Technical University of Denmark

Thomas Passer Jensen

Simulation of the camera-based challenges in the autonomous Shell Eco-marathon

Including ROS migration and updated brake interface

Special course, June 2019

DTU Electrical Engineering
Department of Electrical Engineering

Simulation of the camera-based challenges in the autonomous Shell Eco-marathon, Including ROS migration and updated brake interface

Author:
Thomas Passer Jensen (s134234@student.dtu.dk)

Advisors:
Jens Christian Andersen (jca@elektro.dtu.dk)
Jesper Schramm (js@mek.dtu.dk)
Claus Suldrup Nielsen (csn@mek.dtu.dk)

DTU Electrical Engineering
Automation and Control
Elektrovej
Building 326
2800 Kgs. Lyngby
Denmark
Tel: 45 25 35 76

studieadministrationen@elektro.dtu.dk

Project period:	February 11th, 2019 – June 27th, 2019
ECTS:	5
Education:	MSc
Field:	Electrical Engineering
Remarks:	This report is submitted as partial fulfilment of the requirements for graduation in the above education at the Technical University of Denmark.
Copyrights:	

Preface

This report is the result of a 5 ECTS point special course conducted at the Technical University of Denmark. The project is carried out as a member of DTU Roadrunners.

Thomas Passer Jensen

Contents

1 Introduction

The autonomous ecocar competes in the Autonomous UrbanConcept category of Shell Eco-marathon.

In 2018 the team had a very successful competition, winning both the Autonomous UrbanConcept and the Internal Combustion Engine UrbanConcept categories. The goal since then has been to identify and mitigate the weaknesses of the autonomous system.

To work towards that goal, the project description of this special course was written as

> *The DTU Ecocar has been augmented with autonomous capabilities by equipping sensors and actuators, enabling it to win the Shell Eco-marathon Autonomous UrbanConcept category in 2018. The aim of this project is to continue the development on the autonomous control systems, focusing on the brake control for parking, the simulation environment and increasing robustness of the navigation software. The goal is to improve the performance of these systems before the upcoming Shell Eco-Marathon 2019 competition for self-driving vehicles.*

Since then, two bachelor projects were started by other students. One project with the purpose of designing a closed-loop controller for parking, with the objective at stopping at a certain position. The other project with the goal of increasing the robustness of the navigation software.

Because of this, the purpose of this special course shifted to work mainly on the further development of the simulation environment and performing important practical tasks, while also guiding the new students to understand the autonomous systems and how to interact with them.

This year the autonomous competition took place in Oss, Netherlands while the mileage competition will take place in England in July. The competition this year was an opportunity for the team to test out some new algorithms for navigation. The results from the autonomous competition this year in

Figure 1: Team photo from the Autonomous UrbanConcept competition in Oss, Netherlands 2019.

the Netherlands are not directly related to the product of this special course, and hence will not be elaborated. Instead this report describes the work carried out on the simulation environment, software stack maintenance and brake interface. Each section is divided logically into the following sections:

- **Section 2 - Simulation environment**: Two new challenges was developed for the simulation environment to allow testing of the challenges where the camera is needed. Furthermore, an additional graphical interface was developed for the simulation environment.
- **Section 3 - Migration to ROS Melodic Morenia**: The whole software stack was updated to the newest version of ROS and Ubuntu.
- **Section 4 - Brake interface**: The interface to the brake actuator was simplified and a new driver was developed.

2 Simulation environment

The simulation environment is set up in Gazebo, an open source robotics simulator. The autonomous ecocar is simulated in Gazebo with the help of a number of plugins writen in C++, which make sure that the simulated ecocar behaves similarly to the real ecocar. The setup of the environment and development of these plugins are described in [1].

In this project the development of the simulation environment continued. The product of this work is

- A Gazebo plugin to draw road markers in simulation.
- A GUI plugin to ease the use of the environment.
- A script to start the simulation GUI together with visualization and plotting tools.

Each of these will be elaborated in detail in the coming sections.

2.1 Road markers

Two of the autonomous challenges in the Shell Eco-marathon 2018 and 2019 included markers on the ground which were a part of the challenge:

- The *Manoeuvrability challenge* (or parking challenge) has a parking spot which is a rectangle on the ground made with tape. The objective is to park with all fours wheels inside the parking spot, and no wheels touching the tape.
- The *Obstacle avoidance challenge* which has gates to go through, and bonus squares on the ground for extra points.

