Aalto-yliopisto, Sähkötekniikan korkeakoulu
ELEC-D0301 Protopaja
2019

Final report

Project #08
Speed Demon

Date: 30.8.2019

Kaisu Hiltunen
Taige Wang
Pyry Vaara
Ture Saarniniemi
Olli Paloviita

Information page

Students
Kaisu Hiltunen
Taige Wang
Pyry Vaara
Ture Saarniniemi
Olli Paloviita

Project manager
Olli Paloviita

Sponsoring Company
Futurice

Starting date
4.6.2019

Submitted date
30.8.2019

Table of Contents / Sisällysluettelo

Abstract

The goal of our project is to make a fast 1:10 autonomous car with the ability to drive autonomously around a track using computer vision. The goal was accomplished by using donkeycar system, a dedicated software specialized in computer vision and machine learning. With the support of Jetson Nano mini computer and Intel T265 tracking camera, the chassis was able to learn the driving pattern from manual control, and provide self-driving functionality to the chassis.

The chassis was built on top of Tamiya TT-02 chassis, a 1:10 scale radio-controlled car. Various of 3D printing was deployed on the chassis, including some custom designed parts as well as parts from PELA Blocks design kit, a package which consists LEGO-compatible Parametric 3D Printed Blocks and Gadgets. With the support of TensorFlow, Jetson Nano was able to study distorted and undistorted images captured from Intel T265 cameras, constructing a trained model of the self-driving car.

In the end, the chassis was able to drive on a path that has been trained previously without manual control. The final result of this project was performed by racing on a track mat with competitors from different corporations.

1. Introduction

The goal of the project was to build a 1:10 scale RC-car and teach the car to drive using machine learning. This was accomplished by using donkeycar, an open-source platform for self driving small scale cars, which can be installed on single circuit computers, such as Raspberry Pi or, in our case, Nvidia Jetson Nano.

With donkeycar the car can be trained to follow human behaviour by acquiring data from the car's camera while a human is driving the car around a track. The data consists of pictures and the values of throttle and angle during those pictures. These pairs are then used for training a convolutional neural network with tensorflow.

2. Objective

Our objective was to have a working self-driving car able to navigate through a race track on its own. Our ultimate goal from Futurice was to have a car that manages to beat Futurice Tampere's Markku.ai, in a race. The attributes required for this were speed and consistency, and our car achieved a respectable level on both of them.

The car was demonstrated on the day of Protopaja course's demo day on 22.08.19 by having a race with Markku.ai. This took place in the morning of the demo day, and later in the day we had a stand, where we offered live demonstration of the project.

3. The car

3.1. Chassis

We chose to use Tamiya TT-02 as the base of our car, since it was recommended to us by Futurice. It's a 1/10 scale 4WD R/C-car chassis with resin parts and Tamiya's torque-tuned brushed motor, giving us top speeds of around 25-30 km/h. However, we soon figured out the speed was a lot more than the car could handle on its own, so we ended up using only 20% of the maximum speed.

3.2. Other parts

- Nvidia Jetson Nano
- Intel RealSense T265
- Sunfounder PCA9685
- HK15298 High Voltage Coreless Digital Servo
- BAKTH 7.2V 4500mAh NiMH RC Battery Pack
- TAMIYA 45057 RC ESC TBLE-02s (Electronic Speed Controller)
- Buck Converters
- USB Power Bank
- Ultrasonic Sensors
- Arduino Uno

3.3. Power solutions

The goal was to have only one battery on the car: the ~7.8V main battery, which directly powers the drive motor. This would improve the car's power-to-weight ratio by eliminating two additional batteries, making the car faster. Because the Jetson Nano, Arduino Uno and the steering servo all require 5VDC, the battery voltage had to be lowered. For this purpose we used two separate buck converters: one for the servo and the Arduino and another one just for the Jetson Nano.

In the end the Jetson Nano turned out to require more current (around 3A) than the converter was able to provide (~2A). For this reason we had to use the power bank in addition to the car battery, however, the dodgy 4xAA battery contraption was fortunately omitted from the race-ready car.

3.4. Connections

Most of the connections in the project revolve around the PCA9685 board. The board allows Jetson Nano to control the motors, while the power for the steering servo is coming from the car battery. This was necessary, since the Jetson was unable to power the servo on its own. The PCA board was controlled by Jetson through an I2C bus, allowing Jetson to send PWM signals to give the motors the values for steering and throttle. Jetson Nano offers two I2C buses, located on the pins 3 (SDA) and 5 (SCL), or 27 (SDA) and 28 (SCL), which were connected to the SDA and SCL pins on the PCA board. In addition, Jetson's 5V and GND pins were connected to VCC and GND pins on the PCA to power the board. (Note, that this isn't used to power the servo.)

The electronic speed controller (ESC) and the steering servo were connected to the PCA board's PWM channels 0 and 1, respectively, and since the ESC takes its power straight from the main battery, we removed the power cable between the ESC and the PCA. For the servo we used a buck converter (chapter 3.3.), which was connected to the PCA's V+ port.

