Sedanii Design Report

Georgia Institute of Technology - RoboJackets

Matthew Barulic mbarulic@gatech.edu
Evan Bretl evan.bretl@gatech.edu
Brian Cochran bcochran32@gatech.edu
Austin Keener akeener3@gatech.edu
Varun Madabushi vmadabushi3@gatech.edu
Joseph Spall IV jspall3@gatech.edu

Fig. 1. RoboJackets' 2019 Entry: Sedanii

***Abstract*—This report introduces Sedanii, the Georgia Institute of Technology's entry into the 2019 International Autonomous Robot Racing Competition. We detail the improvements made to our robot's mechanical, electrical, and software subsystems between 2018 and 2019, and how they enable us to effectively complete the competition challenges introduced for 2019.**

I. Introduction

RoboJackets is the Georgia Institute of Technology's competitive robotics team. The team was founded in 1999 dedicated to robotics promotion, education, and advancement throughout the Georgia Tech community and beyond. RoboJackets currently has over 270 members, about 15 of which are involved on the IARRC team. RoboRacing consists of three subteams led by a project manager. The project manager's responsibilities include placing orders, setting milestones for the team, and making sure the team hits those milestones. The three subteams — electrical, mechanical, and software — are each led by subteam leads who are responsible for making technical decisions and assisting their members in achieving the set goals.

This year, we continued work on the platform Sedani, which debuted in the 2018 IARRC. The build for this robot started in May 2018, and our performance at the previous competition motivated a series of upgrades across all subteams. Our mechanical and electrical subteams focused on increasing stability, maintainability, and controllability. Our software stack was completely redesigned, taking advantage of our other upgrades to improve our perception and path planning functionality.

II. Mechanical

A. System Overview

The mechanical subteam's function is to support the computing and sensory requirements of other subteams in the effort to efficiently develop equipment for autonomous racing. Several of the upgrades and new features for the vehicle will be covered below along with the considerations taken into account while each was under development.

B. Encoder

Feedback is a necessary component for closed-loop control. Without it, the system is left blindly guessing using feedforward functions of lookup tables. For IARRC 2019, an encoder was added to Sedanii to provide feedback for closed-loop speed control. We considered two possible designs for speed feedback: hall-effect sensors with magnets mounted to the inside of the tires, and a self-enclosed encoder mounted to the drive shaft connecting the front and rear differentials. While the self-enclosed encoder option required more complex modifications to the vehicle, it was also deemed more robust for handling the high-speed driving of the vehicle.

This addition required redesigning two stock components: the central aluminum plate joining the front and rear chassis assemblies and the drive shaft connecting the front and rear differentials. The drive shaft had to be replaced as the stock shaft has key pins welded to each end, preventing the encoder disk from being slid onto the shaft. The replacement shaft simply uses press-fit key pins which are installed after the encoder disk and can be pressed back out if necessary in the future. The replacement central plate simply swapped weight-saving cutouts in favor of mounting holes for the encoder brackets.

With these two stock components replaced, two identical, aluminum mounting brackets were machined and installed to house both the encoder sensor body and bearings. A shaft collar and needle thrust bearings were installed between the mounting brackets to keep the shaft from sliding axially, which would damage the encoder. The final assembly keeps the shaft secured for the encoder while accommodating the high speed of the shaft while driving.

C. Cameras

The new challenges presented in IARRC 2019 and the new techniques being implemented by our software team

