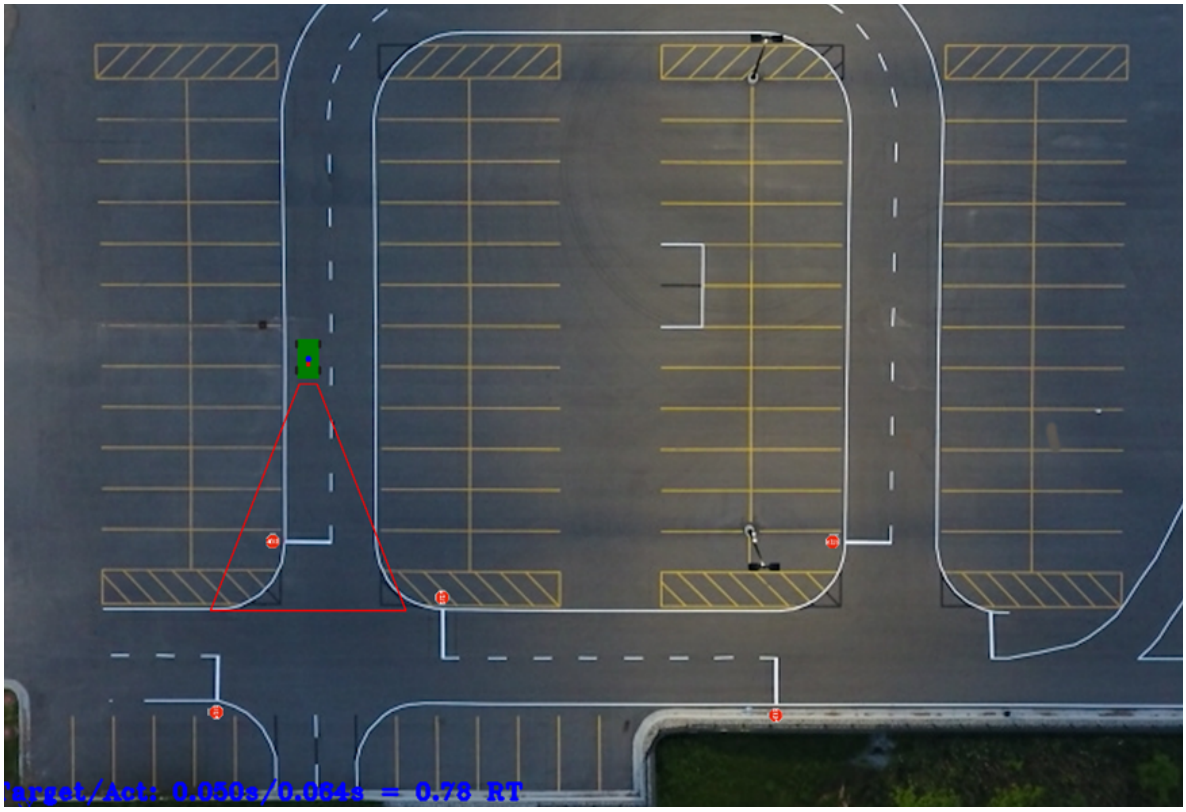

Figure 17: Simulated vehicle, camera and lidar and GPS in Gazelle Sim

9. PERFORMANCE TESTING & ASSESSMENT

Due to the modular nature of the software architecture (see Software Systems section) all functions (nodes) can be tested both independently and with any combination of other nodes. All nodes are thoroughly tested on the field and during development. Integration tests are also performed by creating specific situations for the vehicle to perform on. At the time of publication, most of the nodes have been tested thoroughly and several integration tests have been completed. The tests of most IGVC functions were performed on a test course created at LTU campus shown in Figure 18. There are no major performance issues with the vehicle's mechanical, electrical, or physical hardware to date.

Figure 18: Setup of a test course on campus, parking lot H

10. INNOVATIONS & SUMMARY OF RESULTS

The ACTor project is a research environment for students interested in autonomous software development. The project was designed to be modular, incremental, and dynamic meaning that software modules are easy to switch out, add, or remove allowing rapid prototyping and development. Testing results show that the ACTor vehicle is capable of performing IGVC Self-Drive challenges. The design spawned several research projects involving software engineering, machine vision, and deep learning. Innovations we achieved include:

- Using Docker container to integrate RTK with newer Ubuntu and ROS
- Lua script language to specify vehicle behaviors in high level
- Human and object detection using Yolo v5 Nano
- Extension of Gazelle Sim, a lightweight 2D simulator

REFERENCES

[1] 2021 Self-Drive Design Report, http://qbx6.ltu.edu/chung/stuproj/2021/IGVC2021_DesignReport_LTU.pdf, accessed 05-10-22

[2] Mako g-319, accessed 03-14-21, 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, http://velodynelidar.com/vlp-16.html, accessed 06-01-2021

[5] URG-04LX-UG01, https://hokuyo-usa.com/products/lidar-obstacle-detection/urg-04lx-ug01, accessed 05-15-22

[6] Swift navigation piksi multi gnss, https://www.swiftnav.com/piksi-multi, accessed 06-01-2021

[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] N. Paul and C. Chung, "Application of HDR algorithms to solve direct sunlight problems when autonomous vehicles using machine vision systems are driving into sun," Computers in Industry, vol. 98, pp. 192– 196, 2018.

[9] B. Warrick "Development of LTU ACTor (Autonomous Campus TranspORt) Vehicle Model (Polaris GEM e2) using ROS GAZEBO," 2018, http://qbx6.ltu.edu/mcs/ClassProjectAwards/1718/ACTor_Simulation.pdf

[10] 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

[11] Ian Timmis, Nicholas Paul, Chan-Jin Chung, Teaching Vehicles to Steer Themselves with Deep Learning, 2021 IEEE International Conference on Electro/Information Technology, May 14 - 15, 2021, Organized by Central Michigan University

[12] 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

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