Sedani Design Report

Georgia Institute of Technology - RoboJackets

Matthew Barulic mbarulic@gatech.edu
Evan Bretl evan.bretl@gatech.edu
Sahit Chintalapudi schintalapudi@gatech.edu
Varun Madabushi vmadabushi3@gatech.edu
Joseph Spall IV jspall3@gatech.edu

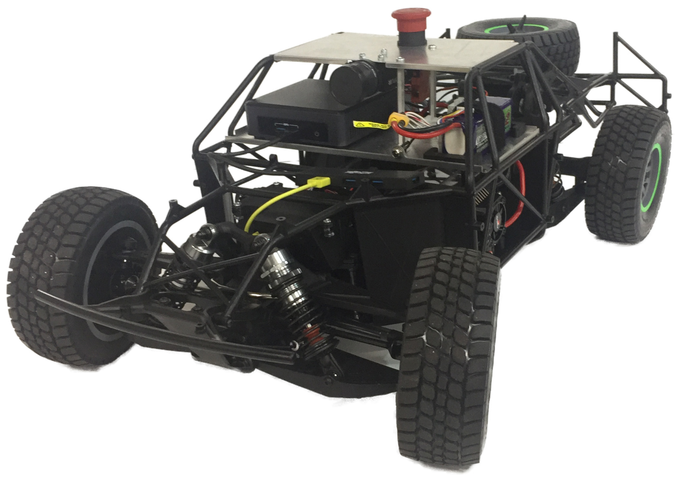
Fig. 1. RoboJackets' new RoboRacing platform, Sedani.

***Abstract*—This report introduces Sedani, Georgia Institute of Technology (Georgia Tech)'s entry into the 2018 International Autonomous Robot Racing Competition (IARRC). We discuss what advances our team has made between 2017 and 2018 and how these advances allow us to tackle the multiple challenges IARRC poses across controls, localization, and perception.**

I. Introduction

RoboJackets is the Georgia Tech 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 200 dues-paying 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 coordinating with each other and the members of their own subteam.

This year we are bringing a new robot to the University of Waterloo, which we call *Sedani* (Fig. 1). This new platform attempts to address usability and power problems with our previous robot, *Macaroni*. The details of Sedani's mechanical and electrical design are described in Section II and Section III respectively. Our autonomous racing software has also seen significant improvements in our steering and stoplight detection algorithms, as described in Section IV.

II. Mechanical

A. System Overview

The mechanical platform is charged with the task of quickly and safely carrying the computing and sensor equipment necessary for autonomous racing. Our team has once again taken the general approach of modifying a remote controlled car to serve as our mechanical platform. This section will cover the design considerations that motivated the changes we've made this year, and then discuss each part of the mechanical design in more detail.

B. Design Considerations

As a small team, RoboJackets has traditionally tried to start with a stable platform by modifying a commercial remote control car to fit our autonomous equipment. While this stock chassis takes care of a lot of the more complicated mechanical problems for us (ie. suspension, drive train, and steering), it also introduces pre-existing geometry against which we have to design our custom portions. In particular, Macaroni, having been built from a 1/10th scale stock chassis, experienced many problems related to trying to fit a lot of computing onto a relatively small vehicle.

Macaroni's small stock chassis required our team to replace the bulk of the chassis's structure with larger, custom parts. This gradually led to a design where each subsystem depended on the others. Trying to upgrade any one piece (ie. changing the computer), required a heavy redesign of a large number of the car's parts. Additionally, because these parts were structural to the car, they were cut from metal for robustness and relative ease of manufacturing in our shop. This resulted in a vehicle that was much heavier than its chassis was designed for. It became difficult to find motors and gearing rated for a 15 pound, 1/10th scale car.

Our mechanical team decided on two changes to our approach to help us avoid these problems when designing Sedani: a larger stock chassis and a more modular design. We broke the system down into two primary modules: the *chassis module* and *compute module*. The chassis module houses all of the electronics necessary to drive the car manually, including any emergency stop and autonomous-manual control switching functionality. The compute module houses the high-level computing and sensing hardware. Sections II-C and II-D describe each of these modules in detail.

Fig. 2. The Losi 1/6th Scale Super Baja Rey, the stock chassis for Sedani.