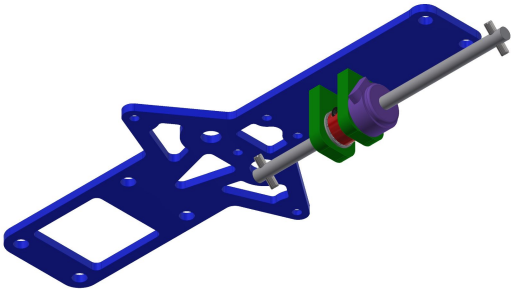

Fig. 2. Digital design of encoder and central plate assembly

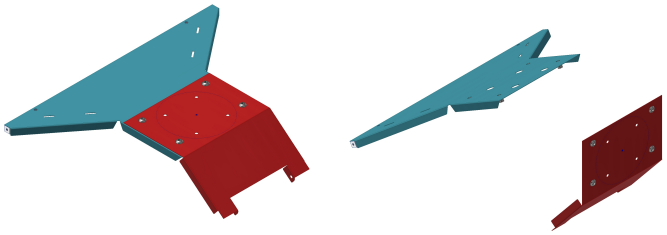

Fig. 4. Digital design of new hood assembly

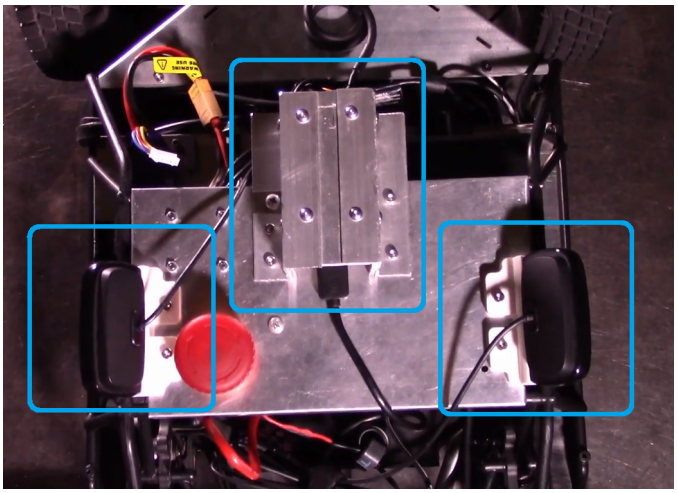

Fig. 3. Top view of Sedanii with the three cameras boxed in blue

required an upgrade to our vehicle's camera suite. Last year, Sedani featured a single, forward-facing FLIR Blackfly camera mounted to the bottom of the top plate of the computing enclosure. The new requirements necessitated moving the forward camera to the top of the computing enclosure and adding two additional side-facing cameras. To protect the relatively expensive forward camera, a new aluminum hood was fabricated using aluminum bar and angle stock cut to the length of the camera body and lens assembly. The new side cameras are cheaper web cameras, so crash protection was deemed not worth the time needed in this year's team schedule. Form-fitting, 3D printed PLA mounts were designed and manufactured to secure the cameras to the hood using the small screw that normally holds the camera body to its stand out of the box.

D. Replacement Hood

Sedanii features, as it did last year, an enclosure for the control electronics which enable computer control of the vehicle. With the Losi chassis intact, the stock hood piece blocks access to this enclosure as it must be completely removed to open the control enclosure. Additionally, each time this hood piece is removed and replaced, machine screws must be removed from plastic parts, risking damage to the thread profile in the plastic pieces. To address this concern, a new hood was designed to replace the stock component. While redesigning this section of the vehicle, we took the opportunity to lower the height of the LIDAR mount to reduce the degree to which it blocks the forward camera's field of view. The design allows for a LIDAR to be utilized, but it will not be used for this competition.

The new hood design features two pieces of sheet metal, cut on a water jet and folded to mimic the overall form factor of the stock hood piece. The two pieces of sheet metal are screwed into two square aluminum beams running the width of the car where the stock hood previously mounted. These beams replace the plastic parts on the original car whose threads risked being damaged. The two sheet metal pieces stay attached at the four corners of the square beams, but can separate at the middle, opening like a draw bridge to allow access to the enclosure underneath. When closed, the two sheet metal pieces are held together with screws which thread into metal tee-nuts epoxied to the bottom of the lower piece. The LIDAR mounts secure to the upper piece and clearance holes are cut into the lower piece to make room for the LIDAR screw heads. This design was chosen because it allows for easy access to the electronics below and is a much more robust design than the original plastic frame.

III. Electrical

A. System Overview

The Electrical subsystem comprises of two distinct parts - the Chassis module and Compute module. Low-level functions such as steering, speed control, and emergency stop are handled by the chassis module, while high-level functions such as sensing, perception, and control happen on the Compute module. These modules feature their own batteries and communicate through a single USB cable. Thus, they can be developed separately and integrated through a single link.