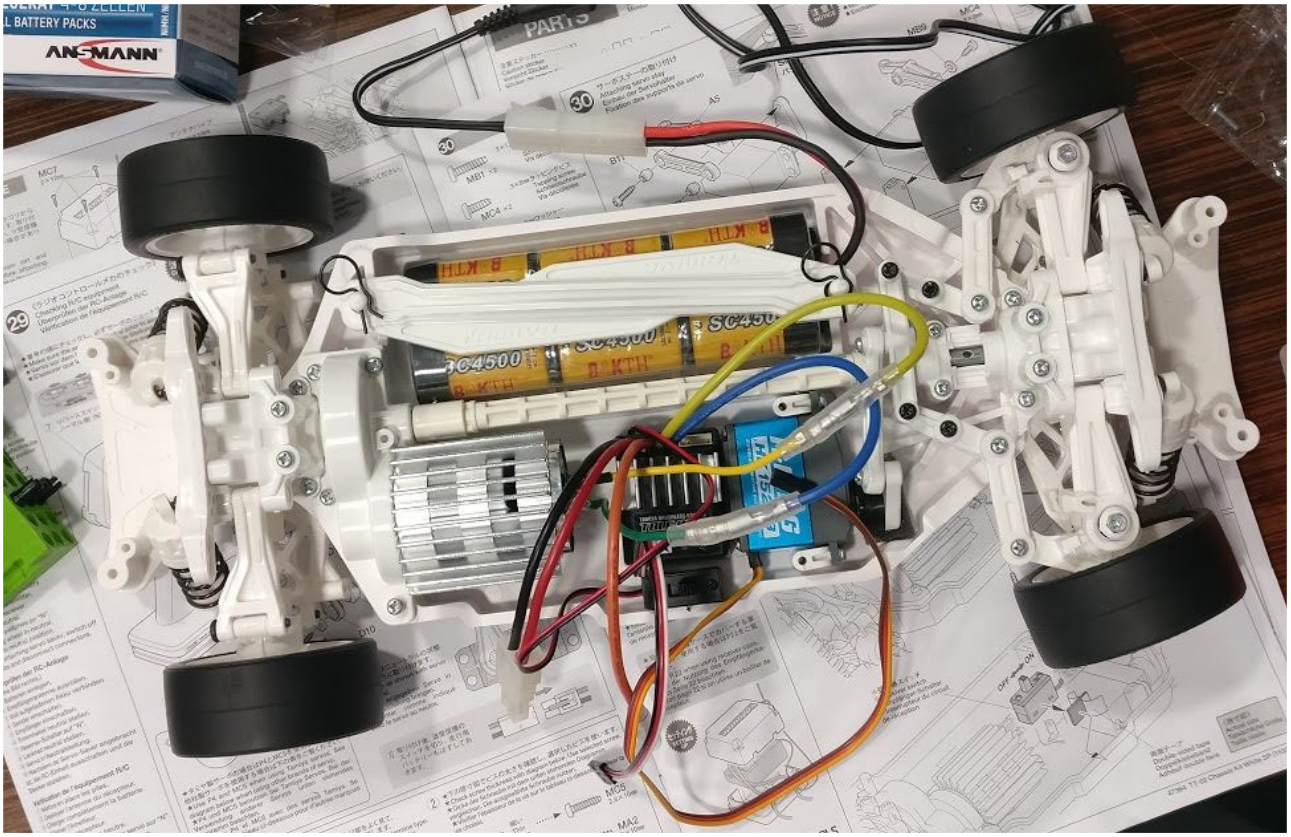

Figure 1. The finished car chassis

4. Nvidia Jetson Nano

4.1. Jetson Nano vs Raspberry Pi

For the driving computer we had the option to choose between a Raspberry Pi and Nvidia Jetson Nano. They are both single-board computers and are used in similar ways, but Jetson Nano offers more computing power and is designed for machine learning, so we decided to use it over Raspberry Pi. However, since Jetson Nano required an external power bank of its own, using Raspberry Pi could enable everything to be powered from the car battery, and therefore result in a lighter car.

4.2 Hardware

Jetson Nano is a single-board computer designed to run multiple neural networks in parallel for applications like image classification and object detection. It has a Quad-core CPU, a 128-core Maxwell GPU and 4GB of RAM. It can be powered through Micro-USB with 5V⎓2A or a DC barrel jack with 5V⎓4A and can be run on 10W or 5W modes.

Jetson Nano runs Linux and has USB and HDMI/DP, which means it can be used just like any other Linux-based computer, making it easy to use. In addition, it has ports for I2C, UART and GPIO, shown in Figure 2.

Unlike newer models of Raspberry Pi, Jetson Nano doesn't include on-board wireless connections, so a separate wifi antenna was used.

Jetson Nano J41 Header					
Sysfs GPIO	Name	Pin	Pin	Name	Sysfs GPIO
	3.3 VDC *Power*	1	2	**5.0 VDC** *Power*	
	I2C_2_SDA *I2C Bus 1*	3	4	**5.0 VDC** *Power*	
	I2C_2_SCL *I2C Bus 1*	5	6	**GND**	
gpio216	**AUDIO_MCLK**	7	8	**UART_2_TX** */dev/ttyTHS1*	
	GND	9	10	**UART_2_RX** */dev/ttyTHS1*	
gpio50	**UART_2_RTS**	11	12	**I2S_4_SCLK**	gpio79
gpio14	**SPI_2_SCK**	13	14	**GND**	
gpio194	**LCD_TE**	15	16	**SPI_2_CS1**	gpio232
	3.3 VDC *Power*	17	18	**SPI_2_CS0**	gpio15
gpio16	**SPI_1_MOSI**	19	20	**GND**	
gpio17	**SPI_1_MISO**	21	22	**SPI_2_MISO**	gpio13
gpio18	**SPI_1_SCK**	23	24	**SPI_1_CS0**	gpio19
	GND	25	26	**SPI_1_CS1**	gpio20
	I2C_1_SDA *I2C Bus 0*	27	28	**I2C_1_SCL** *I2C Bus 0*	
gpio149	**CAM_AF_EN**	29	30	**GND**	
gpio200	**GPIO_PZ0**	31	32	**LCD_BL_PWM**	gpio168
gpio38	**GPIO_PE6**	33	34	**GND**	
gpio76	**I2S_4_LRCK**	35	36	**UART_2_CTS**	gpio51
gpio12	**SPI_2_MOSI**	37	38	**I2S_4_SDIN**	gpio77
	GND	39	40	**I2S_4_SDOUT**	gpio78

