

PROJEKT

ROBOTY MOBILNE

---

## Dokumentacja

# Track Observing Fully Independent Car TOFIC

---

*Skład grupy:*

Patryk SZELEWSKI, 241490

Daniel ŚLIWOWSKI, 241166

Jakub TOMASZEWSKI, 241576

*Termin:* ptTP17

*Prowadzący:*

dr inż. Wojciech DOMSKI

26 maja 2020

# Spis treści

<b>1</b>	<b>Opis projektu</b>	<b>2</b>
<b>2</b>	<b>Założenia projektowe</b>	<b>2</b>
<b>3</b>	<b>Urządzenia zewnętrzne</b>	<b>2</b>
3.1	Raspberry Pi 3 model B+ . . . . .	2
3.2	Google Edge TPU ARM Accelerator . . . . .	2
3.2.1	Konfiguracja urządzenia . . . . .	2
3.3	Serwo Feetch FS5103B . . . . .	3
3.4	Kamera Kamera Genius WideCam F100 . . . . .	3
<b>4</b>	<b>Projekt elektroniki</b>	<b>3</b>
4.1	Spis użytych elementów . . . . .	3
4.2	Schemat połączeń elektrycznych . . . . .	4
4.3	Wykonane połączenia elektroniczne . . . . .	6
<b>5</b>	<b>Konstrukcja mechaniczna</b>	<b>7</b>
<b>6</b>	<b>Makieta drogi i znaki</b>	<b>8</b>
<b>7</b>	<b>Konfiguracja RaspberryPi</b>	<b>11</b>
7.1	OpenCV . . . . .	11
7.2	TensorFlow i TensorFlow Lite . . . . .	11
7.3	Sterownik silników i serwa . . . . .	12
7.4	RealVNC Viewer i Server . . . . .	12
7.5	Sterowanie ręczne . . . . .	12
<b>8</b>	<b>Algorytm wykrywania pasa ruchu</b>	<b>12</b>
8.1	Wykrywanie krawędzi linii . . . . .	13
8.2	Przycinanie obrazu . . . . .	13
8.3	Wykrywanie linii . . . . .	14
8.4	Uśrednianie linii . . . . .	14
<b>9</b>	<b>Algorytm sterowania</b>	<b>14</b>
9.1	Obie linie są widoczne . . . . .	14
9.2	Jedna linia widoczna . . . . .	15
9.3	Brak linii . . . . .	15
<b>10</b>	<b>Zbieranie danych treningowych</b>	<b>15</b>
<b>11</b>	<b>Detekcja obiektów na drodze</b>	<b>15</b>
11.1	Transfer learning . . . . .	15
11.2	Kwantyzacja modelu . . . . .	16
<b>12</b>	<b>System kontroli wersji</b>	<b>16</b>
<b>13</b>	<b>Podsumowanie</b>	<b>16</b>
	<b>Bibliografia</b>	<b>16</b>

# 1 Opis projektu

Celem projektu było zaprojektowanie oraz wykonanie robota mobilnego zdolnego do autonomicznego przemieszczania się po makiecie ulicy. Robot reagować miał na pasy ruchu na drodze oraz wybrane znaki drogowe (znak STOP oraz sygnalizację świetlną). Całość zrealizowana została w oparciu o mini-komputer Raspberry Pi 3B+.

## 2 Założenia projektowe

Głównym założeniem projektu było skonstruowanie autonomicznie przemieszczającego się po makiecie ulicy samochodu. Zakładało się implementację wykrywania pasa ruchu którym porusza się pojazd oraz algorytm sterowania mający za zadanie pozostanie na nim. Ponadto przewidywało się zaprojektowanie oraz wdrożenie algorytmów identyfikacji oznaczeń drogowych takich jak znak STOP oraz sygnalizacja świetlna

## 3 Urządzenia zewnętrzne

### 3.1 Raspberry Pi 3 model B+

Mikrokomputer Raspberry Pi 3 model B+ charakteryzuje się następującymi parametrami:

- Rdzeń procesora Quad-Core ARM Cortex-A53
- Taktowanie  $1,4GHz$
- Architektura  $ARMv8 - A$
- Pamięć RAM 1 GB LPDDR2 @ 900 MHz
- System operacyjny Raspbian 4.19

Dzięki mocy mikrokomputera możliwe jest wykonywanie wielu obliczeń w czasie rzeczywistym.

### 3.2 Google Edge TPU ARM Accelerator

Edge TPU to mały układ ASIC zaprojektowany przez Google, który zapewnia wysoką wydajność wnioskowania uczenia maszynowego przy niskim poborze energii. Na przykład, może wykonywać najnowocześniejsze mobilne modele wizyjne, takie jak MobileNet v2 przy 100+ fps, w energooszczędnym sposób. Posiada również dostępne do pobrania oprogramowanie TensorFlow lite.

#### 3.2.1 Konfiguracja urządzenia

**Instalacja środowiska uruchomieniowego (stan na 26.05.2020) [2]**

Pierwszym krokiem jest dodanie zdalnego repozytorium do systemu:

```
1 echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" |  
    sudo tee /etc/apt/sources.list.d/coral-edgetpu.list  
2 curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -  
3 sudo apt-get update
```

Następnie możemy dokonać instalacji środowiska:

```
1 sudo apt-get install libedgetpu1-std
```

W tym momencie należy podłączyć akcelerator do dowolnego portu USB Raspberry Pi za pomocą dołączonego do Google Edge TPU kabla USB typu C.