After the previous IARRC, we upgraded our control board and integrated feedback from the newly-added encoder. Our chassis control state machine was completely redesigned to support these new features and provide a robust controls solution. This improves the safety of our system and ensures accurate transitions between states such as running, e-stopped, braking, etc. In addition, it permitted speed control through a PID loop incorporating a feedforward model for fast step-response performance.

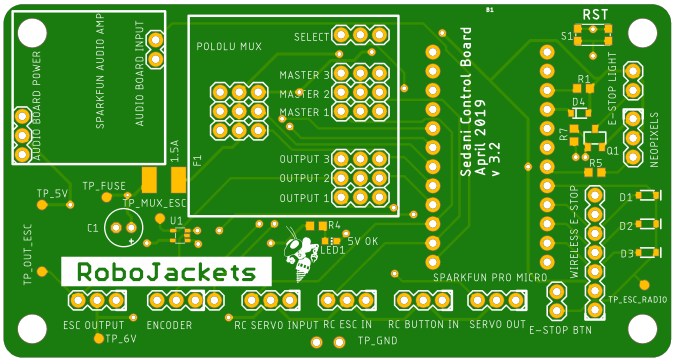

Fig. 5. Control Board PCB

State	Condition
DISABLED	If the wireless remote or emergency push-button are pressed
TIMEOUT	If communication with the NUC has not occurred since timeout duration in an Autonomous state
MANUAL	If the RC remote is in control of the robot
FORWARD	If the NUC commands autonomous forward motion
FORWARD BRAKING	If the robot is going forward and the NUC commands zero speed
IDLE	If the robot is at zero speed and the NUC commands zero speed

TABLE I
A LIST OF STATES IN OUR EMBEDDED STATE MACHINE

B. Control Board

The purpose of the control board is to take control information from the main software stack running on the Intel NUC and output control signals to actuate our electro-mechanical system, a steering servo and an ESC for the brushless DC motor. The low level chassis controls are implemented on a Sparkfun Pro Micro development board with an Atmega 32u4 microcontroller. Other major features carried over from previous versions of the board include a multiplexer to switch between our RC remote and software control and a wireless receiver for controlling the emergency stop system.

The team performed a major revision on the main control board. The board is now powered directly by the on-board ESC voltage regulator, allowing the Chassis module to function without the on-board comupter enabled. The original emergency stop system featured a relay, but has been replaced by a solid state system to reduce package size and increase product lifetime. A status LED was added to the mechanical emergency stop pushbutton to indicate the state of the system. Minor additions also include test points to improve debugging on the PCB and power indicator LEDs for all power rails to verify correct operation.

C. State Machine

The previous firmware system ran using a complicated state machine that simply focused on current inputs and did not take into consideration the current state. Due to the need for an easily debuggable firmware system in order for successful maintenance of the platform, the team re-designed the state machine from scratch. We broke down the possible states of the robot to six distinct situations, detailed in Table I.

At the beginning of each state, the code runs the easily configurable function associated with that state. Afterwards, the status variables required for transitioning are evaluated with clear conditionals, with all possible transition destinations encapsulated within each state. Figure 6 shows the flow of transitions and possible states of the robot. Re-writing the state machine implementation as a Mealy machine, with different destination states depending both on the starting state and status variables, allows for more granular control of transitions rather than just focusing on status variables, simplifying and solidifying control flow.

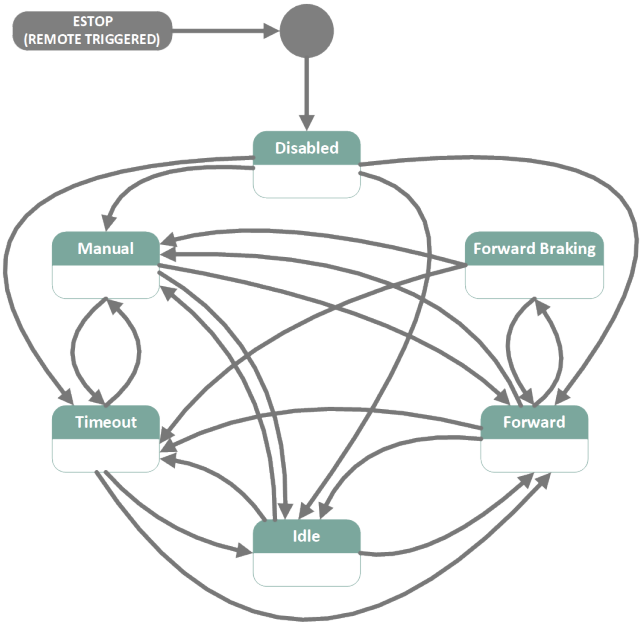

Fig. 6. A simplified version of the firmware state machine showing states and transitions.