Figure 2. Jetson Nano J41 Header Pinout

4.3 Software

All of the software developing was done on the Jetson by using a laptop to control it through ssh. The first thing to do was to install donkeycar, an open source project for autonomous RC cars. Donkeycar comes with all of the features needed for a self-driving vehicle, but installing it required us to first manually build OpenCV on the Jetson. Donkeycar's website has a tutorial for getting started. However, it doesn't have support for the camera that was used, but since Futurice had experience on the camera, we used their help to get the camera working properly.

For driving the car, there are multiple options to choose from. When the drive script is executed, it will set up a local web controller, which can be accessed through the car's ip address. This web controller shows the camera feed from the car and the values for throttle and angle, which can be adjusted by keyboard, on-screen joystick, device tilt, or a gamepad, which in our case meant using an Xbox One controller plug into the laptop we were using to monitor the car. It is also possible to connect a controller straight to the car, which would reduce latency, but that requires the controller to be compatible with the car. Most widespread bluetooth controllers should work. However, we didn't have a controller with bluetooth connection, and since the latency was already almost unnoticeable (~0.2s), we decided to go with what we had.

To train the car we had the choice between training on Jetson Nano or a separate computer. Both have their own advantages, namely, training on the Jetson doesn't require sending the data back and forth between the car and a computer, and training on a separate computer allows the use of Jetson to gather more data while the data acquired earlier can be trained at the same time. Out of the two, we chose to train on a separate machine by using rsync to transfer the data between the car and the computer.

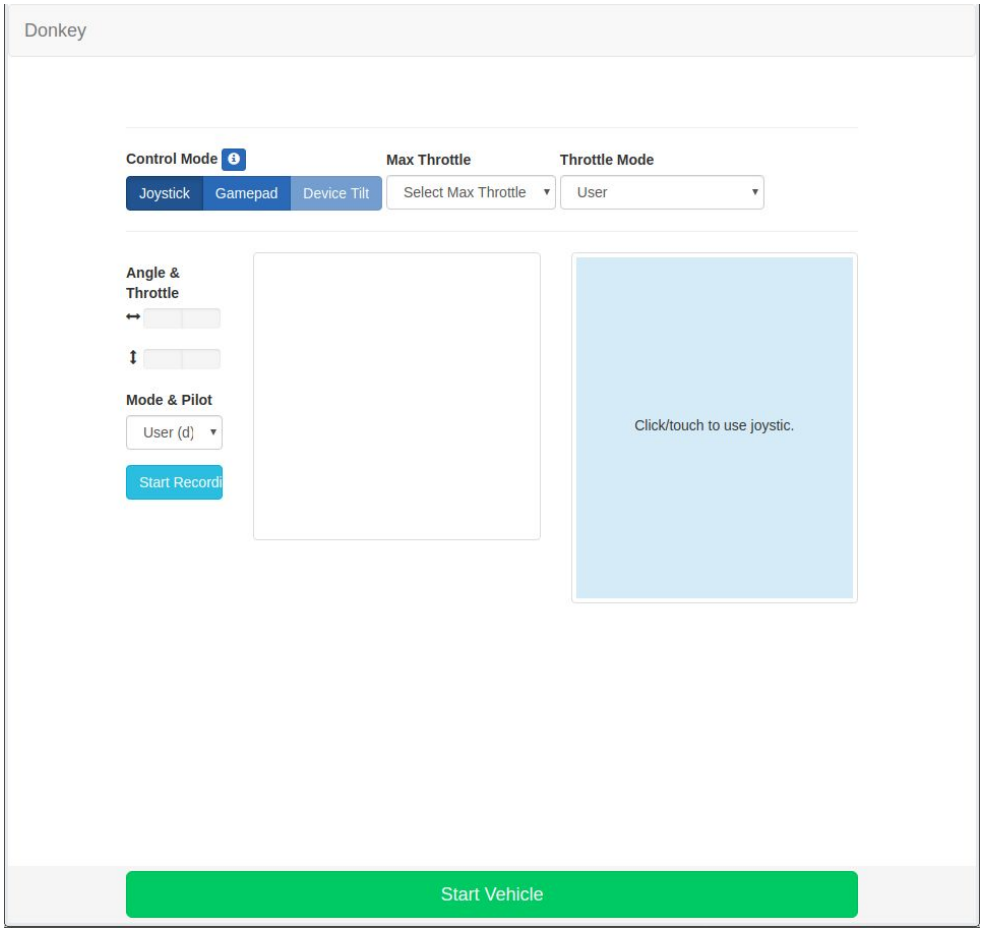

Figure 3. Layout of donkeycar's web controller

5. Ultrasonic sensors and obstacle detection

5.1. Practical issues

The chassis is made of plastic, and most of the parts used in the project are 3D printed. In prior testing, it is evident that the car running under high velocity may hit some obstacles, result in damaging the chassis. With technical difficulties, the player may not have enough time to stop the car, therefore installing some sensors can reduce the chance of hitting obstacles, providing a safe driving environment.

In addition, several traffic cones will be placed in the final contest. Hitting a traffic cone result in lower points in the race. Distribution of sensors can prevent the chassis from hitting traffic cones and thus assist the driving.