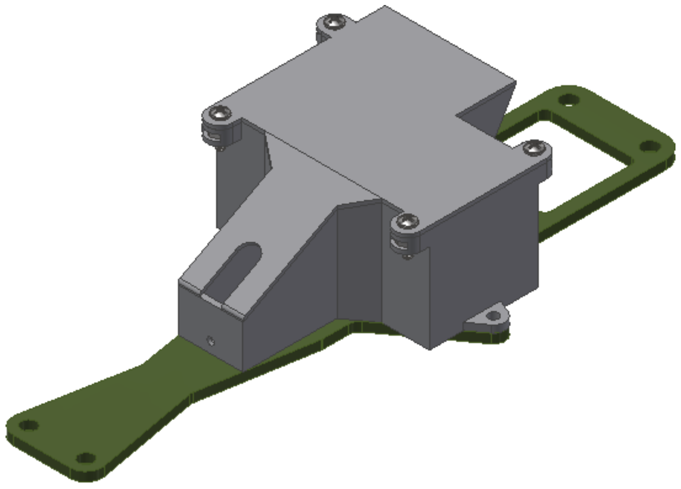
Fig. 3. Digital design of Sedani's chassis electronics enclosure.

C. Chassis Module

Sedani's chassis is based on the Losi 1/6th scale Super Baja Rey 4x4 Performance Desert Truck [3], shown in Fig. 2. Inspired by the work of Georgia Tech's AutoRally project [4], we were searching for an affordable car close to 1/5th scale. The Losi truck is a brand new product (released in March 2018) which offers a lot of great racing technology at an affordable price. Our second criteria was ease of modification. We were looking for a chassis with a variety of places to mount extra hardware and built robustly enough to support extra weight. The Losi truck is built using largely flat plastic and metal parts that are screwed together. We took advantage of this by strategically replacing a small number of the plastic parts and using the existing screw points.

The chassis module required two modifications to the stock chassis: an enclosure for the chassis control electronics (Fig. 3), and a replacement battery compartment with mounting holes for the vehicle's electronic speed controller (ESC) (Fig. 4). Both of these new parts were manufactured using a combination of 3D-printed PLA plastic and laser-cut Acrylic. These techniques allowed us to quickly and easily create parts that match well with the existing geometry of the car. Because neither of these components bear any significant loads during a vehicle crash, we are comfortable making them out of plastics.

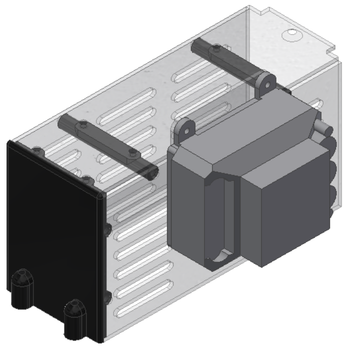
Fig. 4. Custom battery enclosure with ESC mounted.

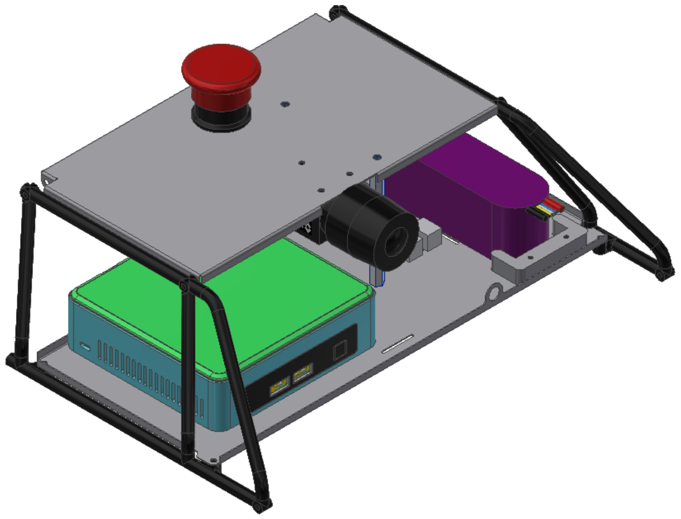
Fig. 5. Digital design of Sedani's compute module.

