Building a virtual environment to train autonomous RC cars

Huan K. Tran

tran0966@umn.edu

Abstract

Training autonomous cars demands massive amounts of labeled data that is currently obtained by collecting real driving footage. In this project, I plan to create a low-cost and simple framework that allows users to generate labeled datasets from a simulated environment, train and test their own autopilot in a virtual environment, upload the trained pilot to a modified RC car and let the AI control the car. The solution includes modifying a 1:18 scale RC car to increase its control granularity, building a simulated environment using Gazebo/ROS framework, and applying computer vision, machine learning techniques to close the bridge between the virtual world and reality with a simple color filter OpenCV program.

Problem description

With the development of AI technology and increases in the speed of hardware, autonomous cars that can fully control themselves have nearly become reality. However, currently, all the available solutions for self-driving cars can only reach level 3 autonomy, which means the car has an autopilot mode but still needs to be monitored by human drivers constantly. One of the challenges in level up to level 4 autonomy is to collect enough training data to train the artificial intelligence agent (AI). Many companies, for example, Tesla, have developed their own network to collect driving data from their car fleet and use the data as training data to make their autopilot smarter. Despite the massive data produced from the technique, there are many edge cases missing from the collected data which will potentially confuse the AI agent when it finds itself in one of those rare situations. One way to train the AI with these edge cases is to train them in a simulation with a setup environment that resembles the edge cases. This idea is formally regarded as the domain randomization technique, which has been applied in many branches of robotics, especially manipulator control, to train AI agents to execute their tasks in many simulated environments and then let them operate in the real world**.** For example, the paper by Tobin et al. (2017) shows they could train an object detector completely in a simulated

environment and the robot was still able to operate in the real-world environment. Given the evidence of successes from research targeting traditional manipulator robots, is it possible to apply a similar technique on self-driving cars, which is a different robot that relies heavily on computer vision data?

Regarding the current platforms to develop autonomous agents for RC cars, there are several available options. For example, the MIT Racecar is one of the prominent open source projects that enable users to simulate the car in a virtual environment. MuSHR (Srinivasa, 2019) is another open-source project coming from the Personal Robotics Lab in the Paul G.Allen School of Computer Science & Engineering at the University of Washington. There are also commercial projects like the DonkeyCar (Subedi, 2020) that could be trained to accomplish basic self-driving car tasks. Another prominent option is the DeepRacer platform which comes from Amazon Web Services. The development tool and environment of the DeepRacer platform has a very large set of tools enabling users to easily enter the self-driving car realm.

For the DeepRacer car platform, there have been studies to bridge virtual simulation and the real world by applying reinforcement learning to create a sim2real tool that could map simulation world map real world (Balaji et al., 2019). To answer the question of whether these pseudo data could actually help train AI systems to navigate in the real world, the answer right now is yes as there are many papers describing different very different approaches in solving the real-world simulation gap. For example, the paper by Pan et al (2017) attempted and successfully created an image-to-image translation network converting the non-realistic image obtained from the simulation to realistic images resembling the real world. The generated images were used to help AI agents learn to drive in the real world through training with traditional reinforcement learning solutions. Another successful trial comes from Bewley et al. (2019) when they again used an image-to-image translation network to teach an AI agent to drive with no labeled data. More recently, there are also papers from Gupta et al., Zhang et al. (2019), and Raghavan et al. (2020) attempting to achieve similar results by using different architectures like deep reinforcement learning or Cycle GAN.

Speaking about algorithms that could be used to teach the neural network to drive safely comparable to humans, there seem to be two main branches: reinforcement learning and behavior cloning. Since letting a real car be driven autonomously on the roads is a very dangerous task, as

it could lead to many unforeseeable behaviors from the car that could eventually harm the human drivers involved in the accidents caused by the self-driving car. A trending way for beginners to develop their autopilots is to implement and test their AI on an RC car. There are many recent papers from the community showing successful attempts to implement the reinforcement learning algorithm (Hossain et al., 2020) and the behavior cloning algorithm (Haji et al. 2019) to drive RC cars.

In this project, I plan to create a framework with basic functions that allow users to generate label data, train and test their own autopilot in a virtual environment as well as deploy the trained pilot to a modified RC car that allows the trained network to control the car based on feedback data from the rear camera of an Android cell phone mounted on the car chassis. To allow flexibility in the modification of the virtual environment, I plan to use the Gazebo framework to generate the virtual environment of which configuration could be changed arbitrarily. In order to give a proof of concept for this project, I plan to use a modified 1:18 RC car. The car will have a system to capture images and can be controlled through a UART interface by sending commands to a microcontroller. Moreover, I also attempt to bridge the gap between the simulation and the real world by applying a simple color filter on collected images from the simulation and the real world to make them identical to the eye of the AI agent.