5.2. Hardware

After testing different kinds of sensors, ultrasonic sensors HC-SR04 together with Arduino Uno were selected as the solution of the anti-collision system. The HC-SR04 ultrasonic sensor uses sonar to determine the distance between the obstacle and the sensor, with an effective angle of 15 degrees. Three identical sensors are installed in the front, providing the vision for all three angles, left, right and middle. Sensors are separated at an angle of 30 degrees to ensure the detection area is wide enough. With the help of sensors mount, directions of sensors are consistent. The following diagram by Component101.com describes how ultrasonic sensor works.

Figure 4. The schematic of ultrasonic sensor

All sensors are connected to the Arduino Uno board, where the data of sensors are being processed and transferred to Jetson Nano through USB cable. The Jetson Nano then processes the input data and react upon it.

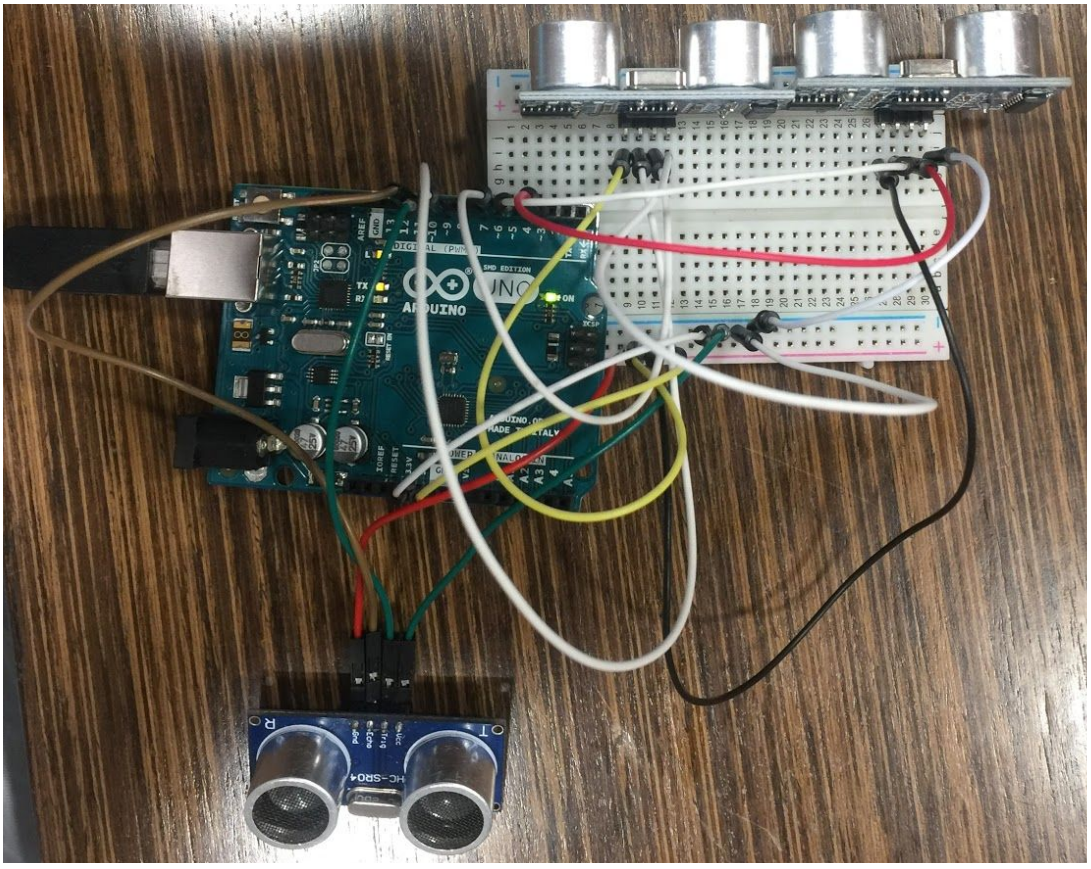

Figure 5. The prototype of sensors and connections

5.3. Software

The Arduino reads the data from ultrasonic sensors one at a time so that sensors do not interfere with each other. Then, the Arduino save the data as a string separated with blank spaces. The data is then transmitted to the Jetson Nano through the serial port. With the help of the Pyserial library, the Jetson Nano is able to interpret the input.

The data has three values from all sensors. This will be processed on Jetson Nano by comparing the values with the reference value to the ground and act correspondingly. If the collision is inevitable despite the turning direction of the chassis, it will stop the car as soon as possible.

To use the data, the controlling system needs to be modified. The system includes two controlling methods, manual control and self-intelligent control with machine learning. In general, the chassis has two ways of manual controlling, joystick and the web-based controller. The web-based controller provides a graphical user interface, and the control can be done by pressing keys. Since it is impractical to have the connection with joystick all the time, the modification will be performed on a built-in web-based controller to change the throttle and angle value. The original script can be found under the path /donketcar/parts/web_controller/web.py. The manual controlling system will be deployed once the T265 camera is in use.

However, there are certain limitations to this change. Because the car has not been trained, we cannot know how the manual controlling system interacts with the self-driving system. If the manual controlling system is going to overtake the self-driving intelligence, then this approach cannot be adapted.

The self-driving controlling system requires some deep understanding of Machine Visions and Deep Learning. This requires a deep understanding of the whole system as well as the library used

in the system, such as gym library. At this stage, we cannot find a way how to integrate the sensors into the self-driving system. With further learning of the system, the deployment will be clearer.

Despite the controlling system, it is worth noticing that the sensors may result in slower responses in the system due to the time delay of the sensors. This is very unlikely to happen but could be a potential risk in training.