Images from each of those challenges are shown in Figures 2 and 3.

Figure 2: An image of the Ecocar Dynamo in the parking challenge at the qualification in Paris, 2018.

These challenges were not in initial simulation environment because it took some more time to work out. The documentation for Gazebo does not mention any specifics about how to put visuals in the scene. The most straight-forward way to do it, is to have a 3D object with texture and import it. However, this does not make it simple to change dimensions, shape or texture from a configuration file. For the future, we do not have any ideas of what challenges Shell might come up with. It would be very convenient to have an easy way of drawing custom visuals in case a new challenge was presented in the coming years.

Figure 3: An image of the obstacle avoidance challenge at the qualification in Paris, 2018. Notice the closest bonus square in the lower right corner of the image.

To accommodate the creation of these challenges in simulation, another Gazebo plugin was written for drawing markers on the road. The `RoadMarkerPlugin` has a class member function `drawBox`, which enables drawing in any dimension and with any defined material. The way it works is by loading a pre-defined mesh which is built into Gazebo, the "unit_plane". The mesh can be scaled and rotated, and can have texture applied to it (called "materials" in Gazebo). The markers will show up on the simulated camera.

Using the `drawBox` function, the parking spots can be drawn using four rectangles. A texture is applied which resembles the one used at the Shell Eco-marathon competition. The `drawParkingSpot` class member function takes care of this, using parameters which can be defined in the world file.

If the RoadMarkerPlugin is included in the Gazebo world file, the plugin will be loaded. Parking spots can be specified using the parameters listed in Table 1. An example can be found in the world file `challenge3.world`. Multiple parking spots can be specified if needed.

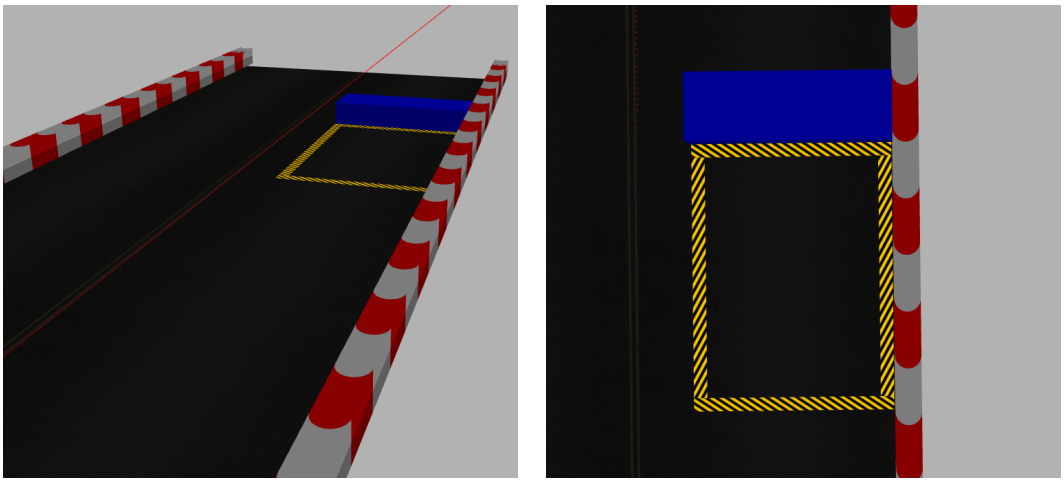
Table 1: Parking spot parameters for RoadMarkerPlugin.

Parameter	**Type**	**Default value**	**Description**
pose	float[3]	[0 0 0]	Parking spot position, given as $[x\ y\ \theta]$.
width	float	1.5	Parking spot width.
length	float	2.5	Parking spot length.
lineWidth	float	0.2	Width of the tape line.

The parking spot graphic in simulation is shown in Figure 4. A blue box is placed behind the parking spot as in the real challenge.

The bonus squares for challenge 4 has a predefined size which is hard-coded in the plugin. The position and orientation can be specified for each bonus square in the world file as well. In Figures 5 and 6 the bonus squares can be seen in the simulated gate challenge.