D. Speed Sensing and Control

The software system's change to include short-term mapping and dead reckoning motion brought about a need for accurate speed control and measurement. The control board was designed to support this with interrupt pins broken out into two connectors. A US Digital E4T encoder was selected with 360 pulses per revolution. This is a quadrature encoder with two channels that pulse high or low to indicate motion of the shaft. We read the encoder through Interrupt Service Routines running on the microcontroller, bound to rising edges on the two encoder channels. A running count of the number

of pulses is kept, and speed is determined by counting the number of pulses that occur within a given time frame (the period of our main control loop at 25 milliseconds).

As we previously did not have any internal speed measurement, our speed control was a feedforward system that used an experimentally generated model to predict the relationship between an input signal (RC Servo-style PWM) to the robot's steady-state speed. Our upgraded system utilizes a PID loop for speed control, with additional feedforward control utilizing the model from previous iterations. A traditional PID loop can be expressed with the following equation:

$$u(t) = K_p \times e(t) + K_i \times \int e(t)dt + K_d \times \frac{de(t)}{dt}$$

The system error, e(t), is the difference between the control setpoint and the system feedback, K_p is the proportional gain, K_i is the integral gain, K_d is the derivative gain, and u(t) is the system output, in our case the output to the motor. The proportional term is used to provide an output response proportional to the system error, the integral error is used to compensate for error build up over time, and derivative gain is used to react to sudden disturbances in the system feedback.

The feedforward system adds an additional term to the equation, which is an independent function based on the velocity of the system.

$$u(t) = K_p \times e(t) + K_i \times \int e(t)dt + K_d \times \frac{de(t)}{dt} + F(v)$$

F(v) is the feedforward equation implementing a 2nd order polynomial dependent on velocity. This additional term increases rise time of the system and improves step response performance by avoiding the ramp up commonly found in PID loops for velocity control. In addition, the feedforward model simplifies tuning by getting us close to our ideal performance. A high proportional gain and low integral gain can be used to account for inaccuracies in the model and remove steady-state error. With preliminary tunings we were able to achieve a rise time under 0.4 seconds and a steady state oscillation amplitude less than 0.1 m/s, as seen in figure 7.

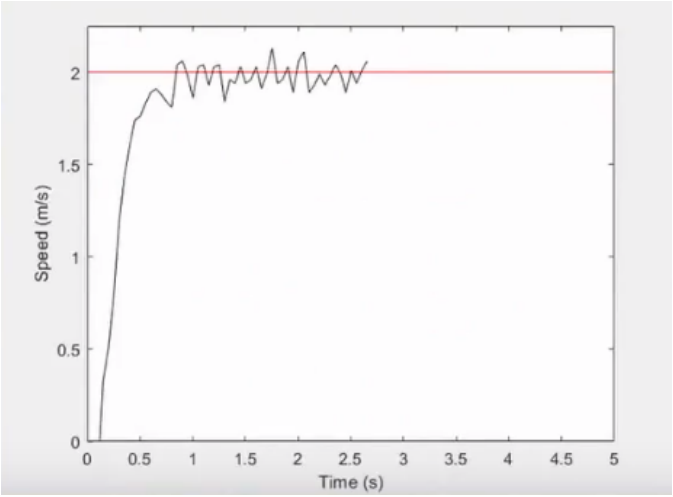

Fig. 7. The 2 m/s step response of our speed control with preliminary tuning

IV. Software

A. System Overview

Sedanii's autonomy stack has a layered architecture comprised of several logical subsystems.

- Perception: From camera images, we compute detection images, or black/white image masks showing the location of important features in the image (lane lines or cones). We typically have an ensemble of different detectors running at any one time.
- Mapping: From detection images and other sensor data, an integrated map of the area around Sedanii is created. The resulting map is a collection of 2D points in the robot's coordinate frame.
- Planning and Control: Different controls strategies are used for different events, including a simple steering controller for straight-line lane keeping as well as a model-predictive controller for snaking between obstacles.