5.4. Limitations and driving test

Ultrasonic sensors were tested on a computer first before the final deployment on Jetson Nano. The result was satisfactory, with all sensors worked. The only problem discovered in this stage was that the sensor was not able to detect any cylinder-shaped object. Otherwise, the ultrasonic sensors were able to detect the distance change rapidly, which provided enough data for obstacle detection.

In the follow-up test, there were many unforeseeable limitations which restricted the implementation. Ultrasonic sensors were deployed on the chassis, but the result was not reliable enough. The output of ultrasonic sensors changed rapidly with random values involved. One of the challenges was that the top ultrasonic sensor blocked the camera, hence the top ultrasonic sensor was removed from the chassis. In addition, the ultrasonic sensors installed on the Arduino used a serial port for connection, which has caused a tremendous amount of problems in the whole driving system, blocking the whole driving system from starting. The procedure for debugging and integration took too much time than expected, to accomplish the best overall performance for the whole project, the obstacle detection solution was compromised and removed in the final version of the self-driving car.

6. Intel RealSense T265 camera usage

6.1. Hardware introduction

Figure 6. Intel RealSense T265 camera.

Intel RealSense T265 camera is a compact, low power camera capable of simultaneous localization and mapping (SLAM). Our Futurice contacts were able to borrow us one, and we were excited to test its possibilities for our use case.

The camera has dual black and white fisheye lenses at the front, and it can measure all 6 degrees of freedom. Based on the camera and other sensor feed, it can make quite good guesses where it is positioned compared to the starting position. It does all of this with its built-in custom SoC. USB 3 also allows a live image feed processing.

6.2. Software introduction

Intel provides a development kit called 'Intel RealSense SDK' done with C and C++ for free. We were especially interested in the Python wrapper to the newest version called pyrealsense2.

The library provides a live feed for sensor values and camera frames. Some of the features are not well documented, but at least some working examples can be found from the source page.

6.3. Image pre-processing

It was known that the final demo would take place on a new track with much darker ground base color, not the one where we would do our testing. That is why we decided to do pre-processing to reduce the impact that the different ground color would have in our image. All of the pre-processing steps were done with python wrapper of the OpenCV library.

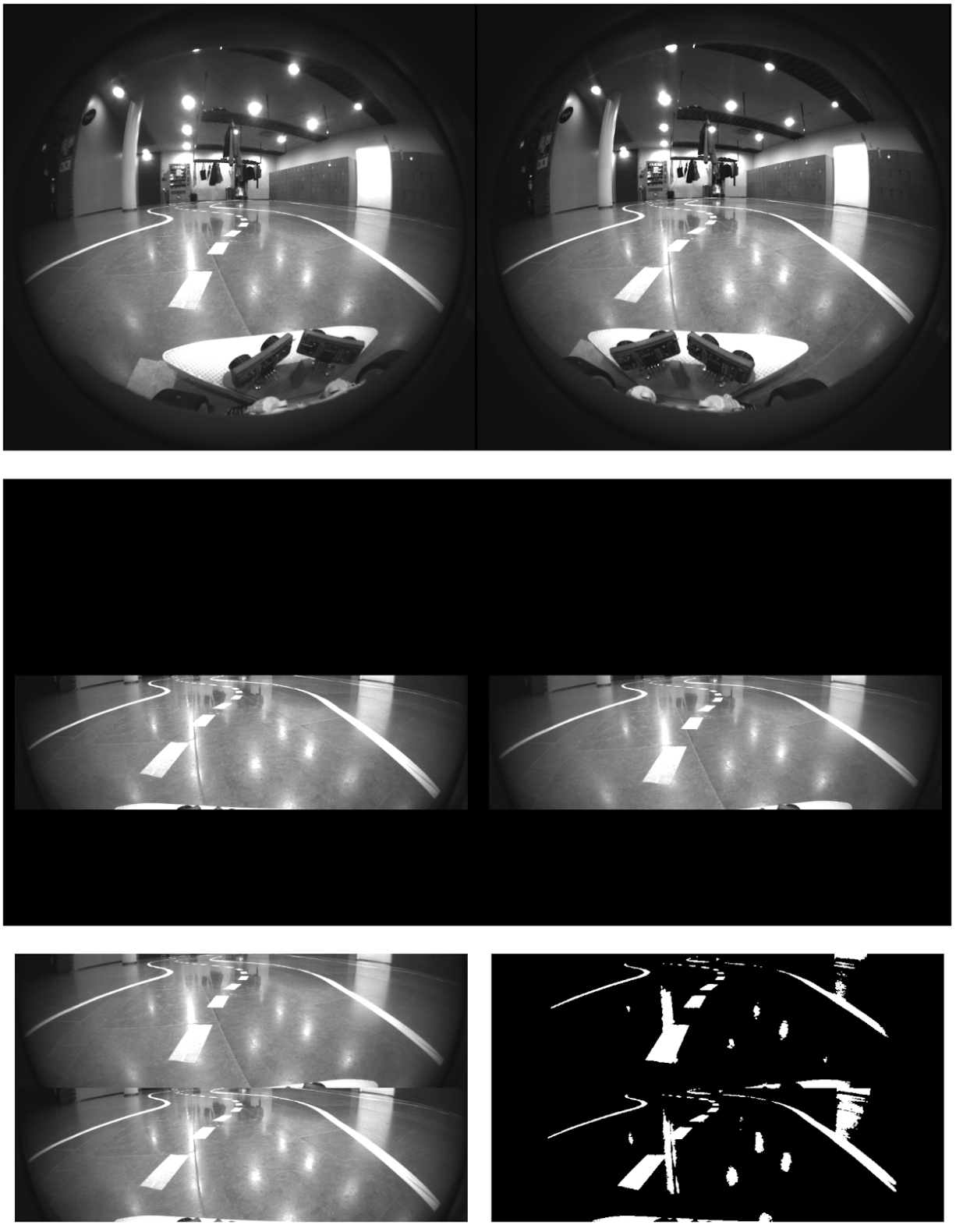

Figure 7. The frame pre-preprocessing steps.