The RoadMarkerPlugin has a texture defined in a model object, which is used for the parking spots and bonus squares. The same texture is used for both the parking spot and the bonus squares, but the aspect ratio is different. The textures can be found at the path `ecocar_gazebo/models/road_markers`, where more materials can be added for future use. How the texture is applied in the material, is defined in `.material` files in the `scripts` folder. The `road_markers` model is not meant to be inserted into the world, it only serves as a container for materials.



(a) As seen from afar. (b) As seen from straight above.

Figure 4: Screenshot of the parking spot in the Gazebo simulation environment (`challenge3.world`).

2.2 rqt plugin for simulation control

An rqt plugin, named `EcocarSimControl`, was developed for easier control of the simulation. rqt is short for ROS Qt, a Qt-based framework for GUI applications. The plugin can be loaded in rqt together with other plugins, making a dashboard of information which can be quite useful in debugging and testing. Previously you would have to start the simulation and the ROS nodes for autonomous navigation manually from the command line.

The plugin is written in Python and resides in the `ecocar_gazebo_rqt` package. The plugin has functions to start and stop the Gazebo simulation environment. The user can choose which world file to launch, and whether to launch the Gazebo GUI or not. Not running the GUI can ease the computation and helps to not clutter up ones desktop. The plugin subscribes to the ROS topics from the simulation and updates values in the GUI, which is shown in Figure 7. The plugin shows the following information:

- The commanded brake pressure (in green text if the brake actuator is engaged, red if not).
- The reference angle and actual angles for the steering system.
- Collision state of the car (colored circle), showing in red if the car has collided with anything.
- Average velocity of the rear wheels which is used when simulating the motor.
- Motor torque.
- Drag force on the body of the car.
- A debug map showing only the track barriers and the car.

The plugin consists of two main files:
`resource/EcocarSimControl.ui`: an XML file defining the positioning of all the UI elements
`scripts/src/ecocar_sim_control.py`: The Python2 code file.

The `.ui` file is most easily edited using a graphical editor such as Qt Creator.

As of now, the list of available world files is fixed in the plugin. Further development could include the option of launching any Gazebo world file.

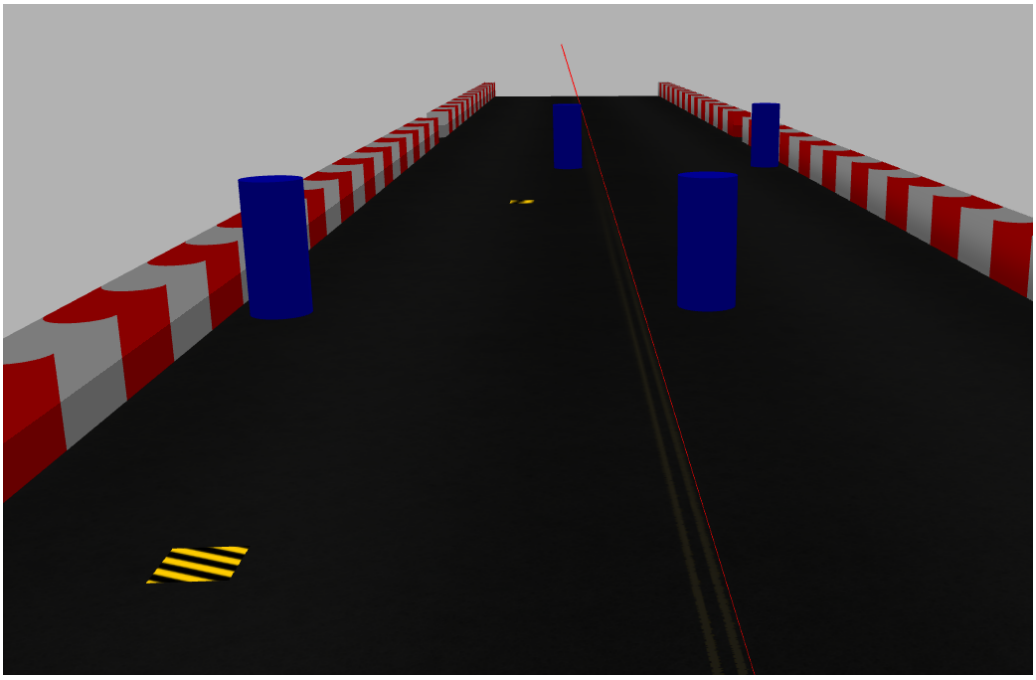


Figure 5: Screenshot from `challenge4.world` showing the gates and bonus squares.