#### Instalacja biblioteki TensorFlow Lite

W celu uruchomienia pierwszej inferencji za pomocą skonfigurowanego urządzenia należy zainstalować bibliotekę TensorFlow Lite przeznaczoną do pracy na urządzeniach mobilnych:

```
1 pip3 install https://dl.google.com/coral/python/tflite_runtime-2.1.0.post1-cp37-  
    cp37m-linux_armv7l.whl
```

## Weryfikacja działania urządzenia

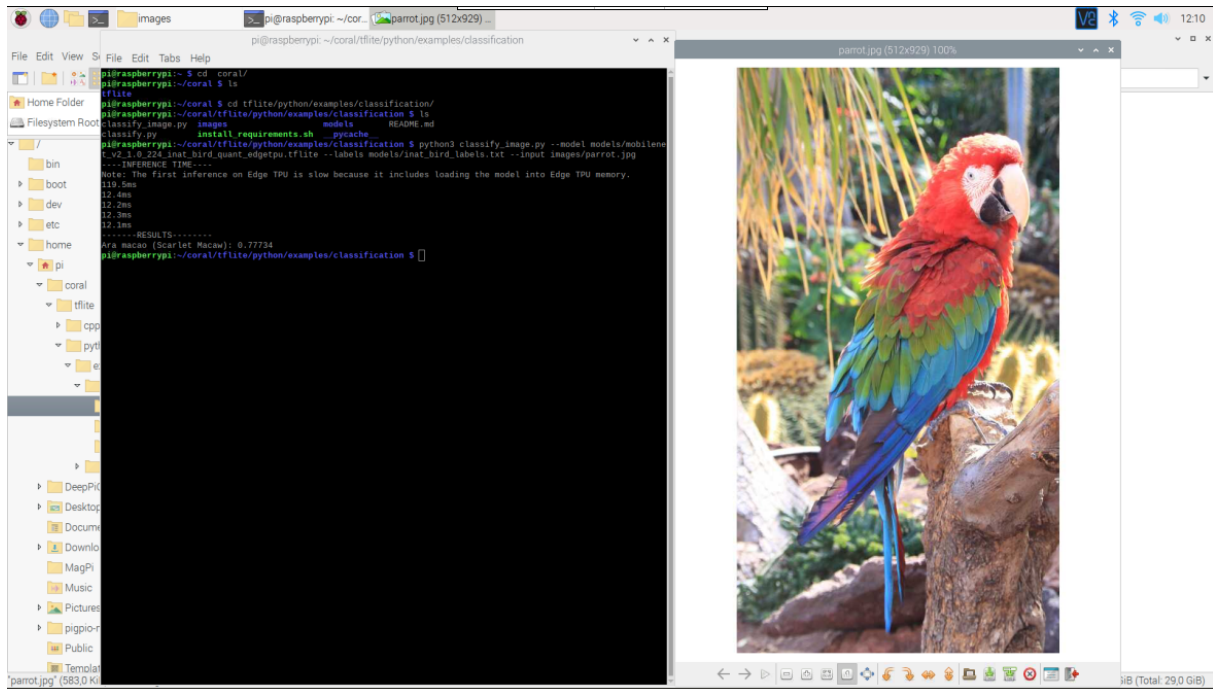
W celu weryfikacji działania Google Edge TPU należy zacząć od sklonowania zdalnego repozytorium z serwisu GitHub:

```
1 mkdir coral && cd coral
2 git clone https://github.com/google-coral/tflite.git
```

Następnie należy zainstalować model klasyfikatora ptaków, pliki nagłówkowe oraz zdjęcie testowe:

```
1 cd tflite/python/examples/classification
2 bash install_requirements.sh
```

Możemy uruchomić klasyfikator na pobranym w poprzednim kroku zdjęciu ptaka.



Rysunek 1: Działający klasyfikator uruchomiony na robocie TOFIC

### 3.3 Serwo Feetch FS5103B

Serwomechanizm sterowany jest za pomocą sygnały PWM. Zakres jego ruchów wynosi 180 stopni. Zasilany jest napięciem stałym 4,8 – 6V

### 3.4 Kamera Kamera Genius WideCam F100

WideCam F100 posiada obiektyw ultra szerokokątny z możliwością obrotu do 120 stopni z możliwością nagrywania w rozdzielczości do 1080p.