Team member and management

Team member

- Huan Kim Tran

Management

- Due to COVID-19 restriction, all the setups described in this paper have been implemented at my home

Tasks

1. Modifying the 1:18 RC car:

- Designing power and control circuit

- Design and implement a new controllable and variable steering angle mechanism
- Design and Implement a new controllable and variable power throttling mechanism for rear wheels
- Microcontroller firmware to interface and control car's throttle and steering angle
- Android application to bridge between high-level software and firmware
- Chassis reinforcement to enable the RC car to carry more weights

2. Building the simulated environment:

- Generating a 3D model of the RC car in the Gazebo environment
- Exploring Gazebo configuration to create a world that mimics the real-world environment
- Exploring Gazebo plugin programming framework
- Exploring Ros programming framework
- Creating Gazebo model plugin to connect ROS middleware and provide accessibility to control the 3D RC car model as well as recording images from the camera sensor
- Integrating computer vision bridge program to let the neural network control the 3D model while taking filtered images from the Gazebo environment

3. Building the computer vision and machine learning framework to create a bridge between the real world and the simulated world:

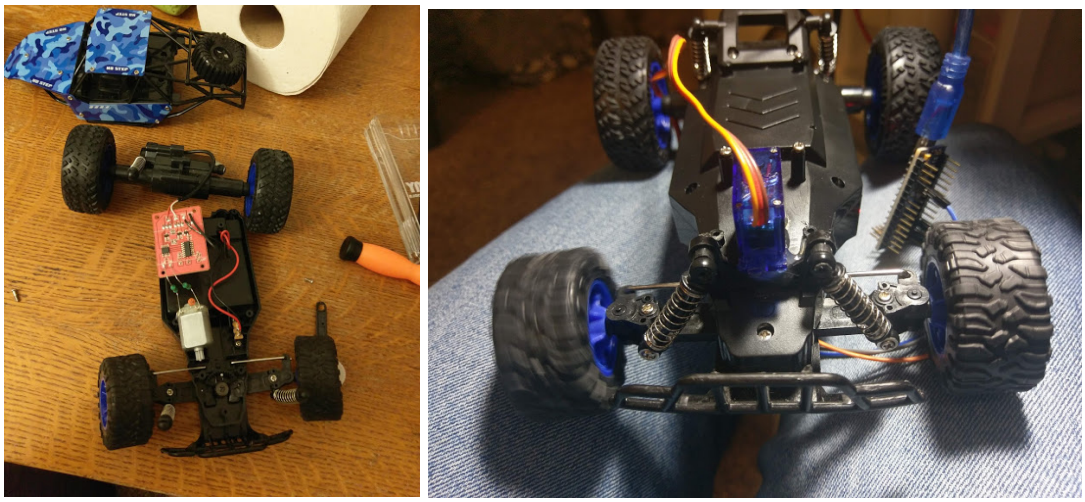
- Creating a filtering application that could filter out desire cues in simulation-generated images and real images captured from cell phone camera mounted on the RC car
- Exploring convolutional neural network architecture to create a deep neural network for behavior cloning implementation
- Exploring TensorFlow Lite and TensorFlow Mobile to deploy the trained neural network onto the Android mounted on the RC car

- Explored OpenCV library to implement image filtering application on the Android phone

Results

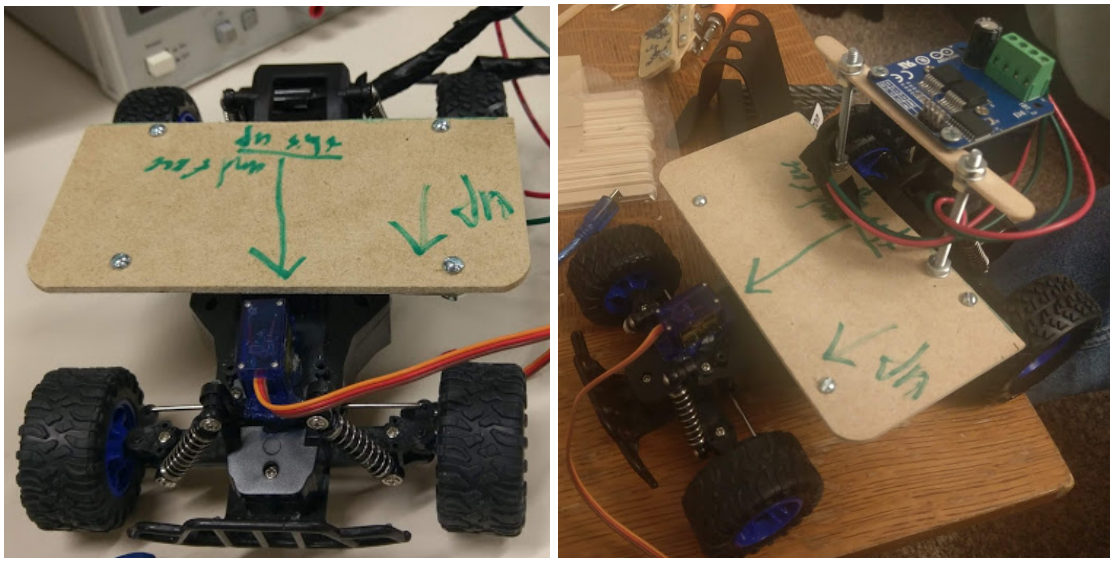
I. RC car modifications

The RC car used in this project was a 1:18-scale RC car which came with an FM controller that only allows 4 commands: full throttle forward/backward and full steering left/right. In order to increase the granularity in the controlling level, the car has been modified to allow 200 different throttling levels (both forward and backward) and 200 different steering angles (left and right). The modification of the RC car structure to meet the desired features took longer than expected due to the incompatibility of the original steering mechanism with the sg90 servo motor.