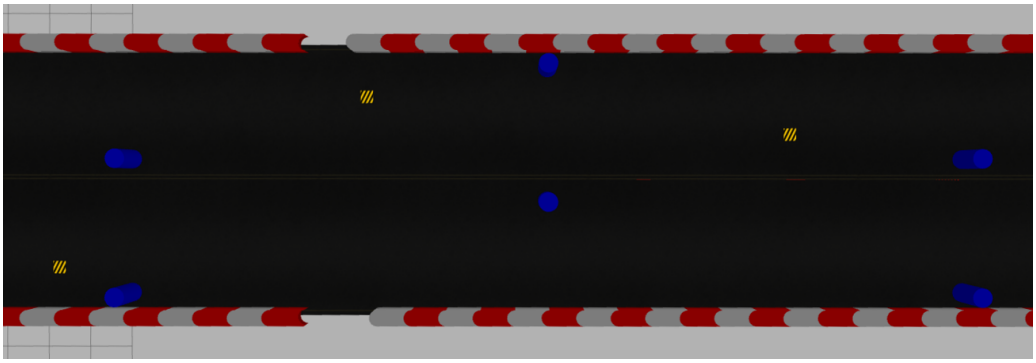


Figure 6: The obstacle avoidance challenge as seen from above.

2.3 rqt simulation setup

A script was developed to start up rqt with the new plugin together with some existing rqt plugins; a camera view (`rqt_image_view`) and a plotting plugin (`rqt_multiplot`).

The rqt configuration is set up in a `.perspective` file. The simulation script modifies the file to load a `rqt_multiplot` configuration which is also in the git repository. It then starts `roscore` and rqt. When rqt is closed, roscore is also stopped.

The full setup is shown in Figure 8. The live plotting has functions to start, stop and clear the plots.

This setup makes it easier for new developers to get into the autonomous ecocar project, because the setup and running of the simulation environment is quite simple:

1. Pull the git repository for the autonomous ecocar.
2. Build the ROS packages in `catkin_ws_sim`.
3. Run the simulation script (`./sim.sh`).

Other convenient rqt plugins are: rqt_launch,

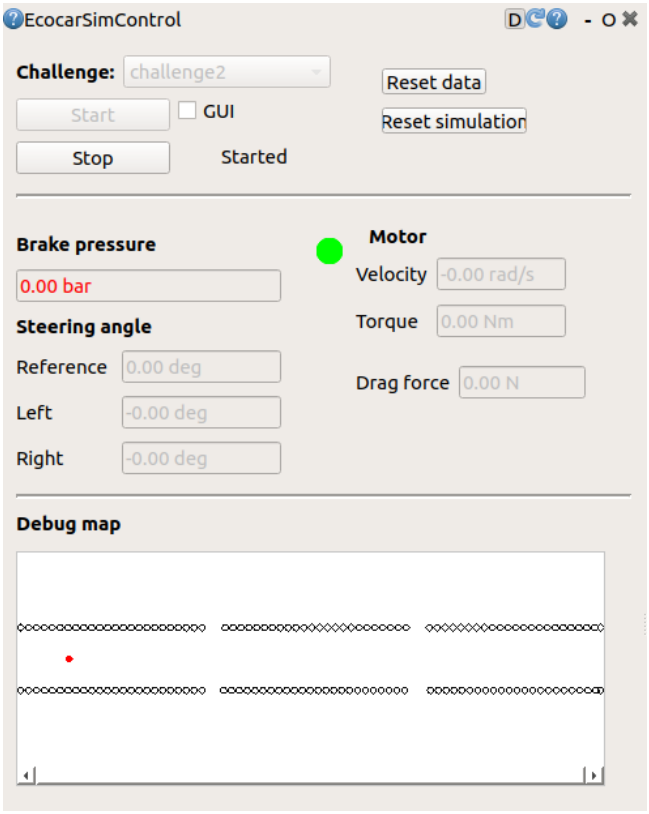


Figure 7: The `EcocarSimControl` rqt plugin. It has buttons to start and stop Gazebo and some debug values from live data. The colored circle (here green) turns red if the car collides with anything.