## 4 Projekt elektroniki

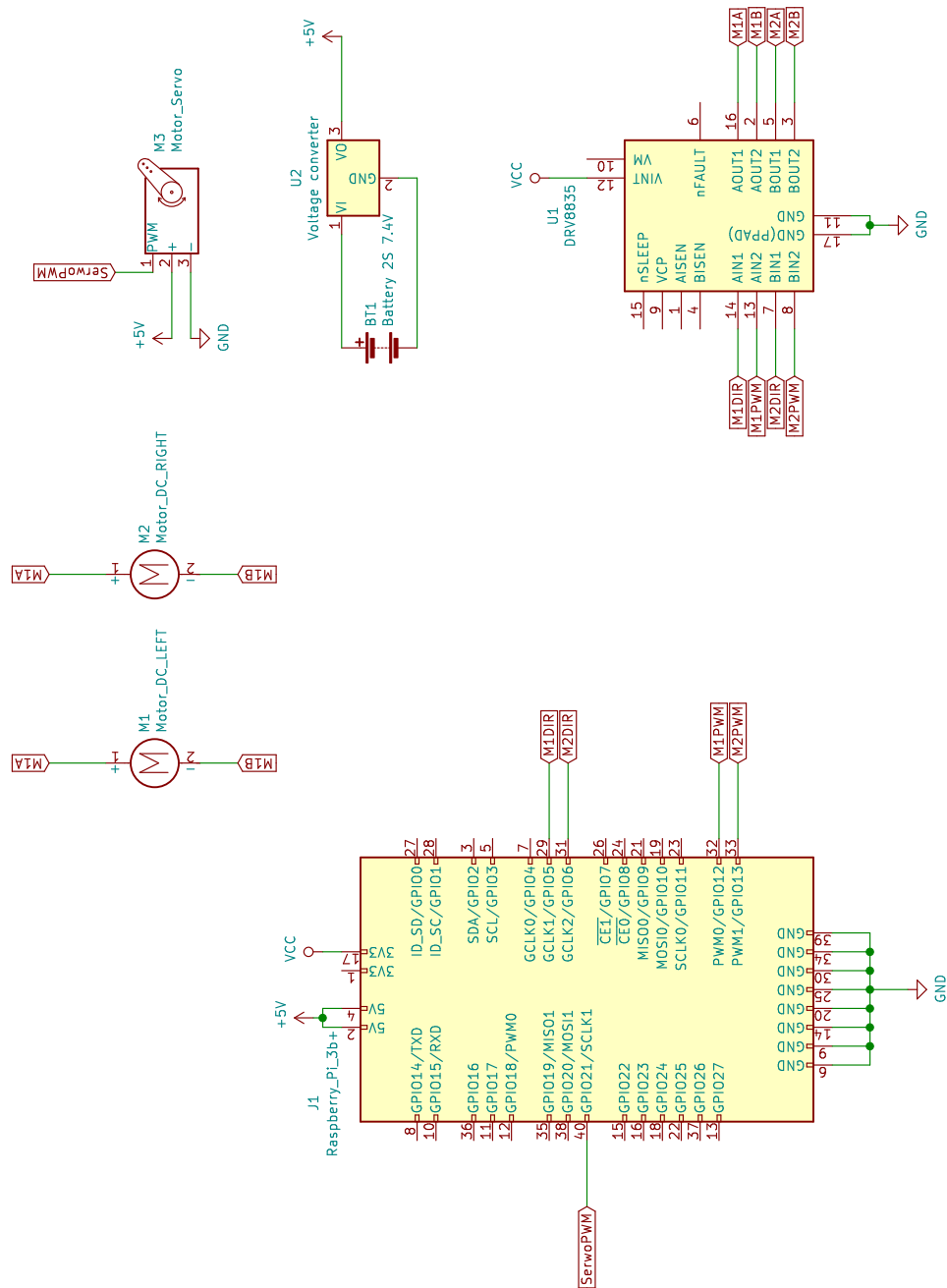
Udało się zrealizować wszystkie zadania związane z częścią elektroniczną robota. Poniżej przedstawiono wyniki wykonanych prac:

### 4.1 Spis użytych elementów

- Raspberry Pi 3 model B+ [5]
- Silnik DC Dagu DG02S-L 2szt
- Wskaźnik napięcia Li-pol 1-8S z buzzerem P309

- Taśma Rasperry Pi - kamera 50cm
- Koła Dagu RW002 65x26mm do silników DG - 4szt.
- Kamera Kamera Genius WideCam F100
- Serwo Feetch FS5103B
- Pakiet Li-Pol Redox 2200mAh 30C 2S 7.4V
- Pololu DRV8835 - dwukanałowy sterownik silników 11V/1,2A
- Przetwornica napięcia - ładowarka 2x USB 5V
- Google Edge TPU ARM Accelerator