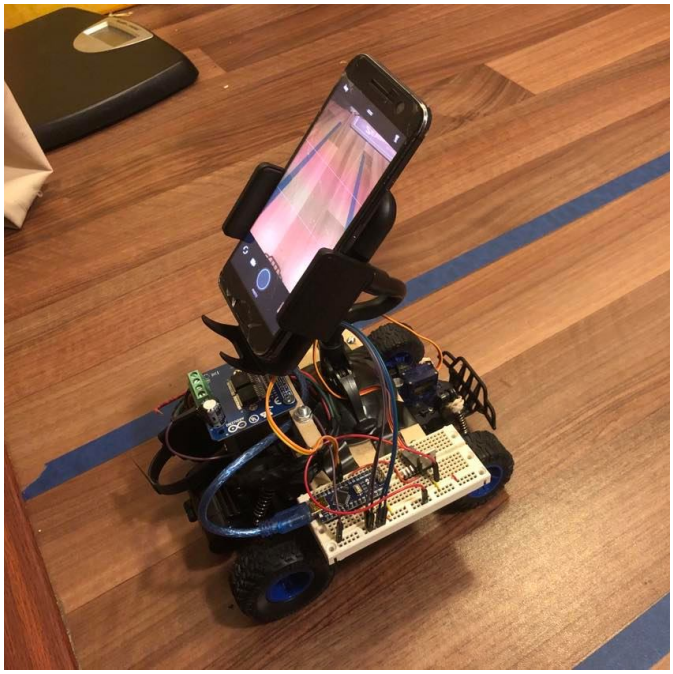


Original car steering mechanic vs new mechanic implemented with an sg90 servo

Besides increasing the granularity in the steering level, the rear motor controller has also been replaced to add the ability to throttle at different levels. The high-current motor driver BTS7960 was employed to achieve the desired granularity.

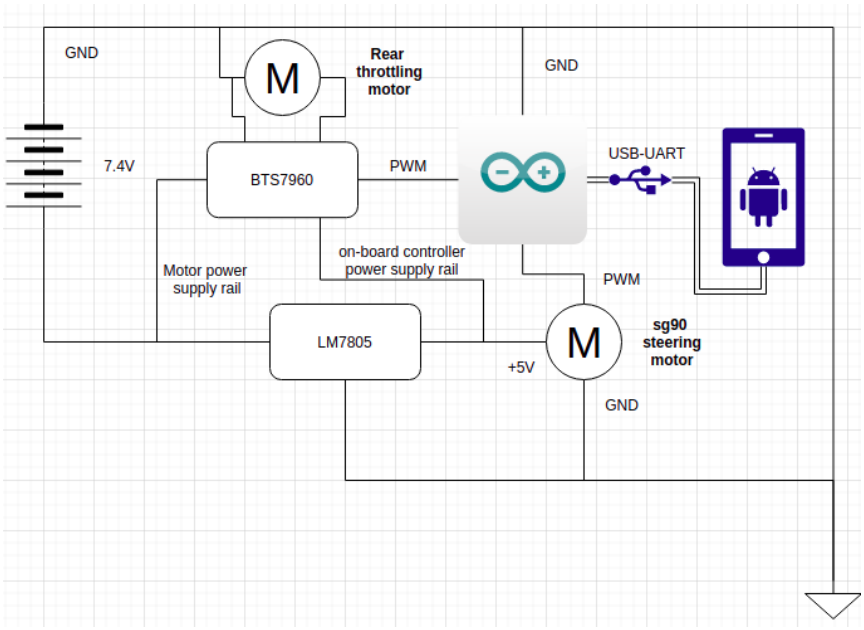


Wooden board mounted to the car body to provide more carrying space (left) and new motor controller mounted (right)



Final RC car setup with the Android phone as a high-level software processor mounted on top

Once all the new mechanics have been installed, the RC car needs a central power rail and circuitry to deliver the appropriate power as well as control signal from the central controller to each actuator.