D. Compute Module

The compute module, shown in Fig. 5, required the replacement of a few cosmetic and roll cage components of the top of the vehicle with an enclosure for the computing and sensing hardware. As this is a larger modification than either of the chassis enclosures and functions as part of the vehicle's roll cage, the computing enclosure needs to be much more robust. To this end, the compute module was designed as two, folded sheet aluminum plates. The lower plate provides a surface on which to mount our computer, computing battery, and various power-management hardware. The upper plate provides a mounting point for the camera and emergency stop button, while adding structural integrity to the surrounding roll cage sections.

The right angle flanges along the edges of the plates provide good mitigation against the plates bending or deforming. They also provide surfaces for panel-mounted components, such as the external barrel connector for the hot swap power system, described in Section III-C1. The compute module is further reinforced by a set of vertical aluminum standoffs that span between the two plates.

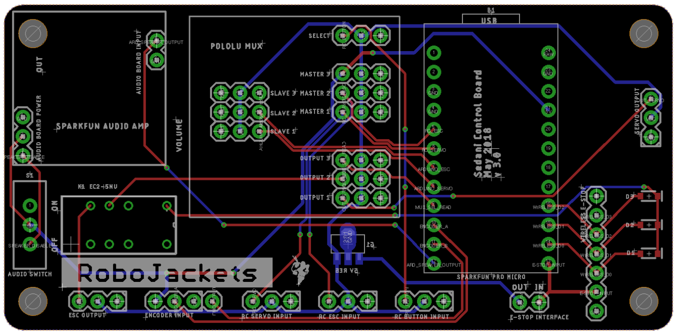

Fig. 6. The custom PCB for interfacing with the chassis electronics system.

III. Electrical

A. System Overview

The system is split into two distinct categories: the chassis and the computing electronics. Chassis electronics handle all drive control, steering control, emergency stop, state notification, and remote data collection. Computing electronics involve mainly the Intel NUC and FLIR Blackfly Camera. Both systems have separate batteries, allowing for each to run independently and to reduce noise between each. The split of the systems allows for maintenance on each individually without requiring the presence of the other system, allowing the mechanical team to work on the chassis and the software team to work on the sensor suite without interfering with each other. This modularity was a goal from the start of the project.

A large focus this year was on pre-manufactured components. In the past, the electrical team heavily experimented with custom printed circuit boards, which allowed for greater specialization for the use case. However, this experimentation lead to a less reliable system. The goal has become to make custom electronics at a slower pace, replacing a few subsystems at a time. This allows for the system to remain stable while improving efficiency and reducing size over time. Additionally, buying pre-made components allows for multiple back-ups that can be replaced in the event of component failure driving or testing.

B. Chassis Electronics

The chassis electronics allow for direct control of the robot. A custom printed circuit board (PCB) was designed to interface with all of the desired electronics in the chassis electronics system, allowing for smooth integration of the various components (see Fig. 6). The majority of components are through-hole, allowing for ease of construction and higher quality of manufacturing. All components are labeled with silk-screen for ease of identification and orientation.

1) Drive and Steering: The drive and steering system use the stock motor, servos, and ESC that came with the Losi car. The drive motor is a Dynamite Fuze 1/6 1200KV brushless motor, designed to carry the robot at stock up to 50 mph. The motor is much larger compared to Macaroni, hopefully compensating for the additional weight from the added hardware. The steering servo is a Spektrum S904 waterproof digital servo, allowing for high torque and fast reaction time. The Dynamite 160A Waterproof Electronic Speed Controller (ESC) serves as the bridge between the microcontroller or remote to the motor. Our speed control relies on a feed-forward relation between PWM input and driving speed. Sedani's ESC, unlike Macaroni's, has a built in failsafe, so a loss of signal connection causes the ESC to brake. The emergency stop system takes advantage of this and is elaborated on later in section III-B4.

2) Microcontroller: For a microcontroller, the team choose a 5V/16MHz Sparkfun Pro Micro. The package allows for a small, reliable Arduino-compatible system. The board is based on the ATmega32U4 chip and has an on-board micro-USB for programming and serial communication. The USB connection receives power from the NUC and allows for information to be passed to higher level control software, discussed further in section IV.