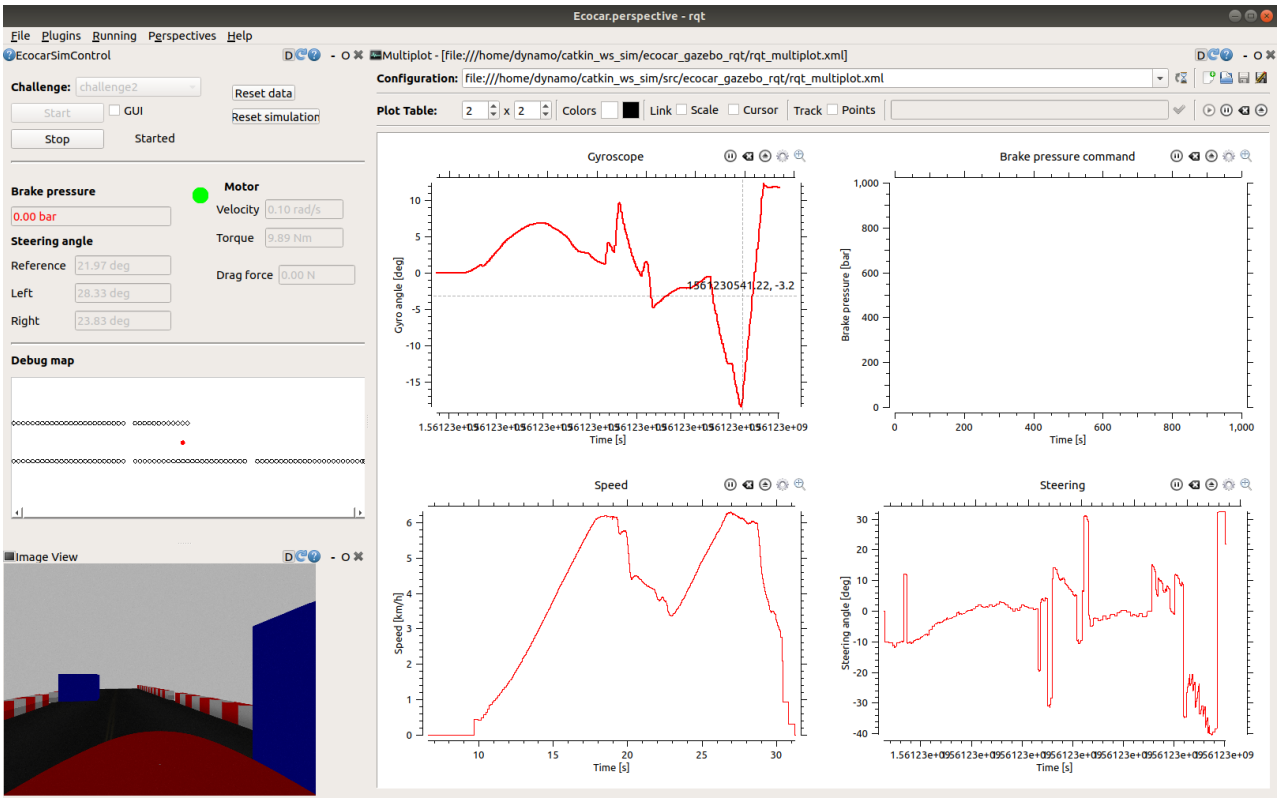


Figure 8: The full rqt perspective. On the left column is the EcocarSimControl plugin with the rqt_image_view plugin. On the right is the rqt_multiplot plugin showing gyro angle, vehicle speed and steering and brake commands.

3 Migration to ROS Melodic Morenia

When the autonomous ecocar project was initiated in 2017, ROS Kinetic Kame was the newest version[2]. Even though ROS Kinetic Kame and Ubuntu 16.04 is still in long term support (LTS) it was decided to upgrade to Ubuntu 18.04 LTS and ROS Melodic Morenia. If we migrate to a new ROS version every two years, the migration will be less effort each time, and we will be able to use new ROS features. It will also be easier for new developers to join the project, because they are more likely to have a newer ROS version installed already.

Figure 9: The logo of ROS Melodic Morenia, the version after the migration.

Like the ROS version is tied to the Ubuntu version, the Gazebo simulation version is also tied to the ROS version, so by updating we also get access to the new Gazebo versions, which are in quite rapid development. An overview of the previous and new versions is shown in Table 2.

Table 2: Versions before and after the upgrade.