Figure 7 above demonstrates the steps that we made to the image frames before feeding it to the machine learning model. Undistorting the fisheye effect was not done, because the results were good without doing so.

The top shows the unmodified images both fisheye cameras concatenated together. After that, only the parts containing the actual track were cropped, and combined together, as demonstrated in image at the bottom left.

The last step was to use thresholding to change all of the pixels either black or white. This was done because we wanted to generalize our results so it would work hopefully on slightly different tracks also.

6.4. Mapping results

Figure 8. A high-tech camera mapping test rig.

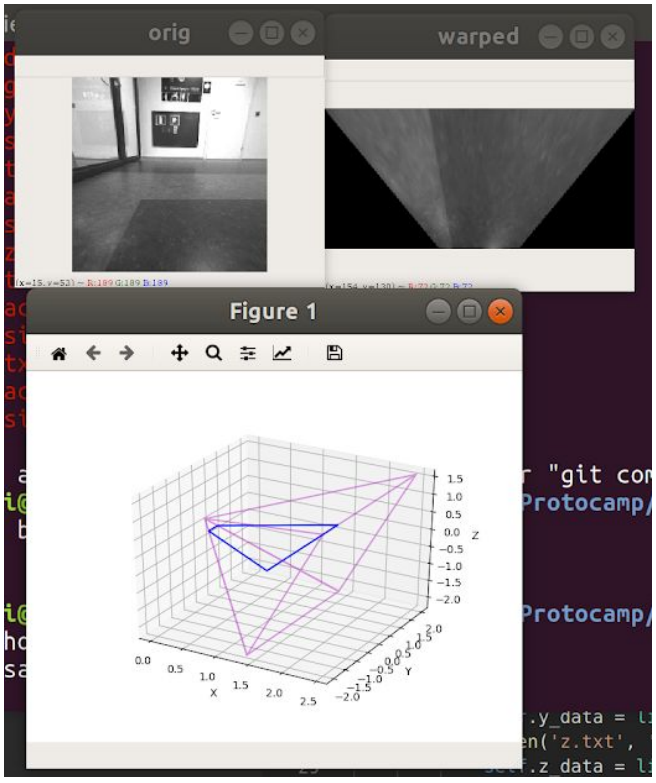

Figure 9. Image warping example.

As can be seen from the images above, we have gotten the library to work. The idea is to test the camera's feasibility in mapping (stitching) the images together from the image feed, so it can be fed to a more advanced AI model.

The image processing has been done with OpenCV and Numpy, which allow live processing with an acceptable frame rate even with Python.

But in the end we did not have a clear use for the mapping tool, so we decided to move our focus to other tasks.

7. Q-learning

7.1. Goals

The Donkeycar system effectively sets a limit to how good of a driver the AI can be; it basically cannot surpass the capabilities of the people training it. In order to overcome this issue, our goal is to implement an AI that would learn and become better eventually beating any human driver. The idea is to do this with reinforcement learning (Q learning). In short, when the AI does something right it gets a bigger number (or "reward") and when it does a bad thing it gets a smaller number. Based on this it learns which actions are desirable in each possible situation.

This will be achieved by generating the environment from the data collected with the camera from an initial test lap around the track. This alone won't quite cut it because Q learning takes the AI thousands of attempts to produce acceptable results. We will attempt to solve this by pre-training the AI with an artificially created virtual track. The AI's self-learning will be evaluated with the help of a MatLab program written for the purpose.

7.2. What was achieved

It turns out that the Gym library, which would have been used for training the AI, had issues. The library as a whole was a valuable tool in AI development and inspired confidence that the goal is reachable, but the one environment in the library that was needed for the car was not functional in the current iteration of the library. This coupled with problems beyond our power made it impossible to fix the issues within a reasonable time frame, and the development of the self-learning algorithm was dropped in favour of more feasible challenges.

As a final verdict: The self-learning AI wasn't achieved. However, without the aforementioned time constraints and external difficulties, the AI would not only have been possible but would also have made the car significantly faster and more agile. The Gym environment in question would have likely been a close-enough match for real-life conditions requiring only little alterations.

8. 3D modeling and printing

8.1 Goals

The original Tamiya TT-02 bumper was too narrow to offer any protection to the front wheels and according to Futurice an impact to the wheels could potentially damage the steering technology. Therefore we needed to develop a broader bumper with better shock absorption qualities. We also needed a mount for three ultrasonic sensors and stand for Intel RealSense T265 camera. Both of them needed an adjustable vertical angle. As there were no readily available options that would fit our needs we had to do the 3D modeling for custom parts ourselves.

8.2 Results

Considering the amount of time we had for this project, our results in 3D printing are good. As an outcome we have three custom 3D models and some customized PELA blocks.

