ACTor: Design Report

Lawrence Technological University

Team Captain:	Mitchell Pleune	mpleune@ltu.edu
Team Members:	Sean Bleicher	sbleicher@ltu.edu
	Charles Faulkner	cfaulkner@ltu.edu
Faculty Advisors:	Dr. ChanJin "CJ" Chung	cchung@ltu.edu
	Nicholas Paul	npaul@ltu.edu

May - 15 - 2019

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.

Signatures

ChanJin Chung [illegible]

Date: May 15, 2019

INTRODUCTION

The **A**utonomous **C**ampus **T**ransp**or**t (ACTor) is a autonomous vehicle research platform developed by university students*. The project started in the spring of 2016 when two computer science students set out to compete in the new IGVC Spec2 competition. The team has grown in size now having four members who collaborate in researching and developing new technologies within autonomous vehicles.

Innovations and Previous Work

The project was designed to be a research platform for all students interested in autonomous vehicles. For that reason the project was designed to be modular and dynamic meaning that software nodes are easy to switch out, add, or remove allowing rapid prototyping and development. The design spawned several research projects involving machine vision, deep learning, and sensor fusion[13]. For more details on how these goals are achieved through design, see the *Software Systems* section of this document.

Team Organization

The team meets once a week to set goals and discuss new innovations and technologies to developed for the vehicle. 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 task by comfortability and interest. The team is comprised of all computer science students so hardware integration was done as a team or using help from a mechanical design lab to accomplish the hardware installation. Each student puts around 5 to 15 hours a week coding, testing, or in meeting to accomplish the tasks set out for the semesters.

Name	**Degree**	**Role**
Sean Bleicher	B.S. Computer Science	Sensor fusion, drive-by-wire system
Charles Faulkner	B.S. Computer Science	Obstacle detection, implementation
Nicholas Paul†	M.S. Computer Science	RTK integration
Mitchell Pleune†	B.S. Computer Science	Integration, waypoint navigation, lane following, hardware

Table 1: Team members and roles

* The Drive-by-wire system was provided by Dataspeed Inc.

† These members participated in IGVC 2017 Spec-2

MECHANICAL SYSTEMS

ACTor is built on top of a modified Polaris Gem 2 provided by joint sponsorship from two companies, Mobis and Dataspeed. Mobis provided the base vehicle, and Dataspeed installed the drive-by-wire system. The provided system includes all hardware and a software abstraction layer to control it.

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. Thr radar are mounted to the front bumper of the vehicle. All components mounted outside the vehicle are waterproof and resistant to damage by rain.

System 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.

Drive-By-Wire

Since the drive-by-wire system was installed by Dataspeed it is covered by a NDA, it cannot be extensively explained. Their system is capable of setting the vehicle in motion at exactly a target speed, and turning the vehicle at a specified radius. Their systems use complex mechanical systems to overcome the stiction of the steering wheel, and can detect abrupt steering inputs made by the driver.

The drive-by-wire system uses a proportional-integral-derivative (PID) controller to handle vehicle speed. Last year it was noticeable that the vehicle jerked and had a slight delay in the response from the system. Upon investigation the PID controller was optimized for the vehicle to drive at a much higher speed. The PID controller was tuned for the competitions max speed of five miles per hour. The result of the tuning was a major reduction in vehicle jerk and removal of the response time delay.

In the past few years of IGVC, the vehicle’s inability to change the gear autonomously has been a problem, causing functional tests such as parking to be difficult to achieve. To accomplish automatic gear shifts an arduino was installed within the drive-by-wire system that acts as a Controller Area Network (CAN) bus. The arduino uses CAN to send and receive messages allowing the vehicle’s gear position to switch the way by which the vehicle’s gear position is set: the physical state of the gear switch or virtually through the autonomous system. The vehicle’s drive-by-wire software driver manages the virtual gear position by looking at the twist messages that are being sent from the twist controller. Positive speed values result in the vehicle being in the forward gear position, while negative speed value result in the vehicle being in the reverse gear position.

ELECTRICAL SYSTEMS

Core Components

Most of the computation happens on the primary computer with the exception of the webserver and vehicle driver nodes on the Intel Nuc.