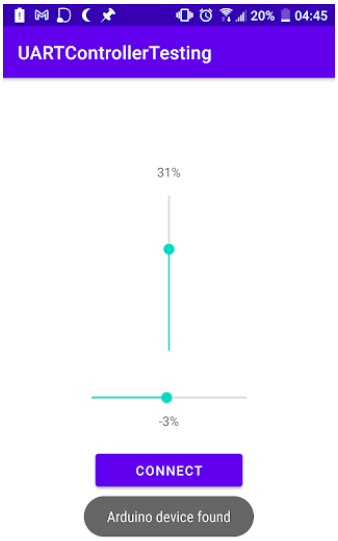


Wiring of power rail and signal path between different components

The bridge between software and hardware in this project is an Arduino nano board that produces PWM signals to the sg90 servo and the BTS7960 motor driver to control the steering angle and speed as well as direction, respectively. The Arduino board provides a software interface that the high-level computer system which processes computer vision-related tasks will use to control the mechanics of the car. To reduce the cost for a portable computing system that is small and lightweight enough to be carried by the RC car but still has enough computing power and software support to run computer vision-related tasks, an HTC-10 cell phone running Android OS 7.0 has been employed. The interface provided by the Arduino takes input parameters through the onboard USB-UART connection, in which the Android phone plays as a USB host providing power and signal through the USB OTG cable. The simple interface only has 2 commands with the following syntax:

- s steering_level_value;
- t throtlling_level_value;

steering_level_value and throtlling_leval_value are integer numbers between -100 and 100, in which -100 means full left turn or move backward with full speed, and 100 means full right turn or move forward with full speed. Each command has to be terminated by a semicolon to prevent ambiguity in parsing commands and increase the robustness of the communication in case of data corruption. To test the ability to command the RC car's mechanics through the Arduino board, an Android app with a simple user interface allowing users to control the throttling and steering levels through two sliders has been successfully made.



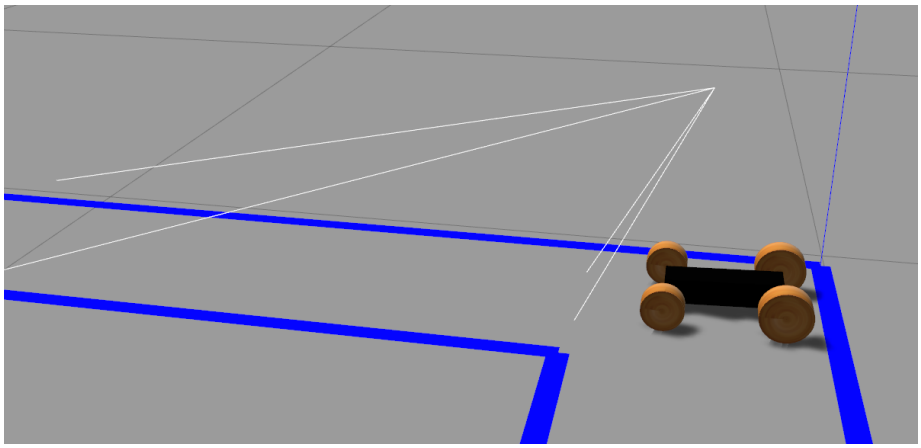
The interface of the simple Android application used to test the Arduino-Android interface

II. Building simulated environment

The simulated environment for the RC car was built using Gazebo, ROS, and OpenCV framework. There are 3 main components: the simplified 3D model of the RC car, the

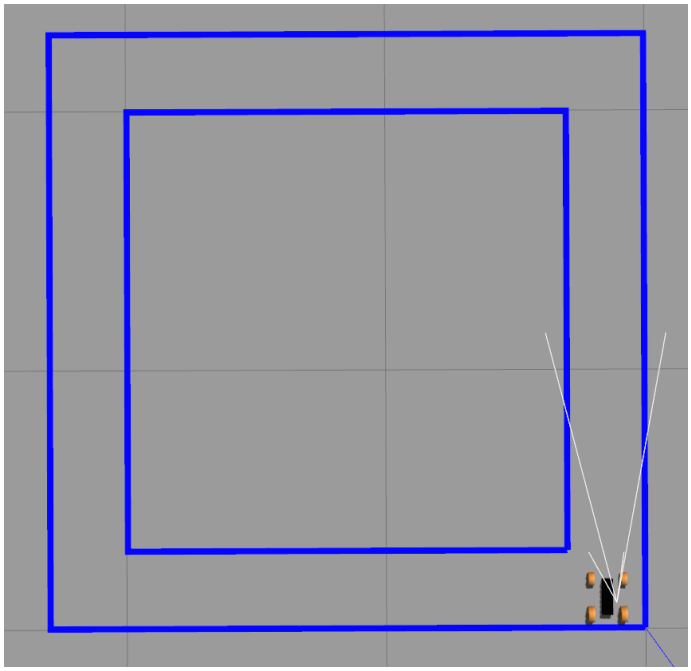
Gazebo world with which the 3D model interacts, and the software comprising Gazebo plugins, Ros nodes, and OpenCV programs, designed specifically for the RC car 3D model. These packages run behind the scene to provide a framework for users to control the 3D model, generate image training data for the AI agent and test a specific version of the AI agent in the simulated environment.