## **4.2 Schemat połączeń elektrycznych**



**Patryk Szelewski**

Sheet: /  
File: Projekt\_elektroniki.sch

**Title: Robot mobilny TOFIC**

Size: A4	Date: 06.05.2020
----------	------------------

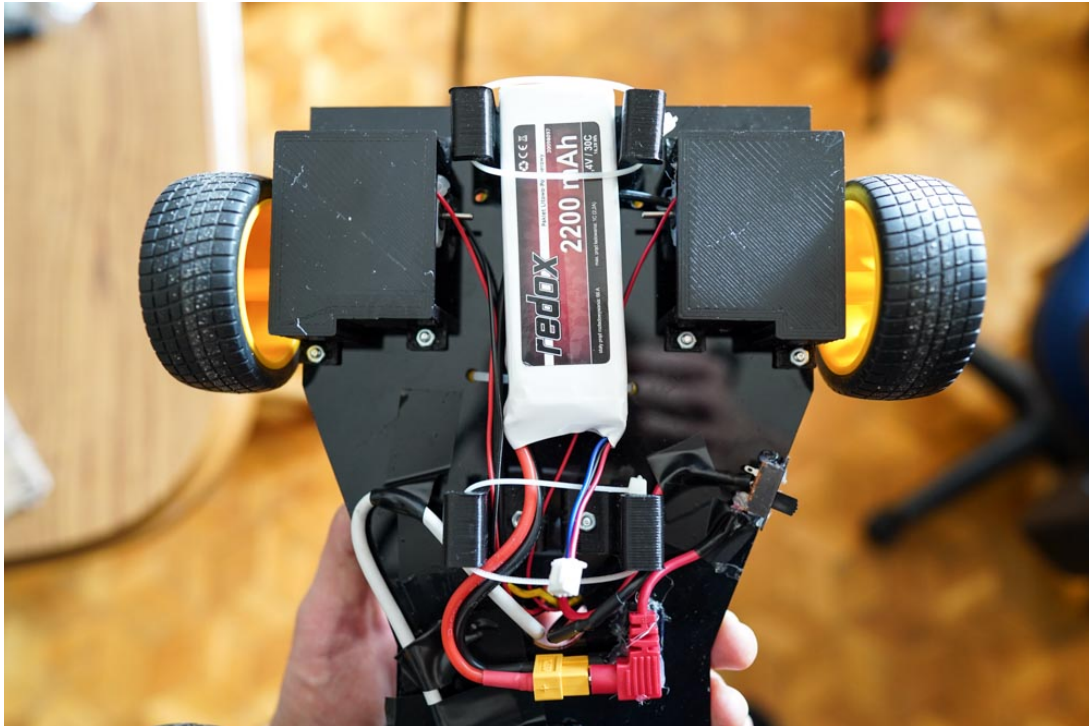
Id: 1/1

Id: 1/1

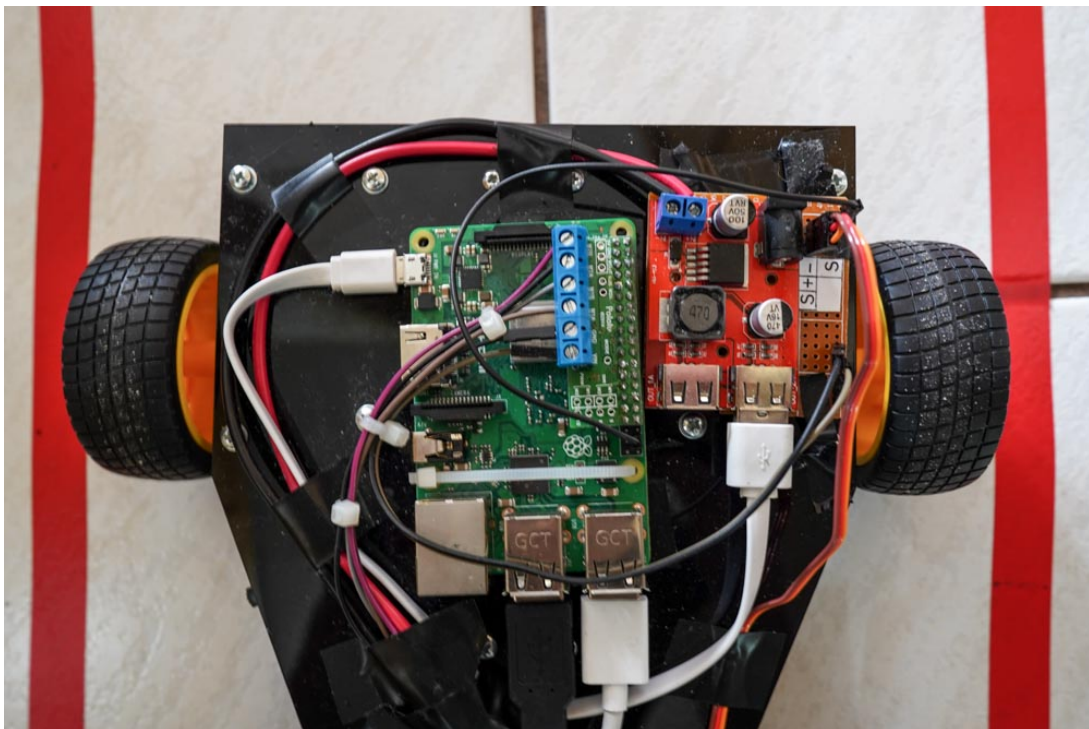
Rysunek 2: Schemat elektryczny robota mobilnego

### 4.3 Wykonane połączenia elektroniczne

Rezultaty wykonanych połączeń elektronicznych zamieszczono na zdjęciach 3, 4.