Primary Computer Component	**Primary Computer Part**
Case	SilverStone Technology Ultra Compact Mini-ITX
Motherboard	GIGABYTE GA-AB350N
CPU	AMD Ryzen 7 2700 Processor
CPU Cooler	CORSAIR Hydro Series H60 AIO Liquid CPU Cooler
GPU	ZOTAC GeForce GTX 1070 Ti MINI 8GB GDDR5
RAM	Corsair LPX 32GB DRAM 3000MHz
Storage	Samsung 860 EVO 500GB M.2 SATA Internal SSD
12V PSU	500 Watt 12 volt DC Input PC ATX-24 pin

Table 2: Primary PC build 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 50 degrees field of view. A ZED Stereo camera was added to provide depth and motion information. The ZED camera[15] captures video of 110° wide-angle at 1080p HD video at 30FPS.

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. Four Autoliv radar[5] have been added to the front bumper of the vehicle providing more than 180° of 77 GHz high-resolution radar detection.

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.

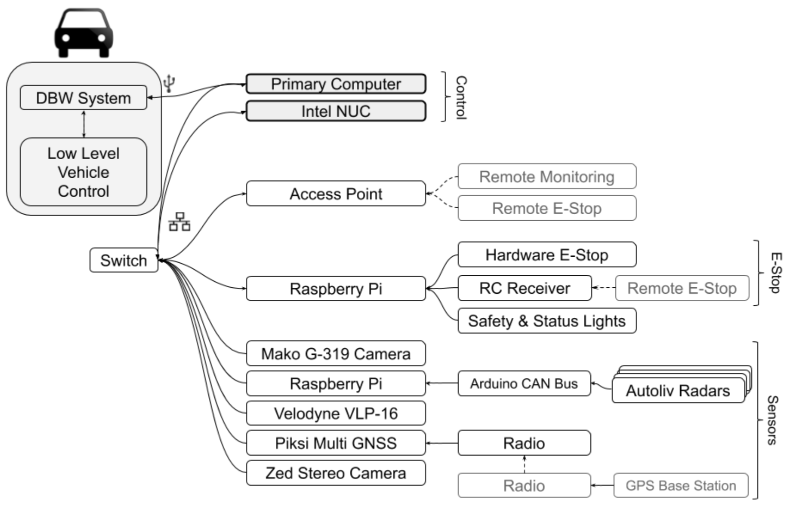


Figure 1: All sensor and control nodes are connected to a local network through Ethernet via a switch or CAN via a CAN Bus. Solid connections are indicate physical cable connections. Dashed connections indicate wireless communication.

All systems of the car have been connected through either Ethernet and CAN buses instead of USB, as seen in Figure 1. 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.

Item	Price
Polaris GEM e2 vehicle with various options such as doors and trunk	$15,000.00
Polaris GEM ADAS Test Systems (Drive-By-Wire systems) including installation fee	$45,000.00
Intel NUC Mini PC kit NUC7i5BNH Core i5	$543.00
Velodyne VLP-16 “PUCK” 3D LiDAR, 16 beams	$7,999.00
Autoliv (Veoneer) Radar x 5	$723.24
Swift GPS, Piksi Multi GNSS	$1,644.56
Mako PoE Camera	$1,031.00
ZED stereo camera	$449.00
Main Computer with GPU	$1,800.00
Miscellaneous items	$2,000.00
Total	$76,189.80

Table 3: Estimated cost of ACTor vehicle

Power Connections

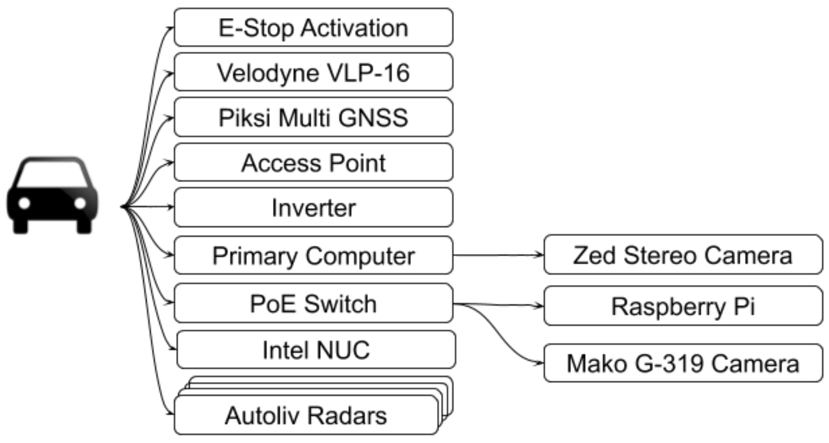


Figure 2: Vehicle, sensor, and components power distribution

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 connections are shown in Figure 2.