The RC car was modeled in Gazebo using a simplified version of itself with 5 links (4 wheels and 1 chassis), 4 joints, and a camera oriented and positioned at the same angle and height as the camera of the HTC10 in the real-world setup. The model's camera was finely tuned so that its parameters (field of view, image resolution, noise, etc) match with the real rear camera of the cell phone.



Simplified 3D model of the RC car in Gazebo

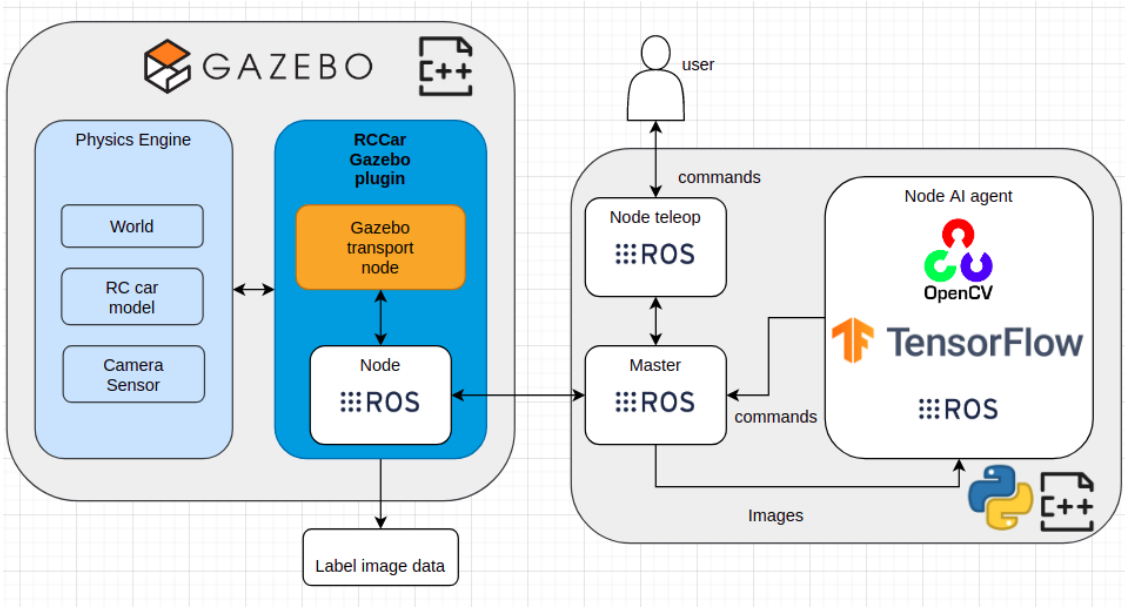
The world the 3D model interacts with can be arbitrarily constructed with an unlimited number of models (if it is still the range of the computer memory), however, in order to reduce the complexity, a simple square lane with two blue marker lines on either sides have been used.



The simple world used to teach the car to stay inside lane in Gazebo

The 3D model of the RC car was built with an integrated Gazebo plugin designed specifically for the car. The plugin is started along with the car when the model is inserted into the simulated world. Internally, there are two main components running continuously: A Gazebo transportation node and a ROS node. The Gazebo node subscribes to various topics of the Gazebo environment to manipulate the model's link and joints properties like front-wheel steer angle, rear-wheel angular velocity, etc, receive camera sensor captured images, listen to Gazebo physics engine's events. Running along with the Gazebo transportation node is a ROS node that publishes three topics to the ROS network: the ~/RCCar/control/front topic which takes in commands having syntax like the UART steering command, the ~/RCCar/control/rear topic which takes in commands having syntax like the UART throttling command, and the ~/RCCar/camera/top_camera/ topics that published timestamped images taken by the simulated camera mounted on top of the RC car model. With these three topics, any external module having connections to the ROS network would be able to control and manipulate the car model inside the Gazebo environment. It is possible to control everything in Gazebo with the Gazebo