Part	Design platform	Printing program	Material	Printer
Bumper	Autodesk Fusion 360/ Solid Works	Ultimaker Cura 4.1.0	white TPU 95A	Ultimaker S5
Sensor bottom	Onshape	Cura Lulzbot Edition	red ABS	Lulzbot TAZ 5
Sensor top	Onshape	Ultimaker Cura 4.1.0	red PLA	Ultimaker 2+
PELA technical beams 1x12	OpenSCAD	Cura Lulzbot Edition	red ABS	Lulzbot TAZ 5
PELA Arduino Uno Technic Mount	OpenSCAD	Cura Lulzbot Edition	white ABS	Lulzbot TAZ 5

Table 1. Technical information of 3D printed parts

8.2.1 Bumper

First the car bumper attachment parts were measured with a caliper to make the design fit perfectly on the Tamiya TT-02. The 3D model was designed with Autodesk Fusion 360 and then further developed in SolidWorks. After several experiments and prototypes, we chose thermoplastic polyurethane (TPU) 95A as bumper material.

TPU 95A is semi-flexible and rubber like material (Ultimaker Technical Data Sheet TPU 95A). It is more elastic than PLA flex, which allows to make thicker walls and thus giving a more stable form to the bumper. Print settings are important for optimal results: we used grid infill pattern with 35 % infill density and infill wall count was set to two. The bottom and top layers were set to zero, making the design hollow and thus giving it more flexibility and better shock absorption qualities.

TPU 95A can be a bit difficult material to print because it tends to detach from the glass plate. Ultimaker recommends using glue on the plate to secure the print attachment (How to print with Ultimaker TPU 95A). We used paper glue in addition to brim plate adhesion.

Figure 10. Bumper

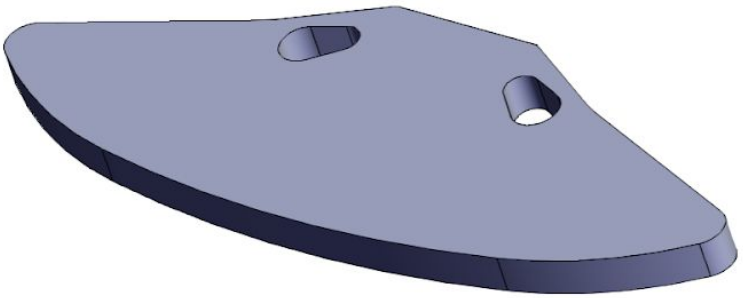

Figure 11. 3D model for bumper

Print settings for bumper	
Infill Pattern	grid
Infill Density	35 %
Infill Wall Count	2
Top/Bottom Layers	0
Wall Line Count	3
Plate Adhesion	brim

Table 2. Print settings for bumper

8.2.2 Sensor mounting

The HC-SR04 ultrasonic sensors were measured with a caliper and fitting 3D models were designed with Onshape. The setup consists of four parts that are attached to each other with screws. The screw attachment enables angle adjustment in vertical direction, which might be important to reach optimal ultrasonic sensor readings. After adjusting the angle, the joints may be glued to avoid any movement during racing. The bottom part fits on top of the RC-cars front bumper and replaces the original part that attaches the bumper.

We chose red Polylactic acid (PLA) and red Acrylonitrile Butadiene Styrene (ABS) as materials. While ABS is less environment friendly, it is stronger and tougher material than PLA (MATBASE, designsite). The bottom part, which is referred to as sensor bottom, was printed with red ABS to reach a more durable result. The ultrasonic sensors are attached to the three top parts. Top parts were printed with PLA as they don't need to withstand similar stress as sensor bottom.

The onshape objects and assembly view is open for all users: Assembly, top part, bottom part

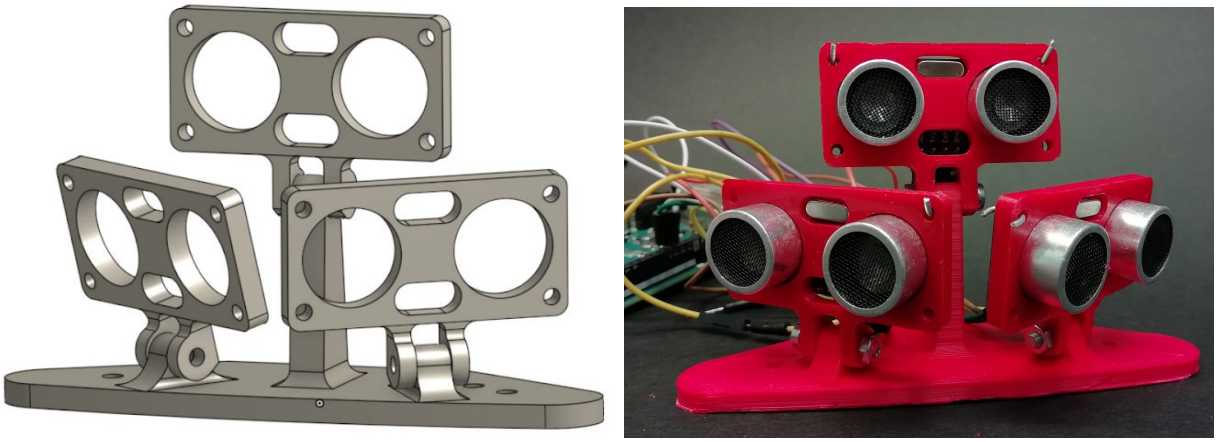

Figure 12-13. Assembly of the sensor mount

8.2.3 Camera stand

The Intel RealSense T265 camera is attached to the camera stand with two screws. The design was made with Onshape and consists of three parts: camera holder, leg and an adapter between these two. Further development could be useful to make the design simpler so it includes only two parts. This could be achieved by merging the adapter part to the camera holder. Nevertheless, there has to be at least two separate parts because of the adjustable vertical angle.

The camera stand was printed with red ABS. We faced challenges in making the camera leg durable enough to withstand collision impacts. The seams of printing layers were the most likely place to fail in crashes. Printing the leg lying down achieved more resistant results as the layers in the print were vertical instead of horizontal.