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 main computer.

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 allows the software to be quickly test and allowing for smooth implementation of new hardware and software.

Architecture and Design Strategies

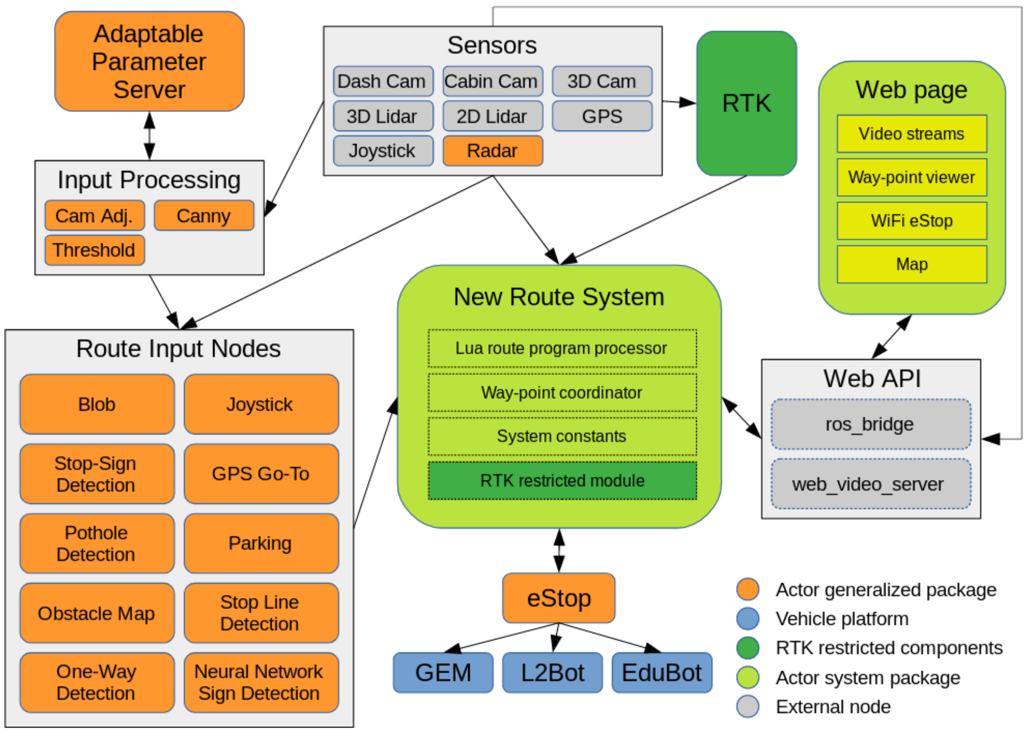


Figure 3: Basic data flow through core packages

The system is divided into several independent but connected packages: sensors, input processing, route input, route system, web API, and RTK. Figure 3 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 GPS, lidar, radar, or camera has its own node or nodes that are responsible for converting data into usable formats and publishing information onto the ROS network. The system provides abstraction of data through placing the responsibility of reading and converting the data in the input packages, 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.

Input Processing

The input processing package cleans and prepares the data for the route input package. Currently the nodes within this package are 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 importance of the route input package is that in provide 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's important to make sure they are running while the route system needs them.

The route input package contains our lane centering algorithm called "blob", obstacle maps for avoidance and emergency monitoring, detection of stop sign, one way sign, stop line, and potholes, gps follow, and parking maneuvers. Each of these node can be combined together to accomplish the many tasks and functions the ACTor vehicle will be performing.

Route System

The vehicle is entered into IGVC Spec2 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 will 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 estop, get the local obstacles, check the current waypoint target, and the ability to talk to many RTK components.

Our system is centered around the router, and it's primary goal is to feed the route script as much data as possible. In this diagram, the RTK system is shows as a single block, as it is thoroughly documented already. The "RTK restricted module" is a single Git patch file that adds functionality to the route node to read information from the RTK system, allowing the route system to be open sourced with the patch kept secure.