Sedanii's software builds on Robot Operating System (ROS), a distributed computing framework for robots. ROS provides message-passing functionality so that our subsystems can be run as separate processes and so that software components can be swapped in and out as necessary. We write ROS nodes primarily in C++, with some in Python where speed is less critical.

B. Perception Layer

Sedanii's perception layer features several feature detectors. Each detector follows roughly the same interface, subscribing to a 3-channel color image from the camera and publishing a 1-channel image identifying the presence of visual features at each pixel location. Most utilize a combination of OpenCV's efficient implementations of elementary computer vision algorithms as well as our own operations on top of those results.

The most important of these detectors picks out high-brightness lane lines. It uses a hybrid approach between edge detection and color detection, utilizing the strengths of each. The image is first run through the Laplacian edge detector (a simple linear kernel), which captures all edges in the image. There is no way to tune the sensitivity of this edge detector such that we find all the lane lanes and only lane lines, so we tune it to have many false positives and virtually no false negatives. Next, the same camera image is run through Gaussian adaptive thresholding, which compares the brightness of each pixel to a weighted local neighborhood to identify if that pixel is significantly brighter than its surroundings. When properly tuned, this method gives us few false-positive lane line detections, but it can sometimes miss far-away sections of lane lines. We utilize both approaches by using the Laplacian edge detection results to extend the adaptive thresholding results. Edges which border white regions larger

than a certain size are used for a flood-fill that extends the boundary of the color-thresholded region. Edges which do not border positive regions from thresholding are ignored. This hybrid method gives Sedanii reliable lane line detections across various lighting conditions.

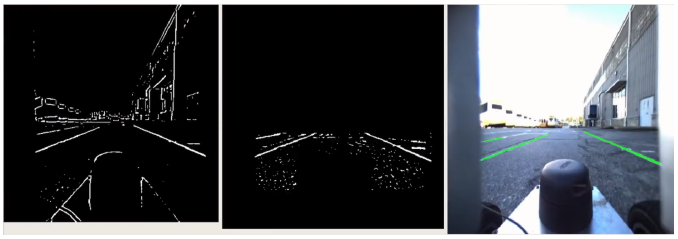

Fig. 8. Progression of lane detection operations, resulting in the green overlay of detected lanes on the far right image

To detect the orange cones, we first use color thresholding in the HSV perceptual color space to identify strongly orange regions of the image and then filter by size/shape to reduce false detections. Because orange is so visually distinct from everything else in a typical camera frame, this approach works much better for cones than it does for lane lines (which are visually fairly similar to the concrete underneath). Sedanii utilizes this information in two ways. For the obstacle avoidance challenge, the cone detection image is published in the exact same format as the line detection image. For the drag race, we also make an independent measurement of our distance to each cone based on its apparent height in the image and a pinhole camera model.

To detect Urban Challenge directional signs, we run a Canny edge detector over our front camera frame to get general shapes of the image. OpenCV's FindContours function is used to gain extra information about the shapes, including the size, number of edges, and similarity to known arrow shapes. An arrow must have seven edges, a correct width to height ratio, and have a high similarity to the known arrow shape based on Hu Moments. Once shape is classified as an arrow, it must be detected multiple images in a row in order to be counted as a directional sign.

Detecting the start light calls for a separate detector. We have improved on our algorithm for this since last year. The old detector required many hand-tuned inputs (size of circles, expected distance, etc) and had many false positives, whereas the new detector requires little to no hand-tuning. To track changes over time, a history queue of frames is kept. If a green circle is found, we look back in our history for a red circle of the same size above the location the green circle was found. Red areas in the image are determined by subtracting the green and blue channels from the red channel and thresholding. High red areas (like the red start light) are left. Green areas are found by multiplying the green and blue channels and subtracting the red channel. Multiplication is a darkening operation when colors are in the [0, 1] range, leaving only places that are high in both green and blue (the color of the bluish-green start light) and subtraction removes results with high red as well. Circles

Fig. 9. Sign detector testing with multiple candidate signs in the image

are identified using OpenCV's FindContours function, which outputs several measures (moments) from which circularity can be calculated.