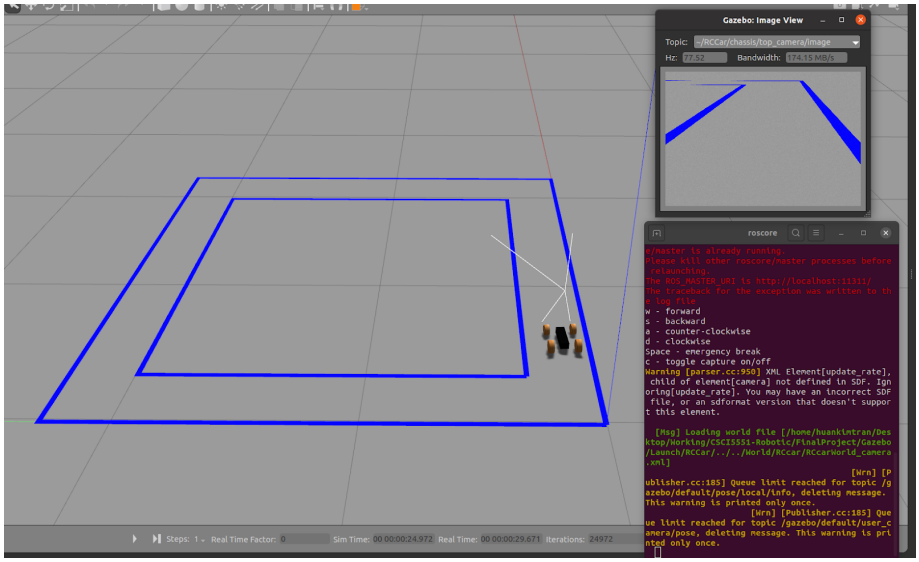
transport node alone, however, the Gazebo framework is very limited in integration capability as it only has support for C++ programming language. Since the ROS framework supports C++ programming language, it can be easily embedded into the Gazebo plugin. Thanks to this integration, the RCCar framework becomes not only more modular as simulation and logic units have been separated but also more flexible as users now can apply supporting libraries written in both Python and C++ programming languages. ROS framework itself supports easy integration to 3D model with Gazebo ROS plugin, however, this method only works with 3D models generated by URDF source code, while the RC car model generated in this project was built from SDF source code, thus the reimplementation of the Gazebo transport node.



The internal mechanic of the plugin embedded inside the 3D RC car model

With the modular structure, the RC car framework can allow any agent to control the 3D model, this means the controller can be a human or an AI agent. To accomplish data generation tasks, a ROS teleoperation node given controllability to a user has been built. Using this node, the user will have total control of the RC car model steering angle and throttling level. Moreover, the user can also toggle the data collector module on and off to start and stop the recording of camera sensor images labeled with frame counter value

and steering wheel angle. The future plan for the framework is to have a ROS AI agent node, which subscribes to the camera sensor topic to receive the current view of the RC model in Gazebo world, feed the received image to a convolution neural network to generate appropriate commands to ride the RC model and keep the model inside the lane.

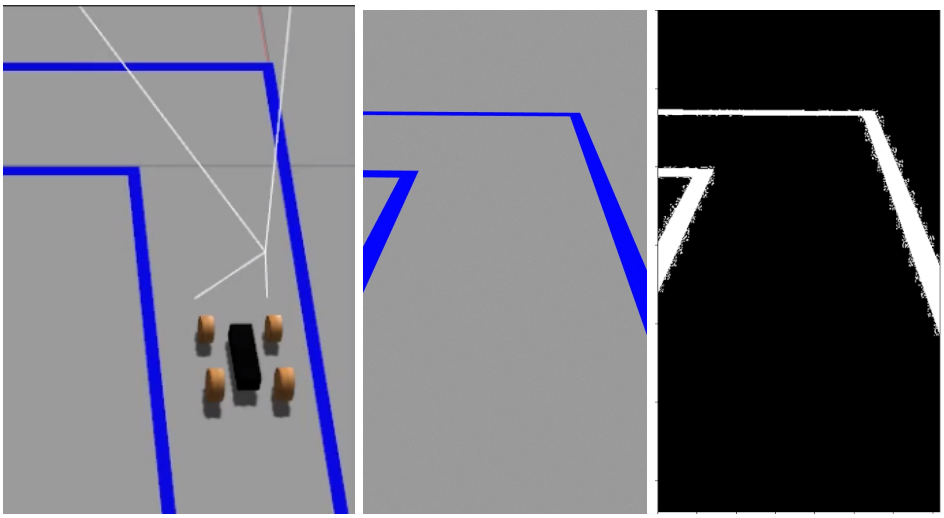


The look of the interface with ROS teleoperation node terminal and view from camera sensor

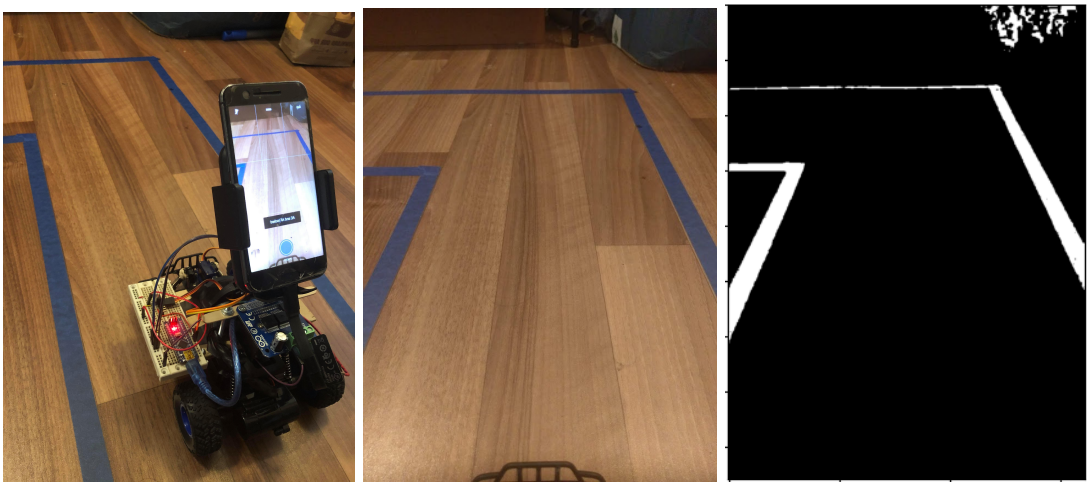
III. Building computer vision and machine learning framework