Web API

The ACTor vehicle receives routing data and displays useful diagnostic information through a custom designed web page 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 componentes can be added to the server without having to modify the entire page or write complex css to make everything work together nicely. The website displays a live feed of the "blob" nodes output allowing viewers to see where the lane centering algorithm is trying to center the car. The route system takes input from the web server. The input consists of a editable text field full of the current route the car is going to take. This allows rapid development of new routes and the ability to easily edit older routes.

Robotics Technology Kernel (RTK)

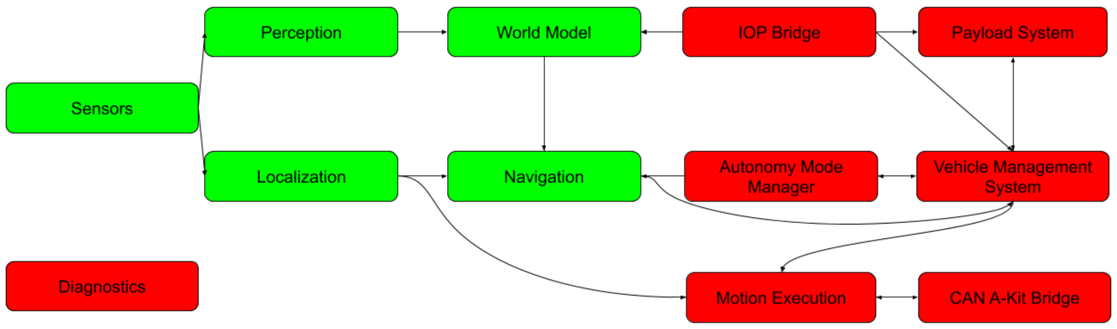


Figure 4: Packages used from the RTK system

In order to make the best use of the limited preparation time for IGVC and improve the RTK system as much as possible, we will be focusing on a limited subset of the RTK system as shown in Figure 4. Each node represents a package within the RTK system. The green packages are the packages that were viable to be integrated into the ACTor system, while the red packages did not fit anywhere in our current software design. We combined the existing design and implementation of the ACTor navigation system with the existing RTK navigation system. The ACTor navigation system is robust, extensible, and has proven itself useful by playing a primary role in winning 1st place in IGVC two years in a row.

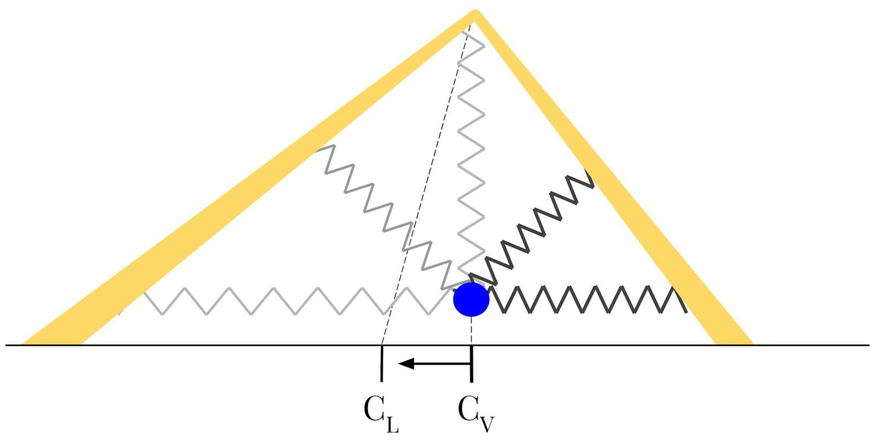


Figure 5: The "blob" algorithm uses simulated springs to push the vehicle's center C_V toward the center of the lane C_L. Thicker lines represent more compression and therefore more influence on the direction the point will move.

Lane Following

Lane detection and centering is combined into a single algorithm nicknamed "blob," designed specifically for early use in this project until it can be integrated into the obstacle avoidance subsystem. It leverages OpenCV to do the processing on a color image.

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.

Finally, a point C_V (see Figure 5) 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.

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 environment.

Automatic Parameter Adjustment

Autonomous vehicles are immensely complicated machines, and the algorithms that govern them require countless parameters. Our Adaptable Parameter Server tunes multi-dimensional parameter sets to optimal values using gradient descent.