Rysunek 3: Gotowe połączenia elektroniczne w robocie mobilnym

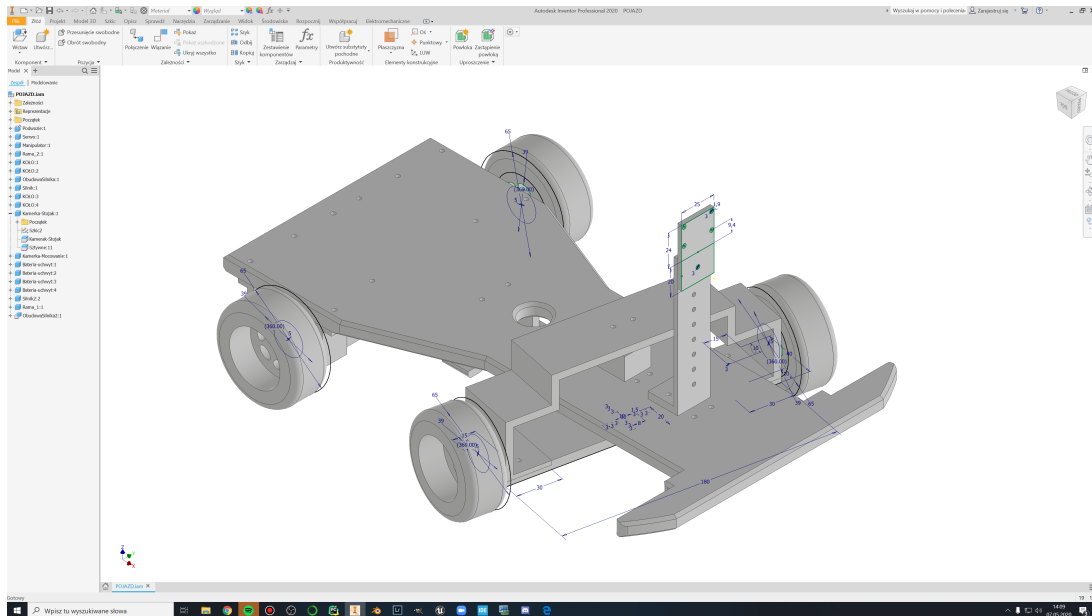


Rysunek 4: Gotowe połączenia elektroniczne w robocie mobilnym

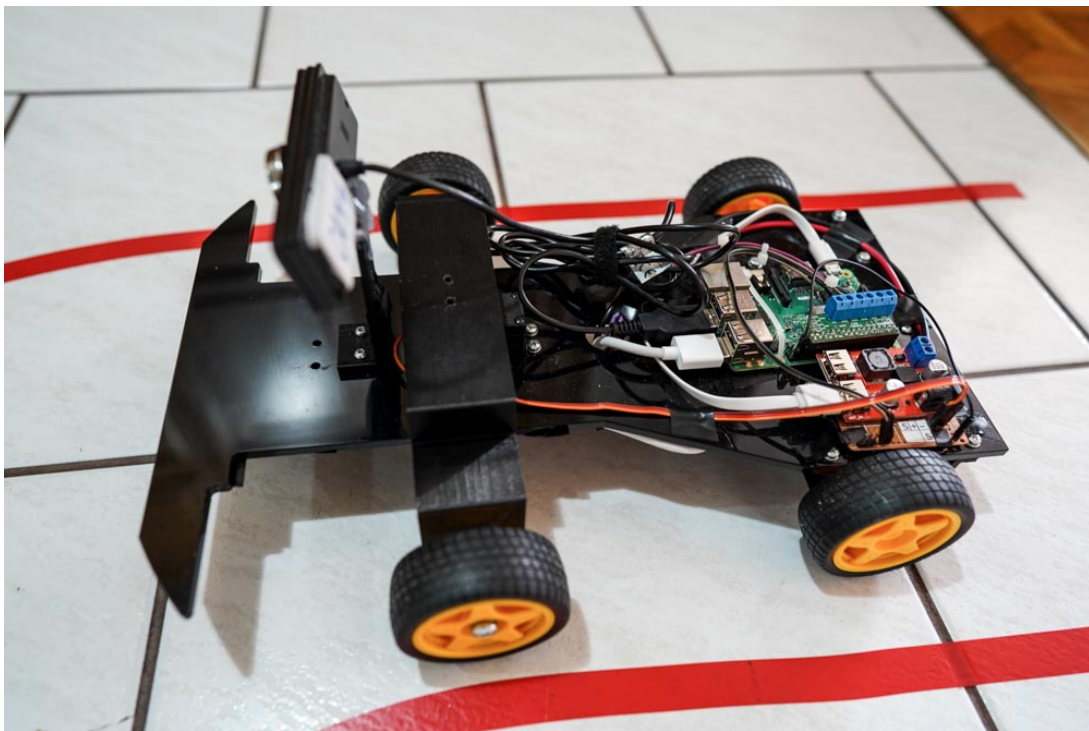


## 5 Konstrukcja mechaniczna

Konstrukcja Pojazdu składa się z 10 wydrukowanych na drukarce 3D elementów z materiału PLA, oraz jednego elementu (podwozia pojazdu) wyciętego laserowo w plexi. Całość konstrukcji mechanicznej zaprojektowana została w programie Autodesk Inventor [4]. Udało się zrealizować wszystkie przewidziane zadania dla kamienia milowego Mechanika, co widać na Rys. 5 oraz Rys. 6. Konstrukcję mechaniczną pojazdu uznaje się za zakończoną.