	Previous version	**New version**
Ubuntu	16.04 LTS	18.04 LTS
ROS	Kinetic Kame	Melodic Morenia
Gazebo	7.x	9.x

The migration process included:

- Updating instructions for installation from the repository for the main software and the simulation.
- Updating of method for automatic ROS node launching.
- Updating the code of the simulation plugins to work with the Gazebo 9 API.

There was quite a few changes to the Gazebo API, because the `libignition` math packages were moved out of the Gazebo namespace into their own packages.

Fortunately, no updates were necessary to the main autonomous software code for it to compile in ROS Melodic Morenia, however all software has to be tested to ensure no run-time errors or bugs were introduced. This testing is still a work in progress.

3.1 Auto start setup

The boot procedure, that is how every piece of the autonomous system is started when the NUC is turned on, was not well documented. The setup on the NUC in use was investigated to get a thorough step-by-step procedure which could be documented and implemented in the new version.

The auto start sequence is as follows:

1. The dynamo helper node is started from the autostart[3] script in `/home/dynamo/.config/autostart/autostart.sh.desktop` which runs the script in `/home/dynamo/Scripts/autostart.sh`
2. Any other ROS nodes are started using the launch file in `base.launch` in the dynamo package, which is run at boot using the `robot_upstart` ROS package.
3. The dynamo helper ensures that the `teensy_node` is always running. That node receives messages from the steering wheel over CAN bus, including the `autopilot` field. When that field is true, the dynamo helper starts the nodes as specified in [1].

The reason that the `dynamo_helper` node is not also started using the launch file, is that when it is started that way, it is not possible for it to start other nodes using the `gnome-terminal` command, which opens a terminal window with the ROS nodes.

3.2 System install script

During the migration it was noticed how many pieces of the system that were undocumented, and how many steps there was to the installation process. To ease this install in the future, and to reduce the chance of forgetting a step, an installation script was developed, which would setup everything. The script is to be run on a fresh install of Ubuntu 18.04 LTS (with the username `dynamo`), and after it is done, every part of software is set up correctly, including the auto start procedure.

The install procedure is as follows:

1. Update Ubuntu packages in case the installer is not completely up to date
2. Install the lightdm display manager instead of the default gdm3. Auto-login does not work on the NUC in the current version of gdm3, and we cannot login with a password when running without a screen attached.
3. Set up autologin for lightdm by writing a config file.
4. Install the full ROS Melodic Morenia distribution, and the `robot_upstart` package.
5. Install some extra software for development and ease of use; Chromium, Gitkraken, VSCode, TeamViewer etc.
6. Clone the autonomous-ecocar git repository into the home folder. This also includes the autostart file for starting dynamo-helper.
7. Set some scripts from the repository to executable.
8. Build and install libraries that are not from Ubuntu packages: libserialport, libphidget, Velodyne driver, SwiftNav.
9. Build the ROS packages in `catkin_ws`.
10. Setup `robot_upstart`.

The script runs with minimal user intervention, only requiring input to select lightdm as the default display manager and to input the password for the git repository. The password is saved for subsequent git commands.

4 Brake interface

Since Shell Eco-marathon 2018 a new motor speed controller for the brake motor was selected[4] . The chosen controller was the Jrk G2 motor controller from Pololu [5].

The control scheme is as follows. The brake motor pushes on a the brake pistons to control the brake pressure. The brake pressure is measured and received on the Intel NUC in the autonomous system from the Auto board Teensy 3.6. A PID controller in a ROS node then controls the brake motor speed target.

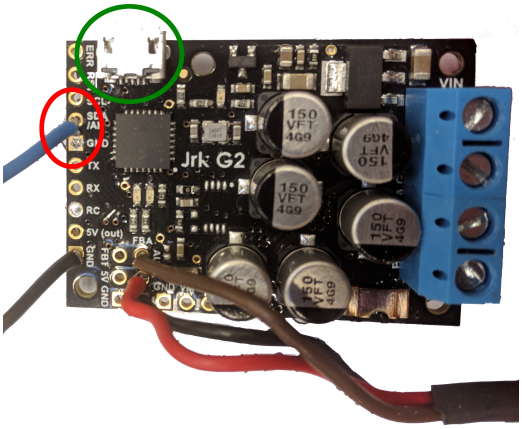


Figure 10: The Jrk G2 motor controller. Encircled in red is where the DAC was connected, and in green the USB connection which is now used.

