Georgia Tech RoboRacing “Macaroni” Design Report

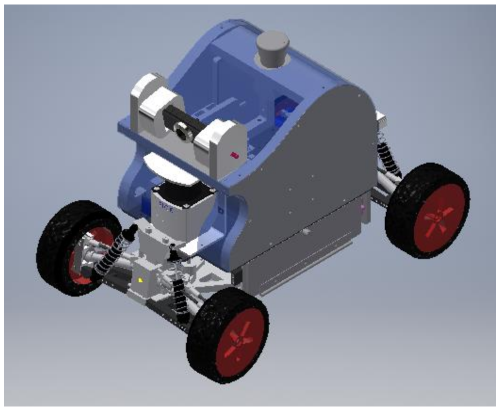
Evan Bretl Ransomed Adebayo Ryan Waldheim Sahit Chintalapudi

***Abstract*—This document comprises Georgia Tech RoboRacing’s submission for the “Written Report” section of the IARRC 2017 competition. It describes the hardware and software of the robot called Macaroni (*Abstract*)**

I. Team Introduction

RoboJackets RoboRacing, is one of several undergraduate robotics teams within the RoboJackets organization at Georgia Tech. We have redesigned most of our hardware and software to improve on last year’s IARRC entry. The most notable improvements are the new stochastic path planner and a more nimble physical platform, which we have named Macaroni.

II. Mechanical Design


Full CAD design

In comparison to the RoboJackets robot of the 2016 IARRC, Macaroni possess several key structural and aesthetic improvements. These include but are not limited to the addition of a lower mount plate for electronics, relocation of the camera and emergency stop switch as well as the redesign of the roll cage and polycarbonate protective sheet. the mechanical design aspects will be divided into 4 subsections: overall design/materials used, drivetrain, sensor mounts, and robot protection.

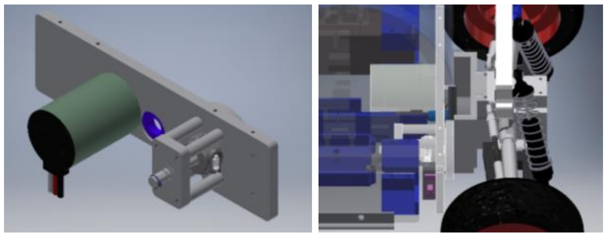
A. *Overall Design/Materials Used*

Although there are distinct differences between Macaroni and the previous robot, the fundamental build platform was kept the same. The bottom skeleton of Macaroni was obtained from a 1/10 scale Traxxas Slash 4X4 RC car. The 4 wheels, front and rear differential gears, servo, differential mounts, and suspension assemblies are among the components that were transferred to Macaroni. All connections and linkages that attach the wheels to the differential, as well as the gearing used to drive the differential box were also adapted to the new design. The majority of these parts were made out of plastic with a select few made out of metal. The remaining custom parts were machined out of aluminum (AL6061) or out of clear polycarbonate.

There are several noticeable changes in Macaroni’s overall mechanical structure from its predecessor. Firstly, there is now a protective aluminum and plexiglass roll cage to protect the electronics from impacts and water splashes. Second, the chassis has been lowered in order to make the robot more nimble, to make sure the LIDAR’s plane is low enough to sense obstacles, and to increase the space available to mount electrical components. Third, the tower at the rear of last year’s car has been replaced with lower, lighter mounts for the camera (which is now mounted to the front of the roll cage) and the E-Stop switch (which now protrudes from the highest point of the roll cage).

B. *Drivetrain*

The drivetrain refers to the components which physically perform navigation of the vehicle. Macaroni is powered by a brushless DC motor. Similar to its predecessor, the drive motor is mounted on a 1/4in aluminum plate, approximately 7in by 2in, as shown in the figure to the right. This mount is located close to the rear differential and secured to the top and bottom plate. A 13-tooth gear is mounted to the spindle of the motor. This gear then drives a larger 54-tooth gear which is directly connected to the rear differential and subsequently drives two rear wheels. Another drive axle has been added this year, connecting the front and rear differentials, and completing the four-wheel-drive system that came with the vehicle.

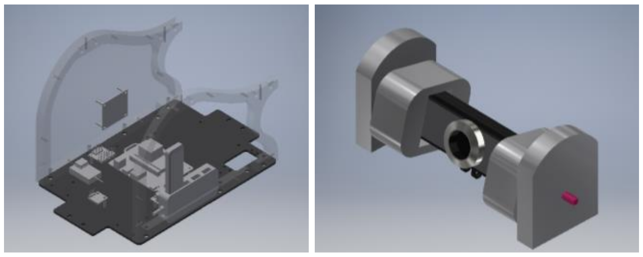


Above: motor mount isolated (left) and in car (right)

Steering on Macaroni is achieved using the already existing servo drive and linkages available on this Traxxas model. Also similar to the robot used last year, this servo was mounted on the top mount plate of the robot.