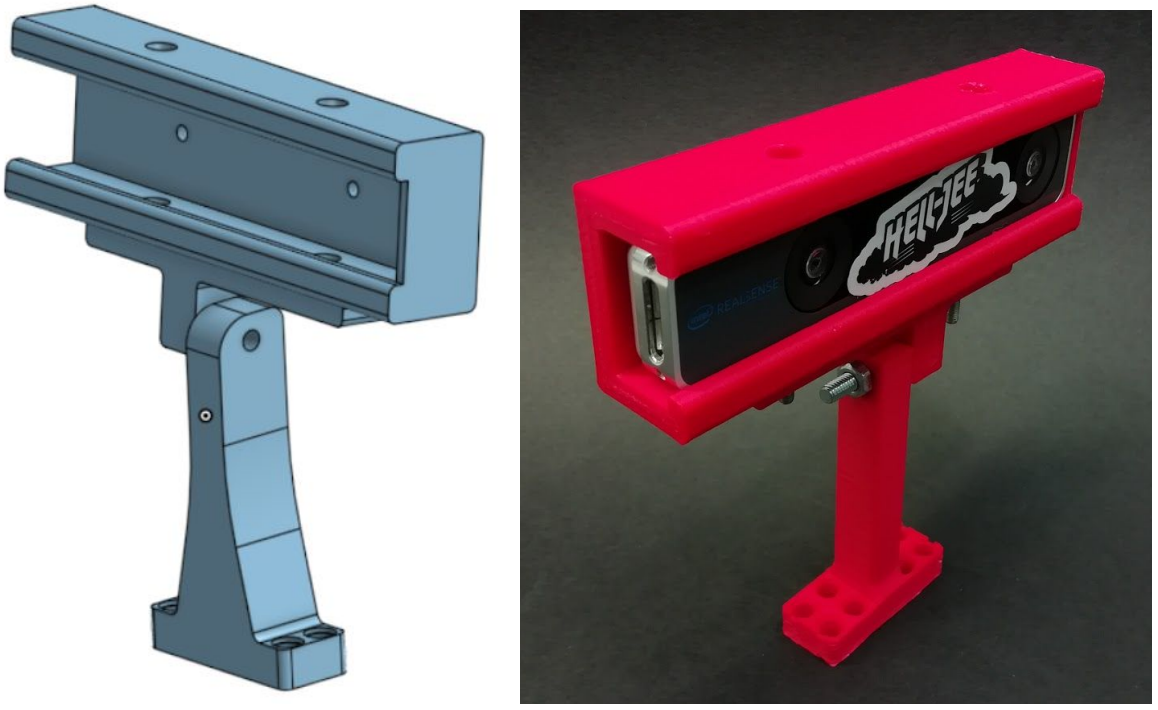

Figure 13-14. Camera stand for Intel RealSense T265

8.2.4 PELA blocks

The open source code PELA blocks were calibrated to Ultimaker Extended 2+ 3D printer. The same calibration worked also with TAZ 5 printer and the Ultimaker S5.

We modified the PELA Technic Beam to be 12 holes long and 1 hole wide so that it could be used to attach Jetson Nano to the car PELA Drift Car Center Beam. The PLA version couldn't survive crash testing so the following beams were printed with ABS.

We also printed a PELA Arduino Uno Technic Mount for the Arduino Uno which was used to control the ultrasonic sensors. We used white ABS for the technic mount and printed it with TAZ 5 printer.

9. Reflection of the Project

9.1. Reaching objective

In the end, we were able to finish all of the work marked as necessary in the original plan. We also had many things listed as extra features, but after researching into those, many features were left out of the project.

9.2. Timetable

Getting all of the individually working parts together became much bigger challenge than we expected. That is why a large portion of the project was done on the last week, even though we tried to avoid that happening.

Individually, many parts of the project had about the workload that we anticipated. For example ground mapping and q-learning were known to be very challenging, and that is why we needed to leave those out of the final result.

9.3. Risk analysis

We had two major risks: time running out and random technical problems we could not see beforehand. These challenges were known and informed via the team chat.

We managed both of those problems by doing drastically more work towards the end of the project. It probably was not the most optimal way, but some of the technical difficulties were much more challenging to figure out than we could have imagined, and we wanted to finish the project on time.

10. Discussion and Conclusions

Overall the project was a success, we were able to finish a fully working and self-driving car and won the demo race.

All of the necessary parts were done, but many of the things we wanted to include had to be left off because of technical difficulties and less time than we anticipated. Q-learning, photo-mapping, ultrasonic sensor integration were all harder problems than we expected in our original plan.

But in the end we were able to overcome all of the major problems related to the design, Donkey Car customization/bug fixing and electrical work.

References

Polylactic Acid (PLA). Matbase.
http://www.matbase.com/material/polymers/agrobased/polylactic-acid-pla/properties

Torben, L. ABS - acrylonitrile butadiene styrene. Designinsite.
http://designinsite.dk/htmsider/m0007.htm

Technical data sheet TPU 95A. Version 3.010. Ultimaker.
http://3dneworld.com/wp-content/uploads/2017/08/TPU95A.pdf

How to print with Ultimaker TPU 95A. Ultimaker.
https://ultimaker.com/en/resources/22235-how-to-print-with-ultimaker-tpu-95a

"HC-SR04 Ultrasonic Sensor." *HC-SR04 Ultrasonic Sensor: Working, Pin Diagram, Description & Datasheet*, 18 Sept. 2017, https://components101.com/ultrasonic-sensor-working-pinout-datasheet.