Our detector for the finish line remains the same as in past years. We threshold by color in HSV (like the orange cones, the blue is fairly distinct below the horizon) and find the longest horizontal line which fits in the detected color region. If it spans enough of the image, we trigger a finish line detection message.

In addition to directional signs, Sedanii recognizes stop bars in order to detect the intersection and decide when to trigger the turn for the Urban Challenge competition. Using the overhead projection of our line detector, Probablistic Hough Lines is used to determine where stop bar candidates are. The angle of each line segment found is determined, and candidates are filtered based on their angle and length. Once a stop bar is determined, we track the Sedanii's distance to it and send a turn message only when we are at a set distance away.

C. Mapping Layer

Sedanii tackles the problem of map-building in a relatively limited sense. Since integrating our IMU and wheel odometry data over time introduces measurement drift, we don't attempt to localize Sedanii globally on the track. Thus, our mapping and localization system is only concerned with the relative locations of our robot over the past several seconds.

The first, most important step in this process is projecting the perception results from each of the 3 cameras onto the ground plane. The end result of this procedure is a point cloud (list of points) representation of all the visible lines and cones (obstacles) at this time. Each of the three cameras on Sedanii has a certain fixed pose relative to the robot's base coordinate frame. From each of these camera poses, and for each pixel in which that camera can see an obstacle, a ray is projected outwards until it crosses the X-Y plane. That crossing point is added to the current obstacle map. We mathematically model

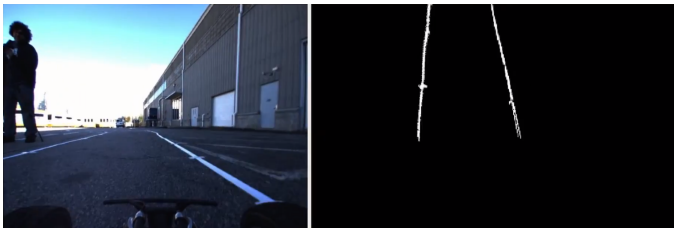

Fig. 10. The projection from camera view to global coordinate frame results in a map of obstacles

each camera using the popular pinhole projection model with a calibrated focal length parameter.

To combine these maps over the past several seconds, we can choose to maintain a relative pose history. This is a chain of coordinate frames linked by odometry data (IMU for heading and wheel encoders for speed). Using this graph, our local mapper finds the coordinate frame transformations between Sedanii's current pose and that of each of the last N measurements. With each of the point clouds now in the same (current, local) coordinate frame, the points are overlain into one point cloud map and filtered so the points have a max density.

D. Planning and Controls Layer

For the circuit race and obstacle avoidance challenge, we have developed a model predictive control (MPC) algorithm which can plan a route through an arbitrary collection of obstacles. We frame the route selection, path planning, and controls tasks as a single optimization problem. The state space for this optimization problem is all paths made up of N sections where each section has a constant steering angle control value. The objective function for this optimization is a weighted sum of several measures, including path straightness, distance from obstacles, and the speed at which that path can be executed. The trade-off between these competing objectives is controlled by changing the weights in the objective function. When no obstacles are present, the cost function is convex and can be quickly optimized with a greedy hill climbing algorithm. However, obstacles in certain locations often turn this into a nonconvex problem in our steering angle control space (e.g. when obstacles directly in front of the robot force a left or right turn). We approximate a global non-greedy optimization procedure using the Simulated Annealing algorithm. This algorithm starts off exploring many parts of the state space (many of which are highly sub-optimal), then transitions to greedily exploiting local improvements. This technique is much more likely to find a global optimum in a limited number of samples than greedy hill climbing or random sampling alone.