An image of the brake motor controller is shown in Figure 10. Initially, the interface between the ROS node and the brake motor controller was the same as for the old motor speed controller. As the old motor speed controller only had an analog input, it was controlled through a PhidgetAnalog 4-Output DAC. The new motor speed controller has a USB port which can be connected directly to the Intel NUC, and it was decided to do this, to keep the system simple, with the least amount of devices and points of failure.

For the USB connection the Jrk G2 takes a 12-bit value which is encoded in two bytes as

0xC0 + target low 5 bits	target high 7 bits

[6]

which in C code translates to

```
uint8_t data[2];
data[0] = 0xC0 + (target & 0x1F);
data[1] = (target >> 5) & 0x7F;
```

where target is an integer truncated in between 0 and 4095.

This implementation was tested, using the `libserialport` library which has been used for all previous USB communication for the autonomous ecocar (gyro and connection to the Auto board Teensy). A library function can return a list of connected devices, from which the correct device can be identified. The devices are identified using a hardcoded serial number (ex. ”A700L690”) which can be matched with the list to obtain a path such as `/dev/ttyUSB1` to connect to with another function in the `libserialport` library.

A problem was identified where this would sometimes work and sometimes not, and after some investigation it was found that the Jrk G2 board was showing up with multiple interfaces when connected, which all had the same serial number (because the serial number identifies devices, not interfaces). The `libserialport` library cannot identify interfaces in any way.

The solution was to switch to the more complex `libusb` library. It can identify any possible USB device descriptor, including the `bInterfaceNumber` which showed to be what we needed.

A class in C++ was written to interface with the Jrk G2 using the `libusb` library, such that almost no code had to be changed in the ROS node. The implementation was tested and merged into the main code.

5 Conclusion

This report describes the continued development of the plugins for the simulation environment. It is now possible to draw markers on the track in simulation using a plugin. The plugin has helper functions which takes care of all the details for drawing, and enables developers to add more markers with other shapes and textures in an easy way. The two last known challenges of the autonomous Shell Eco-marathon is now modeled in the simulation using this plugin.

A graphical user-interface was developed to ease the use of the simulation. New developers can more easily get access to the simulation and debug parts of the autonomous systems in simulation before testing on the real car.

The whole autonomous system was migrated to a new Ubuntu and ROS version, which included updating code to the new API and documenting the install process for the autonomous PC.

The interface to the brake actuator controller was changed, allowing us to remove the USB DAC from the car, simplifying the control signal path. During this task, a considerable amount of time was spent debugging the connection until the problem was identified and solved.

The work carried out in this report has much emphasis on making it easier for new developers to join the team and use the systems. The autonomous ecocar project is more likely to succeed in the future if it is easier for students to get started and contribute to the project.

5.1 Future work

All the different challenges that we have encountered in the real competition are now implemented in simulation, however this does not mean that all work is done. Some of the suggestions from a previous report [1] are also still relevant, and below is a few more areas of interest.

- Ensure that the noise levels on all sensors are appropriate and large enough. If the simulations are more realistic, there is less of a difference from testing in simulation to testing in the real world, which means the simulation will be more useful.
- Add more stuff to the simulation in the background, in the sky and behind the track barriers. The simulated LiDAR data has Gaussian noise, but it has no artifacts as seen in real data.

References

[1] T. P. Jensen, “Pre-competition preparations for an autonomous shell ecomarathon car”, Technical University of Denmark, Synthesis report, 2018.

[2] H. S. Høj, “Platform integration for autonomous self-driving vehicles”, Master’s thesis, Technical University of Denmark, Jun. 30, 2017.

[3] J. Palmieri. (2006). Desktop Application Autostart Specification, The GNOME Project, [Online]. Available: `https://developer.gnome.org/autostart-spec/` (visited on 06/23/2019).

[4] R. Barucci, “Further development and optimization of an autonomous braking system for the ecocar”, Technical University of Denmark, Special course report, 2018.

[5] Pololu. (2019). Jrk G2 18v27 USB Motor Controller with Feedback, Pololu Corporation, [Online]. Available: `https://www.pololu.com/product/3148` (visited on 06/23/2019).

[6] *Jrk G2 Motor Controller User’s Guide*, Pololu Corporation. [Online]. Available: `https://www.pololu.com/docs/pdf/0J73/jrk_g2_motor_controller.pdf` (visited on 06/23/2019).