Rysunek 5: Model 3D pojazdu w programie Autodesk Inventor



Rysunek 6: Gotowa konstrukcja mechaniczna pojazdu

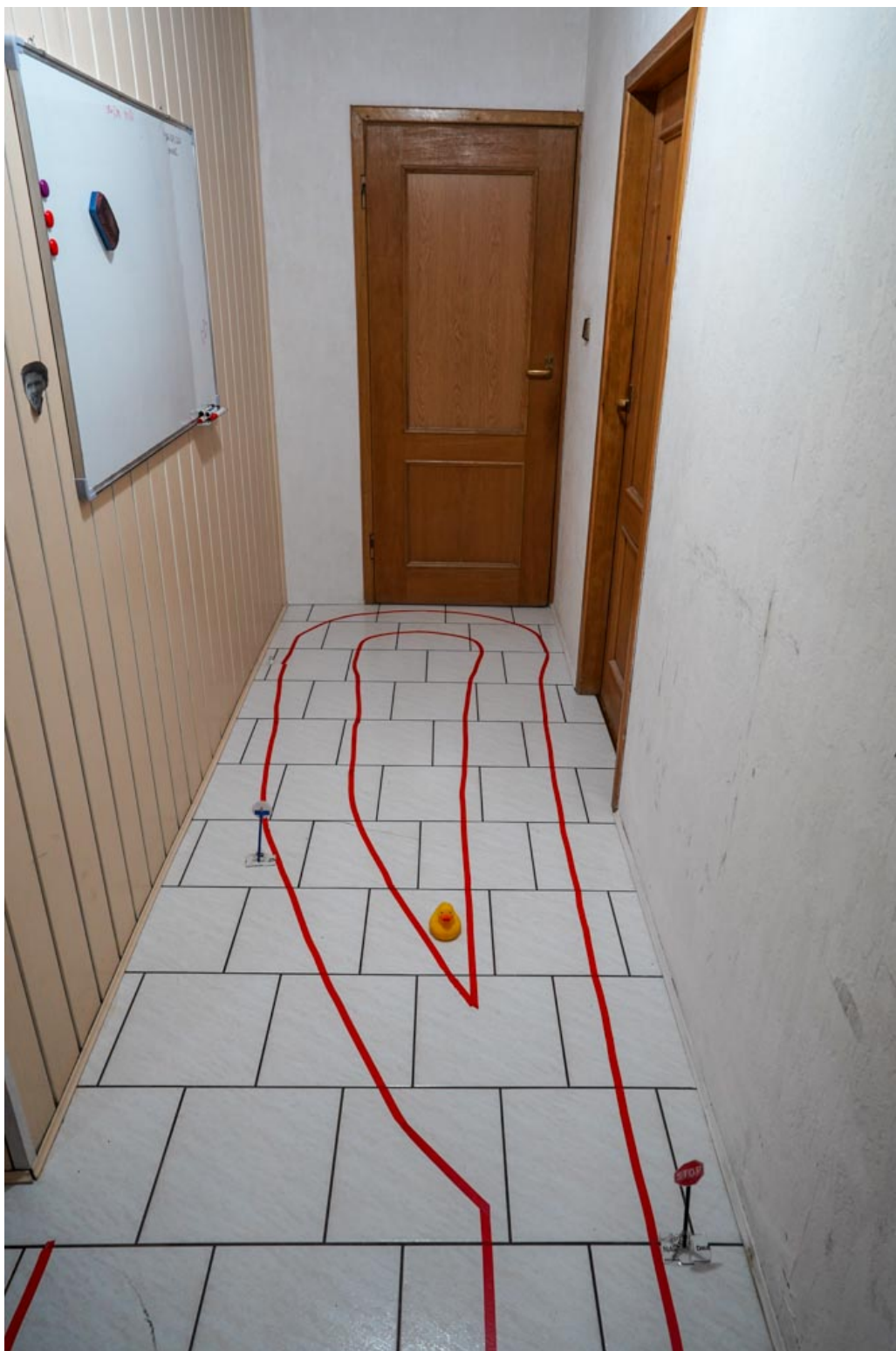


## 6 Makieta drogi i znaki

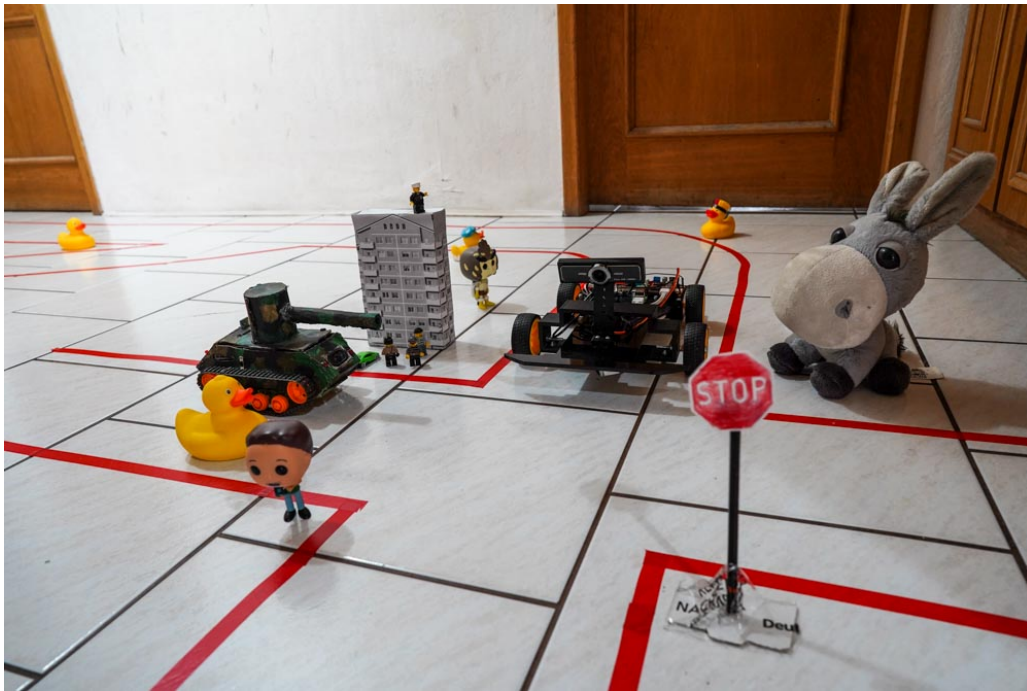
W ramach tego etapu należało zbudować makietę drogi oraz znak stopu i sygnalizację świetlną. Udało się zrealizować te zadanie czego dowodem są zdjęcia na rysunkach: 7, 8, 9, 10. Sygnalizacja świetlna została wykonana za pomocą płytki Arduino.