With a working simulated environment that could generate an unlimited amount of labeled training data, all the data would be useless if there is no way to fill in the gap between the real world and the simulated one. There are a lot of techniques that could be used to create the real-world-to-simulation bridge, nevertheless, since the complexity of the algorithm implementing this bridge tend to be proportional to the complexity and the degree of details of the environment with which the real model interacts, in this project, the objective of the training was kept simple: “ Training an autopilot that takes in images of blue marked lane and output steering angle to keep the car inside the lane .” Given this objective, the setups of the real and simulated environment are very simple, thus the bridging algorithm. The proceedable zone is always marked by two blue lines on two sides, therefore, the converting algorithm could be a color filter that cherrypicks pixels

with the color of the lane marker lines. This filter brings both the real images and simulation generated images into a common space.



View from the camera sensor after applying the filter

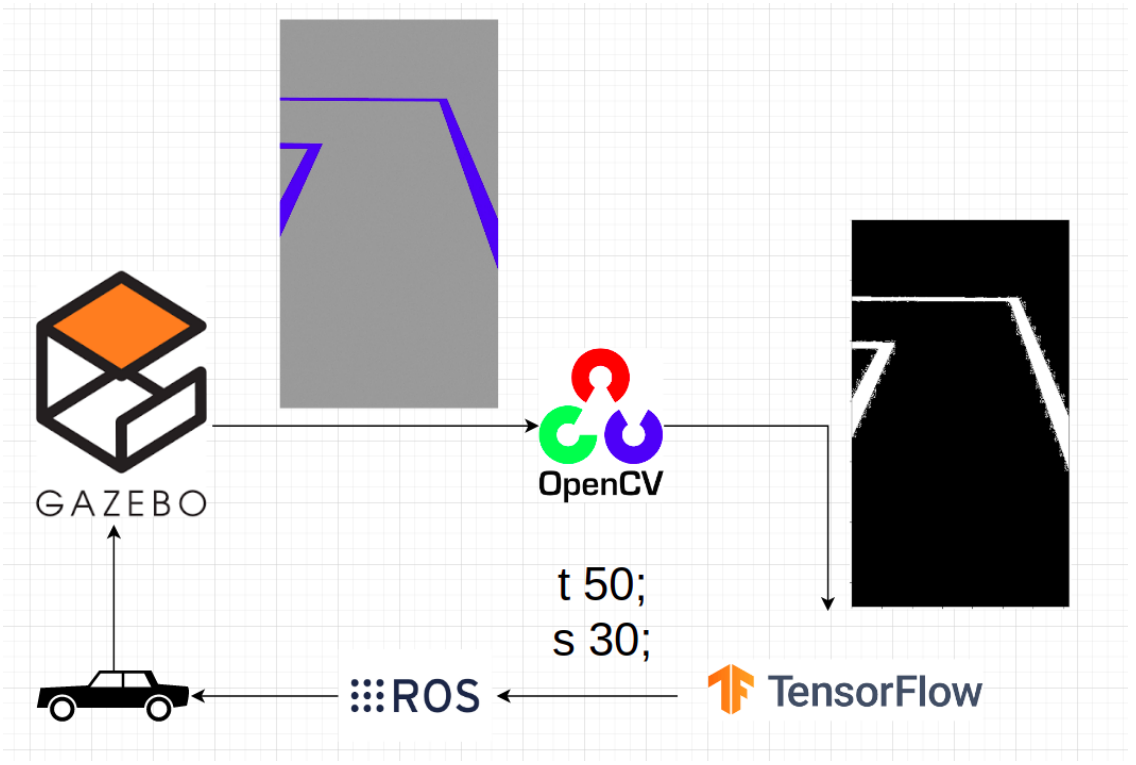


View from the image capture by the cell phone after applying the filter

Future plans

I. Simulated environment

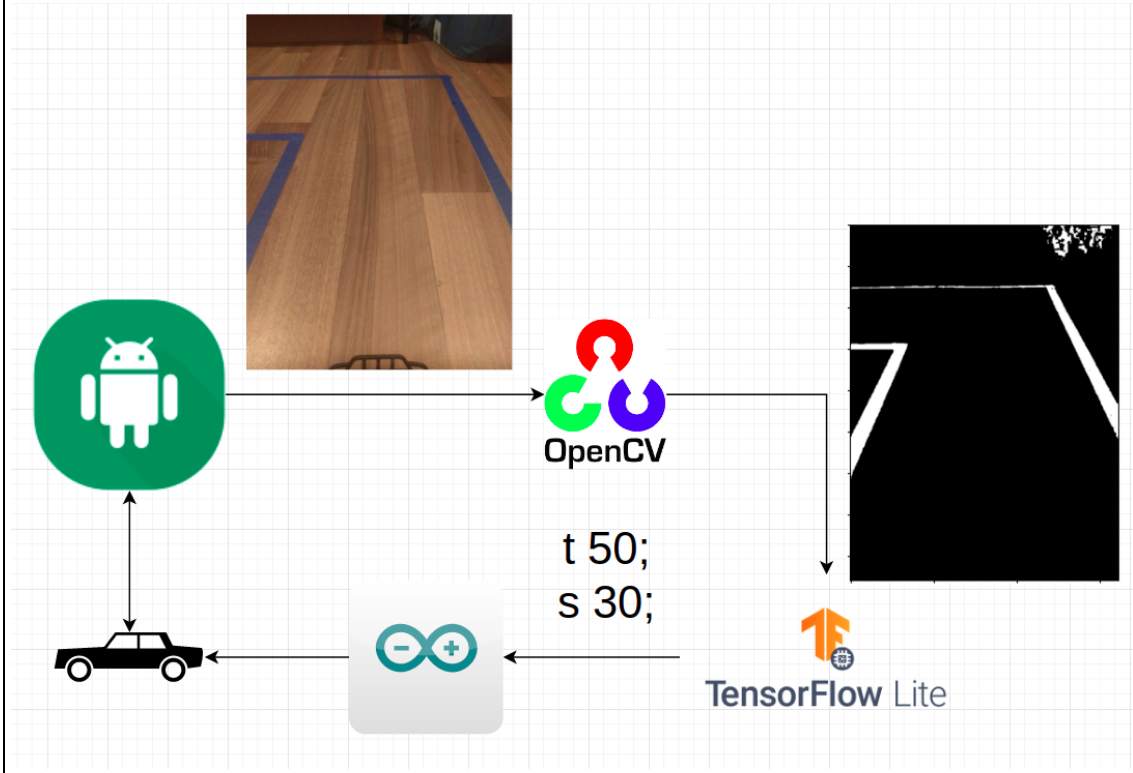
For the simulated environment, one of the important features that are missing from the framework is the ROS AI agent node which takes in the image from the Gazebo camera sensor and output steering command to control the 3D model. I plan to implement this feature by creating a neural network with the TensorFlow library and training the network with the labeled data generated from the framework.



The simulated world loop

II. RC car

Although the framework is now able to generate useful data, it is still missing the ability to allow the trained AI model in the simulated environment to be installed and control the RC car mechanics. In the future, I plan to create an Android app that enables this feature. The development of the app should be smooth since all the libraries have support for Android OS