C. *Mounts for Electrical Systems*

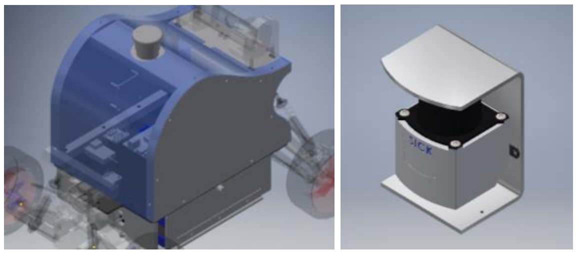
To improve navigation capabilities, the robot has been updated with new electrical equipment; the camera, LIDAR unit, main computer, microcontroller, and encoder each needed new mounts. To accommodate these, a second electrical mounting plate also made of aluminum now sits below the original one. On the lower plate the batteries, motor controller, and encoder are mounted, while on the top plate the LIDAR, Intel Joule, and other related electrical components are mounted. Mounting was achieved using screws and Velcro straps. The camera is now mounted on the top-front of the roll cage of the car instead of higher and farther back. This makes better use of the new camera's wider field of view. Also, the mounting assembly was modified to provide more security and ease of adjustability. The new camera mount fully encases the camera to provide full protection from every angle. The LIDAR is bolted to the top plate of the front suspension assembly. For added protection, the LIDAR mount also includes a sheet metal plate made from aluminum. The plate curves back and around the LIDAR, balancing protection with not blocking the view of the sensor. Finally the emergency stop was moved to the top of the robot and mounted on the roll cage while maintaining the competition specification keeping at least the minimum distance from the ground.



Chassis upper mount plate (left) and camera mount (right)

D. *Robot Protection*

With the increase in sensitive electronics comes the requirement for a more stable and robust way of protecting these components. The protective subassembly of this robot is comprised of two pairs of side panels made out of 1/2in aluminum (AL6061). Shown in figure x, the side panels are bolted to the top and bottom plate and serve to protect the robot from damage from both sides. The bottom side panel is equipped with hinges to allow easier access to the bottom plate. Also included as a protective surface against minor splashes and light rain, a layer of clear polycarbonate plastic was added to the top of the robot.



Roll cage (left) and LIDAR protector (right)

III. Electrical Design

The remodeled electrical system of Macaroni centers its design around the Intel Joule module. Motor control, including PID and E-Stop circuitry, is operated separately using a microcontroller and a radio board assembled on an external protoboard. Data is collected from a LIDAR, IMU, encoder, and camera, mounted on the robot. All on-board components are powered by two separate 11.1V Li-Po batteries: one to drive the motor and the other to power the electronics. This particular design was chosen as it protects the sensitive electronics and sensors from any electrical noise produced by the motors with the added protection that, should the motor stall and cause electrical failures, all data would be safe to better diagnose the problems.

A. *Component Breakdown*

Intel Joule 570x

The Intel Joule handles all vision processing and path planning for the robot. Acting as the central hub for computation, the Joule interfaces with the on-board sensors and ICs using Serial, I2C, and USB communication protocols provided by the Intel Joule Expansion Board. We switched to the Joule from an Intel NUC to save weight and get access to lower-level communication protocols.

USB 3.0 Hub (Amazon Basic USB 3.0 Hub)

The USB 3.0 Hub takes in data from all of the USB devices (the camera, the LIDAR, and the USB drive) and channels them to the single USB Type A port on the Joule Expansion Board.

USB Drive (Corsair 128GB Voyager GTX)

The USB drive stores all the data from the sensors and stores it for future use. This lets our software team run virtual tests by playing back data from special ROS storage files.

LIDAR (SICK TiM551)