Rysunek 7: Makieta drogi



Rysunek 8: Makieta drogi



Rysunek 9: Skrzyżowanie



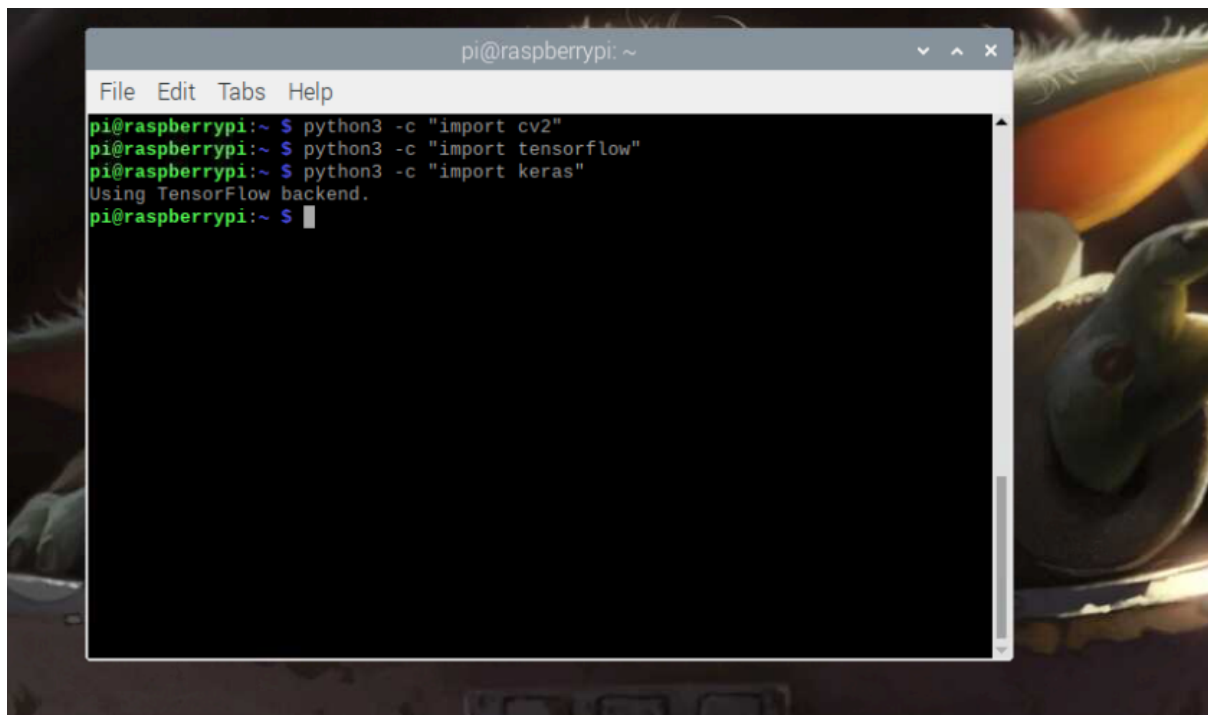
Rysunek 10: Znak Stop

## 7 Konfiguracja RaspberryPi

W ramach tego zadania należało zainstalować wszelkie biblioteki służące do obsługi kamery i sterownika silników. Ponadto napisano skrypty pozwalające na sterowanie serwomechanizmem oraz ręczne sterowanie robotem.

### 7.1 OpenCV

W celu przetwarzania obrazu z kamery zainstalowano bibliotekę **OpenCV**, która jest zbiorem funkcji pozwalających na odczytywanie obrazu z kamery i jego przetwarzanie. Poprawne zainstalowanie biblioteki obrazuje Rys. 11

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the following commands and output:

```
pi@raspberrypi:~ $ python3 -c "import cv2"
pi@raspberrypi:~ $ python3 -c "import tensorflow"
pi@raspberrypi:~ $ python3 -c "import keras"
Using TensorFlow backend.
pi@raspberrypi:~ $
```

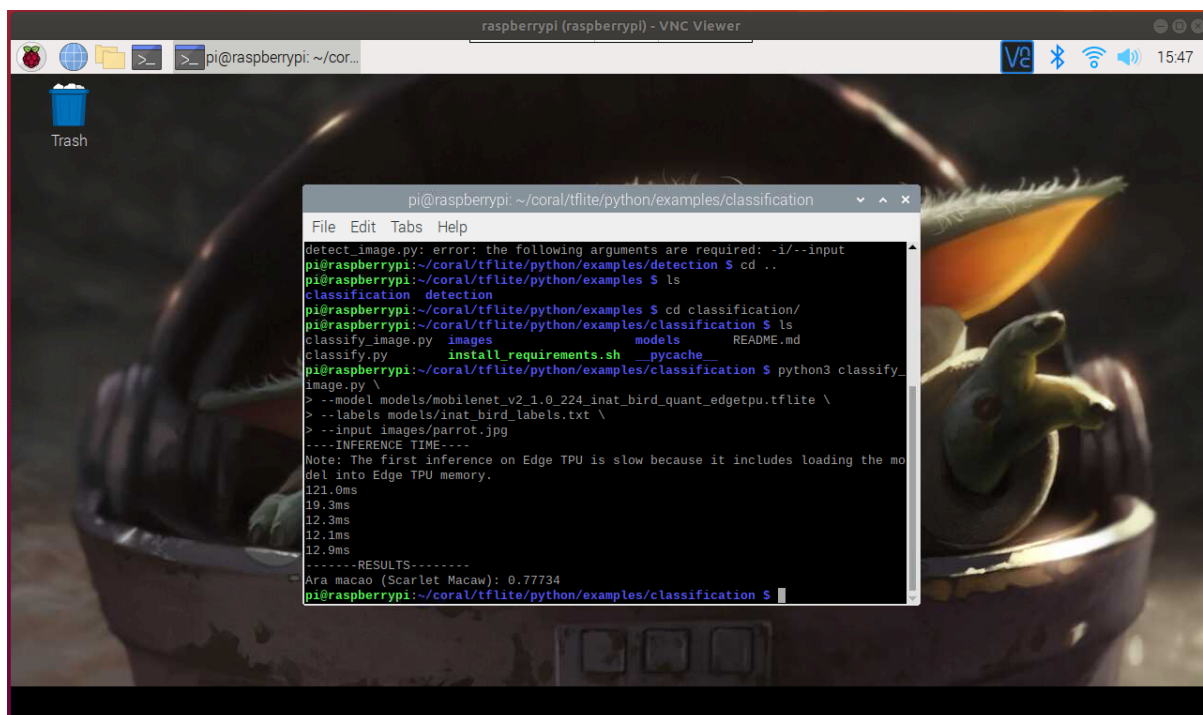
The background of the terminal window shows a close-up image of a robotic gripper holding a small orange object.