3) Remote Control: The Pro Micro interfaces with the ESC and steering servo through a Pololu 4-Channel RC Servo Multiplexer board. The Pololu Mux board breaks out the 74VHC157 Quad 2-Input chip, allowing for either control based on PWM output from the Pro Micro for autonomous driving or from a PWM signal from an HK-GT2B radio receiver for remote control during mechanical testing and data collection. The board also allows for power distribution for the servo and HK-GT2B radio, running at 6V.

4) Emergency Stop: The chassis electronics system interfaces with a Simple 315MHz Latching RF L4 receiver for emergency stopping and state change. The wireless remote has three outputs that both connect to digital inputs on the Pro Micro and all feed in to the driving element of a DFRobot Gravity relay. The connection to the Pro Micro allows for different firmware states to be selected: autonomous course, autonomous drag race, and manual with data collection. The Gravity relay connection closes the normally open path between the ESC output signal from either the remote control or the Pro Micro and the actual ESC, defaulting to an off-mode in the event that the Pro Micro fails. The mechanical relay allows for complete isolation. A standard pushbutton e-stop exceeding the minimum 30 cm also exists for redundancy.

5) Notification: The last element of the chassis electronics is the newly-added sound notification system. The system involves a simple 1.5 Watt speaker connected to a SparkFun Mono Audio Amp Breakout board. The SparkFun Audio Amp serves as a break-out board to the TPA2005D1 chip with 10k potentiometer volume control and shutdown input for deactivation. Simple note patterns can be played with the Arduino tone library, allowing for notification of state changes such as starting autonomous mode or firmware upload. This system is convenient for quick understanding of simple variable changes without a visual display.

C. Computing Electronics

The computing electronics mainly focus on much higher level control of the robot (Fig. 7). A custom computing box contains the power Hot Swap Board, the Intel NUC, and

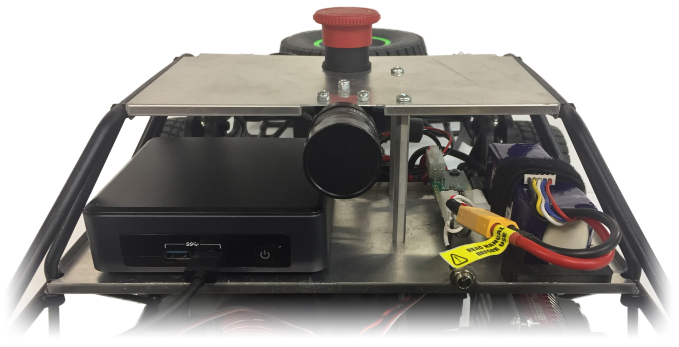

Fig. 7. The installed compute module.

the FLIR Blackfly Camera. The system only has a serial connection from the Intel NUC to the chassis electronics through USB, being completely isolated otherwise.

1) Hot Swap Power System: The entire system is powered through the mini-box.com Y-PWR Hot Swap board. The board allows for the robot to be powered by either a wall-socket AC to DC power converter or the on-board battery. This swapping ability gives greater flexibility to the software team when it comes to making more lengthy code changes while being powered by the 120 VAC outlet without having to replace and charge batteries as often or requiring the NUC to be powered down between sessions.

2) Intel NUC: The Intel NUC acts as the central compute platform. The NUC has an Intel 8th generation i5 laptop processor and 8GB of RAM in a compact 4 inch by 4 inch package. This allows for high levels of computing with a small physical footprint. A more in-depth description of the software use of the Intel NUC occurs in section IV-A.

3) FLIR Blackfly Camera: The FLIR Blackfly USB3 Camera is the main sensor on the robot. It has a computar T2616FICS 1/3" monofocal manual iris lens at 2.6mm. It is a rolling shutter camera to allow for higher frame rate, minimizing blur from incidents such as driving over bumps or moving at high speeds. The camera is essential to the main neural network training and control, being further described in section IV-B2.

IV. Software

A. System Overview

Our software is executed on an Intel NUC running Ubuntu. We use ROS, an open-source robotics framework with C++ and Python bindings, to build the racing logic. ROS is designed to facilitate communication between modular units of code working in tandem to drive the robot. It comes with many advantages, including out-of-the-box sensor support and easy integration with simulation. Our code is open-source under the MIT License and is available at https://github.com/robojackets/roboracing-software In IV-B, we discuss RoboJackets' biggest software advancement between this year and the last by replacing our previous lane detection and path planning logic in favor of a deep learning based approach. In IV-C, we discuss how we handle stoplight detection (challenge 3 of IARRC) and in IV-D, we discuss how we safely stop the vehicle after the finish line.