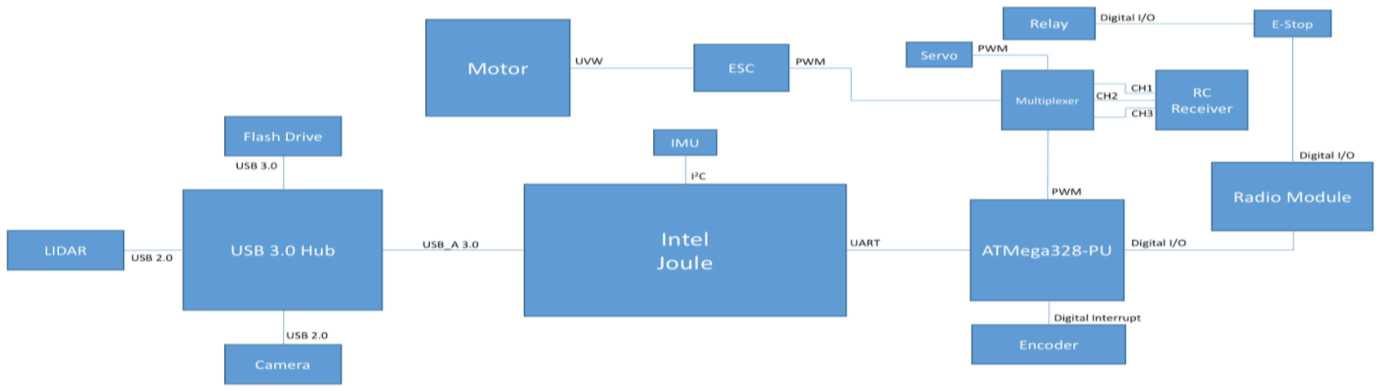
The LIDAR interfaces directly with the Intel Joule Module through a micro USB connection. The LIDAR has a range of 0.05 m to 10 m with a statistical error of +/- 0 mm and a refresh rate of 15 Hz, which gives Macaroni a very clear picture of its 2D surroundings. The sensor is powered directly by the 11.1V Li-Po batteries.

Remote E-Stop (Zrabra XY-R02A)

The Zrabra radio module, chosen for its simplicity of design and ease of use, is mounted onto the protoboard. It communicates with a two button remote that triggers two corresponding output pins. One pin is tied directly to the ESC control while the other is connected to the microcontroller in order to track the status of the E-Stop trigger.

Microcontroller (ATMega328-PU)

All motor control and tuning is handled by the ATMega328-PU. Flashed using the Arduino bootloader, the microcontroller utilizes PID to adjust and maintain the target heading. The status of the multiplexer, E-Stop, and the current speed of the robot is sent to the Joule through a serial communication interface.



Functional block diagram of the Electrical System

Relay Control Board (Beefcake Relay Control)

The Relay Control Board connects directly to one of the digital lines on the Zrabra radio board. This takes the lower voltage (5V) signal of the radio board to either allow current to pass to the motor or cut the circuit in the event of an E-Stop trigger.

IMU (MPU-9250)

Connected to the Joule directly through an I2C port, the nine-degree-of-freedom IMU reads the linear acceleration, angular acceleration, and orientation. The unit includes a gyroscope, accelerometer, and magnetometer.

Encoder (E4T-100-250-S-H-D-B)

We use an optical quadrature encoder with a configurable number of cycles per rotation. The encoder is attached directly to the motor drive axle, meaning a 1:1 gear ratio from the speed of the motor.

Camera (Genius WideCam F100)

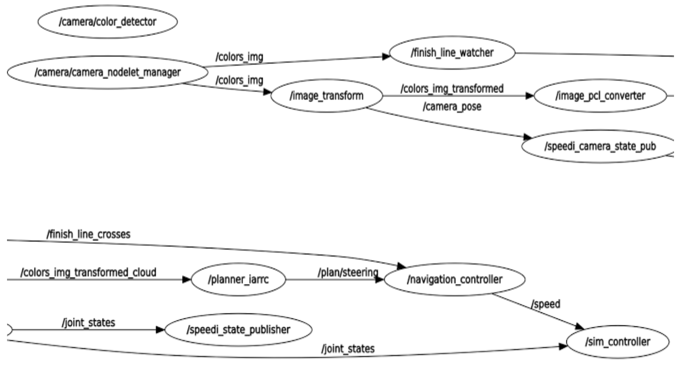
The camera is a wide-angle 1080p HD webcam which communicates to the Joule through a USB connection through the USB Hub. The data collected is then used in vision processing on board the Joule for identifying obstacles.

Multiplexer (Pololu 4-Channel RC Servo Multiplexer)

The multiplexer controls whether the servo and ESC pwm commands that ultimately get passed on to the motor come from either the Joule in autonomous mode or the RC receiver when in manual mode. The selection pins are handled on a third channel from the RC receiver, allowing the manual driver to control whether the robot is in remote or autonomous control. This allows for greater flexibility during testing in order to determine both mechanical and software issues.

IV. Software Design

Our software is a distributed system that runs in several processes on the CPU. Each process is a node in a network linked by publishing and listening for data messages. This is made possible by ROS (Robot Operating System), which is a robotics-focused distributed computing platform that runs in Ubuntu. We chose ROS C++ over ROS Python for its performance. The ROS network runs on an Intel Joule, a 4-core compute module designed for robotics and IoT applications. This Joule communicates with the rest of the hardware by writing information over a serial port to an Arduino programmed in the Arduino language. When the physical robot is not available, we test our code on virtual circuit and drag racing tracks, which are simulated using Gazebo. This software integrates with ROS, allowing us to mock realistic sensor input and robot physics to see how our code behaves. This has allowed us to fix many bugs and even to tune parameters.