In order to efficiently navigate through the drag race with as much speed as possible, we chose to implement a simple midline following algorithm. After obtaining the overhead projection of the current lane detection image, we find the contours on each half of the image and filter out any contours that have less than a minimum area. Within each half of the

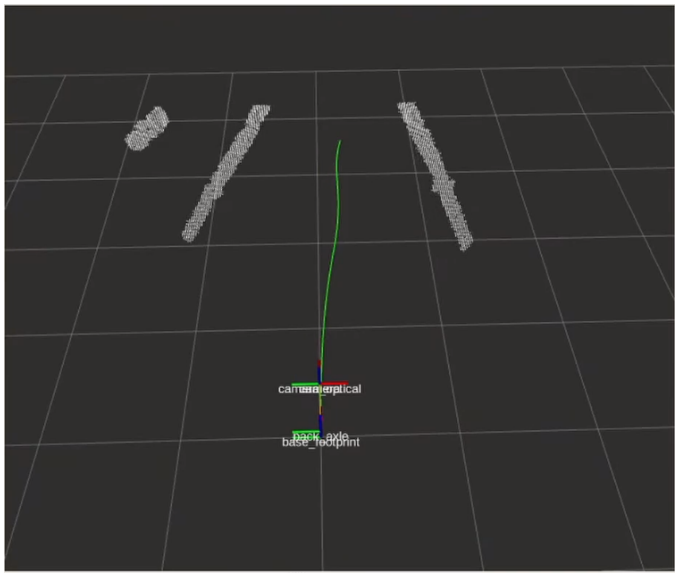

Fig. 11. Visualization of obstacles, in white, and a valid path, in green

image, the largest lane contour closest to the car (the bottom of the image) is selected to be either left or right lane, then a linear regression line based on the contour is calculated in order to generalize its shape. If both the left and right lanes are present within the image, a midline is computed by connecting the horizontal midpoints between the two associated regression lines. If only a single lane is present, the midline is computed by offsetting the associated regression line by approximately half the lane width. Once the midline is obtained, we use a PID controller to maintain the car along a path to follow this midline until the end of the drag race.

A new navigation controller was created for the Urban Challenge. Our solution is comprised of multiple layers. While watching for signs, the controller runs a lane keeping node. When a turn is detected, we use a dead-reckoning method by turning until our IMU reads a 90 degree turn in required direction or by driving straight for a set time. While this is not advanced control, it simplifies the sign following for this competition year and has worked well.

The lane keeper planner runs on the side cameras because the lanes are too close for the front camera. The planner synchronizes the two side camera line detection images and runs a Probablistic Hough Line algorithm on each which outputs a list of line segments. To account for the dashed left lane boundary as well as false positives, we calculate the angle of each line and run a weighted average of angled weighted on the line segment length. From the calculated angle, we plot a steering angle that keeps the car parallel to the lane boundaries.

E. Testing and Validation

The first step in testing Sedanii's autonomy stack is simulation. We use Gazebo because of its deep integration with ROS. The car and all its sensors (3 cameras, IMU, encoders) are modeled with noise injected into the readings, and the

data is made available to our system with the exact same ROS interface as on Sedanii. We primarily use the simulator to test and tune our mapping and planning/control subsystems. Visual features are not highly realistic in Gazebo, so we record files of camera data when running the car for offline testing of perception algorithms. After simulation and offline testing of features, we must validate our results on the physical car. We run monthly test days where we set up tracks for each challenge and run our full perception, mapping, and planning pipelines. While running the car, we record data of the car's state, including camera images, path plans, and steering and speed information. During later review, we step through the recorded robot states to find the causes of failures in order to better improve our system.

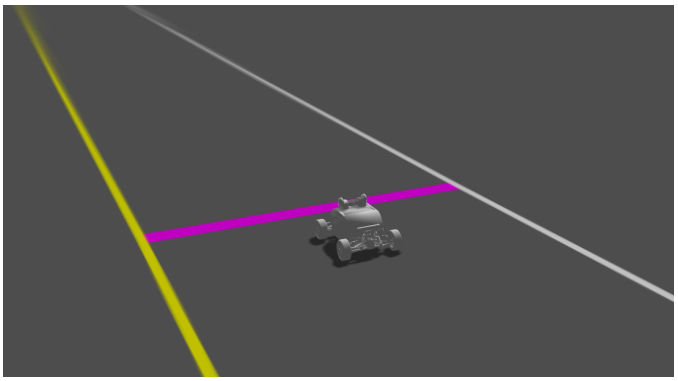

Fig. 12. A Gazebo model of our robot on the track