Often it is a more robust solution to manually tune parameters to a best-case compromise, the wide range of weather/lighting/course conditions that are a possibility throughout the competition, some parameters must change on-the-fly. For this situation, we have developed a black-box parameter server that can tune a multi-dimensional parameter set based on a specific grading function. This node uses gradient descent as an extremely robust algorithm that can work on data in real time. This is also able to optimize a set of parameters even when there is not an ideal solution.

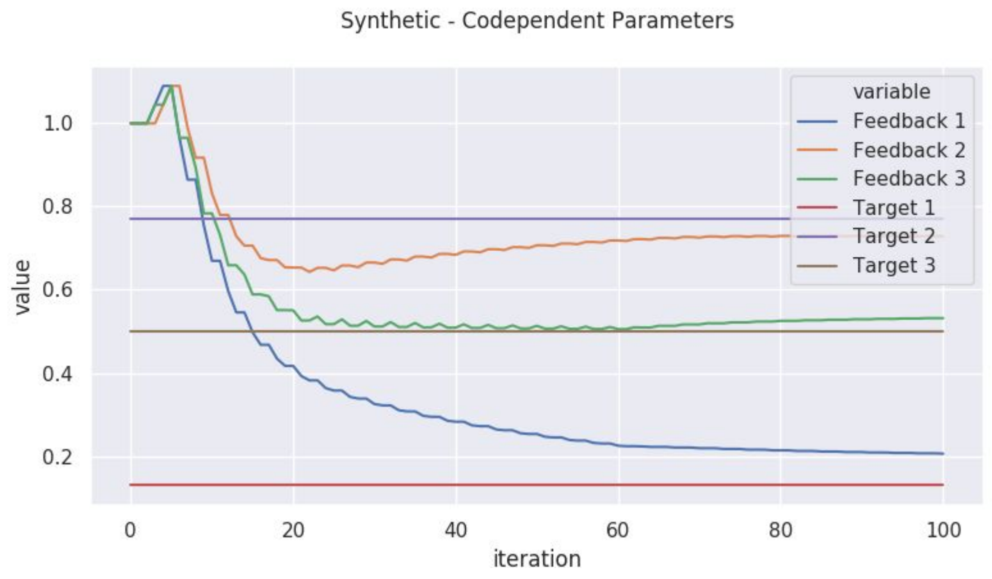


Figure 6: A synthetic test of the black-box parameter server where an ideal solution is impossible. Not shown are three parameters that affect each two out of three of the feedback parameters (outputs of scoring function)

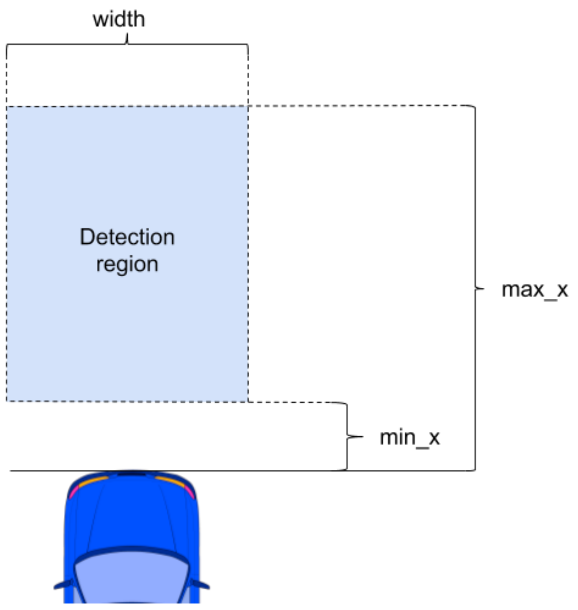


Figure 7: The obstacle detection node allows for detection of objects in arbitrarily defined spaces. This allows different events to occur based on which detection region the object is in.

Obstacle Detection and Resolution

The obstacle avoidance package currently uses input from the VLP-16, Autoliv radar, and a ZED stereo camera. Using functionality built into the TARDEC RTK system a ground and obstacle pointcloud are generated from the VLP-16's input. The obstacle pointcloud is then combined with the input from the ZED camera and verified using the radar inputs.

The current implementation of obstacle avoidance checks regions defined by parameters (see Figure 7). 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.