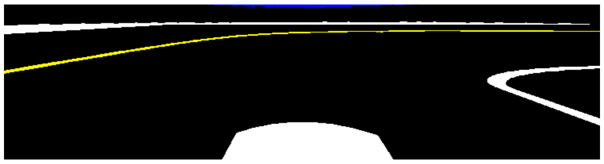
Above: example ROS graph using our IARRC nodes, shown in two parts

Several subsystems within this network of ROS nodes and messages turn the front-facing camera feed into a power signal to the motor and steering servo interfaces: obstacle detection, path planning, and vehicle control.

A. *Sensing: Obstacle and Stop Light Detection*

Our camera is a 1080p HD webcam that is connected via USB to the Joule. Using OpenCV, we identify parts of the image that match color criteria that we tuned to match obstacles. It generates an image where just these obstacles are highlighted, and this image is converted to a point cloud akin to the one a LIDAR unit would generate. Later in the pipeline, the planner uses this localization to determine the course of action. Using this process we are able to identify where the obstacles are and publish this information to our path planner.

The stop light detector works by computing the change in pixel intensities between frames. When the light turns green, the part that was formerly red becomes much darker and the green part becomes much brighter. When the change in color meets a certain threshold, the node indicates to the rest of the system to start working.



Processed image with IARRC track lines highlighted

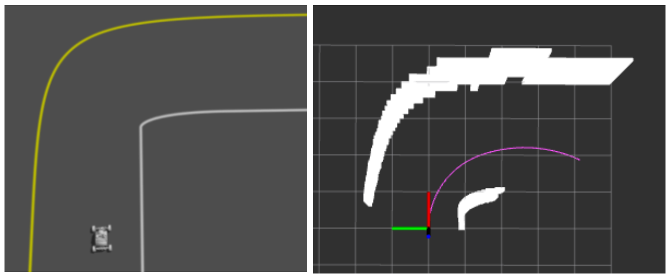
B. *Path Planning and Collision Avoidance*

We use a stochastic path planner to extract a good route through the detected environment. First, random samples are taken from a normal distribution centered at zero. These values are interpreted as potential steering angles, which the planner uses to project the motion of the car through the most recent point cloud. We use an Ackermann steering model and assume perfect traction. Instead of projecting the path using one steering angle for a certain length of time, our planner can break the projected path into an arbitrary number of segments, each with a different steering angle (we have found that using two path segments to provides optimal behavior). The robot's speed at each point in the path is determined directly from steering angle: at maximum turning, the speed is set to one tenth of its straight-line value.

Once the many random paths have been extrapolated, each is assigned a cost as a path integral of position costs. Each position's cost is calculated using the inverse of the distance to the nearest detected obstacle and the vehicle's speed. The paths with the lowest cost are those that balance fast (i.e. straight) trajectories with maximal distance from obstacles. To efficiently find the closest obstacle in the point cloud, we use FLANN (Fast Library for Approximating Nearest Neighbors) to run an efficient nearest-neighbors search.

Once all costs have been calculated, those not close enough to the best cost are filtered out. The remaining costs are clustered using our own implementation of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm. In order to be clustered, the paths are encoded spatially using the different steering values along the path. The cluster of paths with the lowest average cost is chosen, and the average of all first steering samples in these paths is used to set the robot's target speed and angle.

This planner has shown very good results at navigating quickly through tight spaces formed by static obstacles. The emphasis on speed in the cost calculation means that even with plenty of space, the robot will try to clip the apex of the corner as if it had planned a racing line. The algorithm does not make special affordances for moving obstacles; it will not drive behind another vehicle, as it expects to hit it.



Path planning in Gazebo. The pink line represents the path.

C. *Vehicle Control*

After the path planner determines an optimal steering heading and speed, this information is written over serial to an Arduino. The Arduino reads the state of the E-Stop and servo multiplexer. If the E-Stop is not activated and the multiplexer is set to automatic control, the Arduino will compute the appropriate motor PWM given the heading and speed. This is done using a PID controller where error is computed in terms of speed. The PID constants were tuned by first finding a P constant that approaches our setpoint asymptotically and then a D constant that reached the ideal speed. This PID controller is a surface agnostic feedback loop, meaning that its only parameters are a setpoint, a current speed, and PID constants.

As a result, we can do high speed acceleration and braking regardless of the surface. The Arduino is consistently writing back information gathered from the encoder as well as the state of the multiplexer over a serial communication to the Joule.

This concludes the written report for RoboJackets RoboRacing.