B. Deep Learning Based Autonomous Driving

The main means through which the robot decides how to steer is through an artificial neural network. Neural networks approximate functions on input data by applying successive "layers" of linear and nonlinear functions. Each layer consists of multiple "nodes," and the value at each node is a weighted sum of values from the previous layer passed through an activation function. In other words, if x_i is a vector of inputs or values from a layer, g is an (nonlinear, monotonically increasing) activation function, and W is a matrix of learned weights, then the values at layer 1 would be $x_1 = g(Wx_0)$. Networks with many of these layers are categorized as deep learning. They are the state of the art in a wide range of practical AI tasks, in particular those tasks which require processing images or other high-dimensional data. Our neural network implements "end-to-end" control [1], meaning that the input to the network is a frame from the forward-facing camera feed and the output of the network is a steering angle. The target speed of the vehicle is determined from the steering angle via the linear relationship $speed = v_0 - \alpha * |steer|$, where v_0 represents the straight-line speed and α represents a proportionality constant that is tuned to slow down the car a reasonable amount when turning.

1) Rationale: Driving almost entirely via neural network control is a major improvement from our entries in past years. Historically, we have performed color matching on the image feed to identify safe and unsafe regions and used that result to perform local path planning. The decision to change is largely based on brittleness of the color matching with respect to varying lighting conditions. The colors of both obstacles and non-obstacles change for a variety of reasons during competition, making color-based recognition very difficult and time-consuming with hand-designed heuristics. The ideal obstacle-identifying heuristic does not depend much on the lighting of the scene. Conveniently, neural networks have been shown to excel at tasks where processing rich data (e.g. images) is difficult via explicit programming. Given training examples across multiple lighting conditions, a neural network is able to learn lighting-independent rules for steering the vehicle. Another benefit of the new machine learning approach is in development time. The task of the software team in preparation for competition is the relatively fast and easy process of collecting training data for the neural network. Finally, using a neural network greatly simplifies the stack of software that we maintain for competition, as the entire system runs in one Python process. Choosing to use a neural network does have at least one drawback; it is a "black-box" with almost no affordance given to explaining why a certain steering angle was picked for a given input. We consider the benefits in processing rich data and easing development to be worth any drawbacks.

2) Neural Network Architecture: Our neural network is implemented in Keras, a Python library built on top of the popular TensorFlow machine learning framework. It is a convolutional neural network (CNN), drawing its structure from popular image categorization networks. Ours is much simpler than those at the state of the art in image classification; we are working with less data, and so the extra complexity is not worthwhile.

- Input: The input to our network is a color image of dimension 48 rows by 128 columns. The low resolution of the image allows for a smaller, faster-to-train network that is more likely to key on large, important details than small, less-relevant ones.
- Convolutional Layers: We use a convolutional structure in the first two layers, meaning that the learned parameters in the network comprise linear filters that are convolved (passed or slid) across the image [2]. This technique simultaneously reduces the number of parameters to train and increases performance on spatially-correlated data. These two layers have 32 and 64 3x3 filters, respectively. Both use the ReLU (Rectified Linear Unit) activation function $g(z) = \max(0, z)$. Max-pooling is also employed to down-sample the filtered image after each convolutional layer.
- Fully-Connected Layers: The following two layers of the network are fully-connected layers with 128 and 32 nodes, respectively, also using ReLU for the activation function.
- Output Layer: The output of the network is a selection of one of five categories: hard left, easy left, straight, easy right, or hard right. Discretizing the output in such a way simplifies the learning task, and a simpler task means that good results can be found from less data. This output layer is implemented via the softmax activation function, which outputs probabilities over five output nodes that together sum to one.

3) Data Labeling and Training: For many deep learning tasks, such as image classification, labeling enough training data with the correct outputs is a long and painstaking process. In contrast, data collection for our end-to-end network is straightforward. The robot is driven around a track by human "expert" while a script saves pairs of images and steering angles to the robot's storage. At training time, these examples are played back in random order, and the AdaDelta optimization algorithm updates the parameters of the network to bring its output closer to what the human driver chose. As such, our system falls under the "supervised learning" class of machine learning systems. Training goes through every example in a specified set several times before its accuracy ceases to increase.