SAFETY

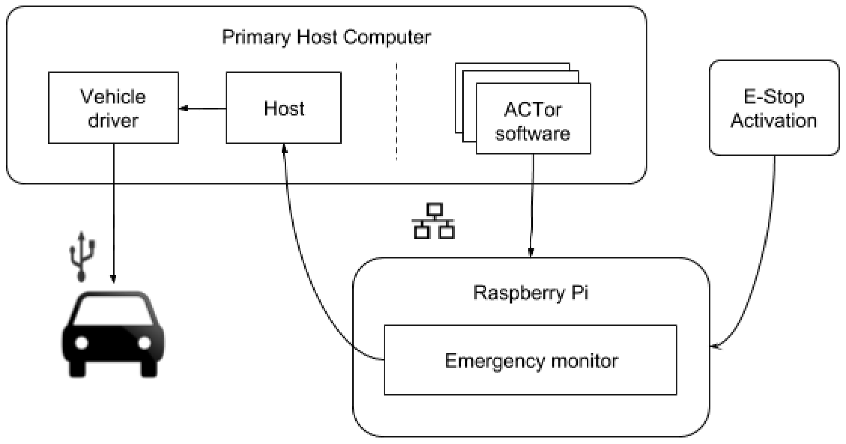


Figure 8: The ACTor software is unable to communicate with the vehicle driver directly. All commands are validated via external hardware. If the external hardware's state becomes invalid the "Host" issues an emergency stop command.

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 8). 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.

eStop Performance

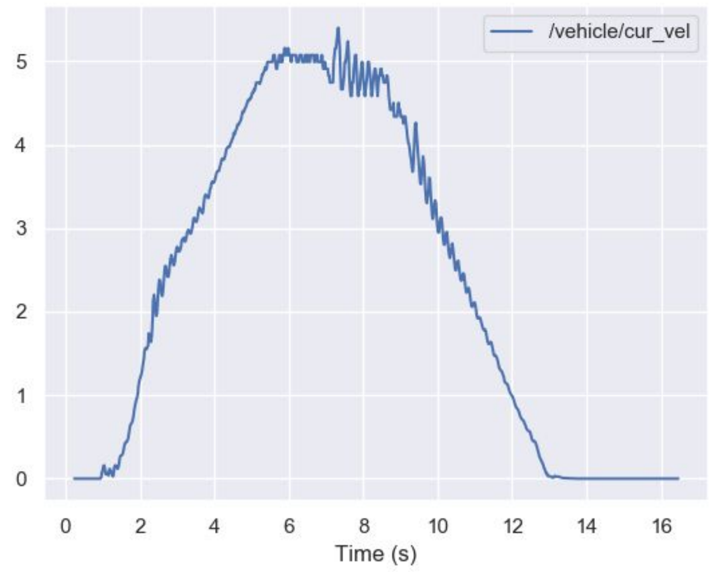


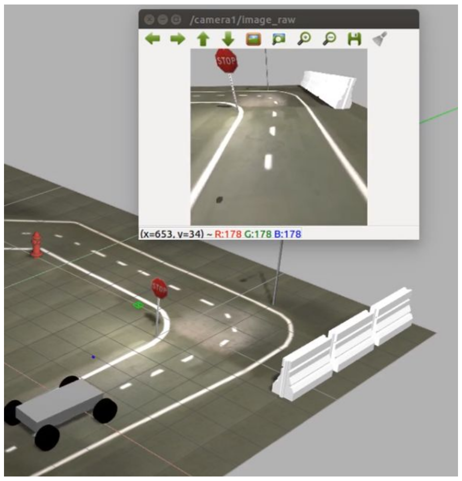
Figure 9: The wheel speed of the vehicle during an eStop event. Values are in meters per second. The estop was triggered at 7 seconds after the vehicle had reached maximum speed of 5 m/s (>11 mph).

There is minimal latency between between the beginning of an eStop event to the stopping of the vehicle. Figure 9 above shows the vehicle speed during an eStop event. In this example, the system was given a configurable target deceleration of 0.5 m/s^2 deceleration during stop. The effects of the motor slowing down can be seen instantly as noise as the vehicle pitches forward. After this non-avoidable compression time of the suspension has stabilized, the vehicle reaches its target deceleration.

Failure Points and Modes

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 external safety monitor due to loss of power and issues an E-Stop command within 200ms.
Raspberry Pi (Safety Monitor) malfunction	Hardware	Low	Primary compute 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, 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.