The real-world loop

References:

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017, September). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 23-30). IEEE.

Srinivasa, S. S., Lancaster, P., Michalove, J., Schmittle, M., Rockett, C. S. M., Smith, J. R., ... & Sadeghi, F. (2019). Mushr: A low-cost, open-source robotic racecar for education and research. *arXiv preprint arXiv:1908.08031*.

Subedi, Subrat, "AI in Robotics: Implementing Donkey Car" (2020). IdeaFest. 81.

Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., ... & Calleja, E. (2019). DeepRacer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning. *arXiv preprint arXiv:1911.01562*.

Pan, X., You, Y., Wang, Z., & Lu, C. (2017). Virtual to Real Reinforcement Learning for Autonomous Driving. *Procedings of the British Machine Vision Conference 2017*. doi:10.5244/c.31.11

Bewley, A., Rigley, J., Liu, Y., Hawke, J., Shen, R., Lam, V. D., & Kendall, A. (2019, May). Learning to drive from simulation without real world labels. In *2019 International Conference on Robotics and Automation (ICRA)* (pp. 4818-4824). IEEE.

Raghavan, N., Chakravarty, P., & Shrivastava, S. (2020). Sim2Real for Self-Supervised Monocular Depth and Segmentation. *arXiv preprint arXiv:2012.00238*.

Gupta, A., & Booher, J. CycleGAN for sim2real Domain Adaptation.

Zhang, Q., Du, T., & Tian, C. (2019). Self-driving scale car trained by deep reinforcement learning. *arXiv preprint arXiv:1909.03467*.

Haji, A., Shah, P., & Bijoor, S. (2019). Self Driving RC Car using Behavioral Cloning. *arXiv preprint arXiv:1910.06734*.

Hossain, S., Doukhi, O., Jo, Y., & Lee, D. J. (2020, October). Deep Reinforcement Learning-based ROS-Controlled RC Car for Autonomous Path Exploration in the Unknown Environment. In *2020 20th International Conference on Control, Automation and Systems (ICCAS)* (pp. 1231-1236). IEEE.