We employ several techniques to get more robust results from limited training data. First, we use dropout regularization before the first fully-connected layer and before the output layer. Dropout randomly, temporarily zeros out weights in the network. Since a node in the network does not have access to every piece of information from the layer before it, dropout encourages the network to look at overall shapes rather than

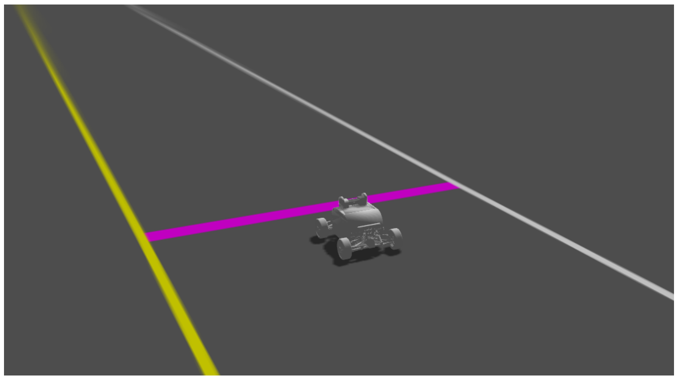

Fig. 8. The Gazebo simulator as used for data collection and model validation.

individual pixels. Second, we apply random horizontal flips to training images (and the corresponding steering angle labels) with a random chance of 40 percent. This helps remove bias from the training examples, which may have more turns in one direction than the other. Third, we add random Gaussian noise into the pixel intensity values of images during training. This has an effect similar to dropout in that it discourages the network from focusing on single pixels or other small features.

4) Testing Methodology and Informal Results: We validated our model both in simulation and on a real robot platform. We used Gazebo, a robot simulator with ROS integration, to run the first tests of the deep learning system (see Fig. 8). It performed well when trained and tested on simulated tracks, including one with false boundary lines and other adverse features. The network could ignore some of these features in a way that our old hand-designed detectors were never able to do. We then trained the network on data from driving our robot around a small test track. Using around 10 minutes of data collection, the neural network was able to guide the car around a track that had a different layout than the one used for training.

C. Stoplight Detection

The stoplight is bright enough that each camera we have used is fairly overwhelmed or saturated. The red and green lights show up as white-tinged-red and white-tinged-blue, respectively. We have solved this problem by looking more for the location of the lights than their color. First, the algorithm finds the difference between the current frame and one it observed a fraction of a second ago. Image matrices are computed for the loss in the red channel and the gain the blue channel (the blue color is more prominent than the green). The elements of the blue gain matrix are shifted by a fixed amount so that they are expected to overlap the red. Then the two matrices are multiplied element-wise and smoothed with a large linear filter. Finding a large value in the resulting matrix means that a large area lit up blue (green) below another large area that became less red, and these two areas are roughly the right distance apart. If a large value is found in the result matrix, a start signal is sent to the rest of the system.

We have tested this system using the robot's camera and the same stoplight model that is used in competition. In addition, we have practiced tuning the parameters of the algorithm in preparation for adjustments at competition. These parameters include the radius in pixels of the stoplight and the distance between the centers of the two lights.

D. Finish Line Detection

Our finish line detector exploits the bright, unique color of the finish line by performing color matching. We prefer the HSV (hue, saturation, value) color space over RGB because it is more robust to lighting conditions. Pixels are identified as potentially part of the finish line if they fall within a range of hues near magenta and have relatively high saturation and value. An image is classified as containing a finish line if enough total pixels match the color criteria and enough of those pixels line up horizontally. For example, a magenta flag on the side of the track is not classified as a finish line because it takes up a small horizontal space in the image.

REFERENCES

[1] Y. LeCun, U. Muller, J. Ben, E. Cosatto, B. Flepp,"Off-Road Obstacle Avoidance through End-to-End Learning" Neural Information Processing Systems, 2005.

[2] "Feature Extraction Using Convolution", Unsupervised Feature Learning and Deep Learning Tutorial. [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

[3] Losi Super Baja Rey product page
`http://www.losi.com/Products/Default.aspx?ProdID=LOS05013T1`

[4] AutoRally
`http://autorally.github.io`