SIMULATION AND PROTOTYPING



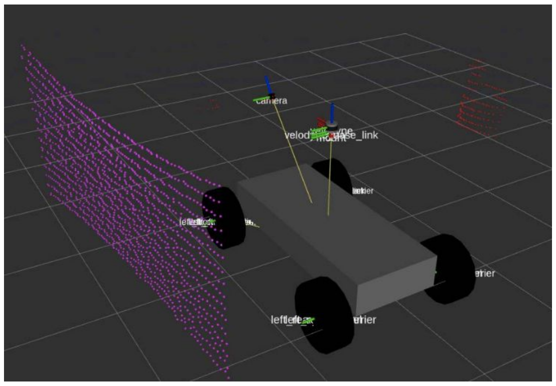


Figure 10: Simulated vehicle and camera demonstration (left) and accurate virtual sensor layout and transforms (right)

Simulation and prototyping is accomplished using Gazebo robotics simulator. Gazebo offers two advantages over traditional prototyping methods. First, it models specific sensors, such as a specific model of a lidar system, and also allows for the specific dimensions of the vehicle itself to be modeled. Just like physical sensors and motors, these simulated models function as independent nodes within the ROS architecture, sending and receiving the exact same data objects within the ROS network. This means that simulated sensor output can be fed to the same sensor input nodes and navigation/control nodes as the physical systems, and likewise the output from these nodes can be sent to the simulated vehicle (see Fig. 3). The physical vehicle and sensor nodes can be swapped with the gazebo nodes transparently to the primary autonomous vehicle software in order to conduct more realistic testing – quickly and conveniently on our laptops.

Second, gazebo provides a simulated world in which the robot can operate, complete with gravity, friction, momentum, light, color, and many other physical properties that allow for a rich testing environment.

Testing algorithms and other concepts is achieved relatively quickly within the simulation. Examples include testing a camera vision lane following algorithm using the simulated camera on the vehicle model driving over a variety of ground images, experimenting with point cloud data from the 3D LiDAR system, and testing work with object detection and avoidance,

PERFORMANCE TESTING

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, the most of the nodes have been tested thoroughly and several integration tests have been completed. The team is expected to develop and test many further integration tests in the weeks leading up to the competition.

PERFORMANCE ASSESSMENTS

There are no major performance issues with the vehicle's mechanical, electrical, or physical hardware to date.

REFERENCES

[1] "Nvidia machines/embedded-systems-dev-kits-modules, accessed: 18 Feb 2018.

[2] "Mako g-319," https://www.alliedvision.com, accessed: 10 Feb 2018.

[3] "1stvision 1" 2 to 3 megapixel oem lens series," https://www.1stvision.com/lens/spec/1stVision/LE-MV3-0618-1, accessed: 10 Feb 2018.

[4] "Velodyne puck," http://velodynelidar.com/vlp-16.html, accessed: 11 Feb 2018.

[5] "Radar systems," https://www.autoliv.com/products/electronics/radar- systems, accessed: 10 Feb 2018.

[6] "Swift navigation piksi multi gnss," https://www.swiftnav.com/piksi- multi, accessed: 18 Feb 2018.

[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," inICRAworkshoponopensourcesoftware,vol.3,no.3.2. Kobe,2009, p. 5.

[8] "Point cloud library," http://pointclouds.org/, accessed: 10 Feb 2018. [9] M. Quigley, E. Berger, A. Y. Ng et al., "Stair: Hardware and software architecture," in AAAI 2007 Robotics Workshop, Vancouver, BC, 2007, pp. 31–37.

[10] A. Oreba ¨ck and H. I. Christensen, "Evaluation of architectures for mobile robotics," Autonomous robots, vol. 14, no. 1, pp. 33–49, 2003.

[11] G. Stein, Y. Li, F. Wei, D. Butani, N. Changani, N. Reddy, C. Chung, and J. Ruszala, "Team bigfoot 2 igvc 2016 design report," 2016.

[12] C. Stein Gordon, Chung, "Rapid development of a mobile robot simulation environment for the intelligent ground vehicle competition," AUVSI XPONENTIAL 2016.

[13] 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.

[14] "Zed Stereo Camera," https://www.stereolabs.com/zed/, accessed: 15 May 2019.

[15]B. Warrick "Development of LTU ACTor (Autonomous Campus TranspORt) Vehicle Model (Polaris GEM e2) using ROS GAZEBO," 2018.