Rysunek 11: Poprawne importowanie OpenCV

### 7.2 TensorFlow i TensorFlow Lite

W celu tworzenia sieci neuronowej [1], a następnym uruchomieniu inferencji należało zainstalować bibliotekę do uczenia głębokiego **TensorFlow** [3] oraz bibliotekę pozwalającą na skorzystanie z zakupionego TPU [2]. Na Rys. 12 znajdują się zrzuty ekranu z konsoli pokazujący poprawne działanie inferencji przeprowadzonej przez testowy program załączony z urządzeniem.





Rysunek 12: Przeprowadzona inferencja

### 7.3 Sterownik silników i serwa

Pobrano i skonfigurowano sterownik serwa. Ponadto napisano skrypty, który pozwala na ustawienie serwa w zadanej pozycji.

### 7.4 RealVNC Viewer i Server

W związku z obecną sytuacją postanowiono skorzystać z oprogramowania RealVNC Viewer i Server, które pozwalają na zdalny dostęp do Raspberry. Tym sposobem przy koordynacji członków zespołu możliwy jest zdalny dostęp do pulpitu urządzenia. Wadą tego rozwiązania jest to, że tylko jedna osoba na raz może kontrolować myszą. Zrzut ekranu przedstawiający ten program znajduje się na Rys. 12. W chwili robienia zdjęcia Raspberry znajdował się w innym domu niż komputer z którego robiono zdjęcie.

### 7.5 Sterowanie ręczne

W celu sprawdzenia działania robota napisano prosty skrypt korzystający z zainstalowanych bibliotek, który pozwalał na ręczne sterowanie robotem za pomocą kontrolera DualShock 4.

## 8 Algorytm wykrywania pasa ruchu

W celu sterowania robotem należało napisać funkcję, która służyła do wykrywania pasa ruchu. Jej działanie opiera się na następujących krokach:

1. Pobierz klatkę obrazu
2. Wykryj krawędzie linii
3. Przytnij obraz do pewnego regionu zainteresowania
4. Wykryj linie za pomocą transformacji Hough'a
5. Uśrednij otrzymane linie do dwóch - lewej i prawej

## 8.1 Wykrywanie krawędzi linii

Pierwszym krokiem jest odfiltrowanie jak najwięcej rzeczy, które nie są liniami. Prostym rozwiązaniem jest filtracja z względu na kolor, jako że czerwone linie znacznie odróżniały się od otoczenia. Nie można tego jednak łatwo dokonać korzystając z modelu barw RGB, dlatego należało zmienić model barw na HSV, które pozwala łatwo wybrać zakres kolorów, które mają zostać.

Eksperymentalnie znaleziono wartości progów górnego i dolnego i wynosiły one kolejno: (0, 90, 70) oraz (10, 215, 180). Efektem tej operacji jest czarno-biała maska, gdzie linie zaznaczone są kolorem białym.

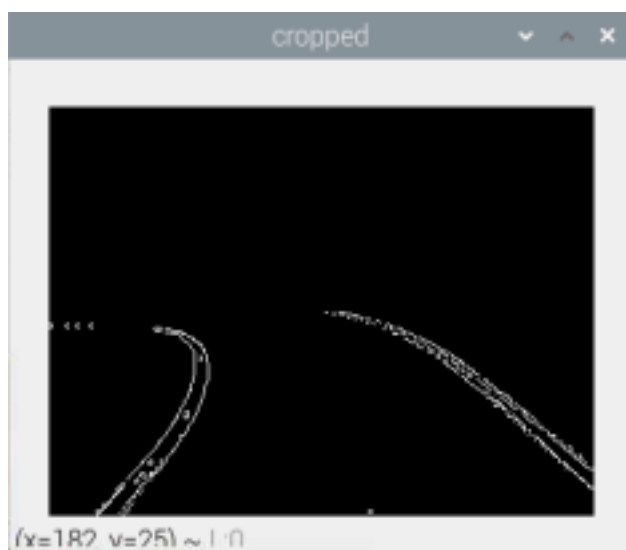
Następnie za pomocą detektora krawędzi Canny wykryto krawędzie linii. Użyte wartości parametrów to: Próg dolny - 200, próg górny - 400. Obraz po tych przekształceniach został przedstawiony na Rys. 13.



Rysunek 13: Wykryte krawędzie

## 8.2 Przycinanie obrazu

W celu zmniejszenia obrazu i dalszemu odfiltrowaniu zbędnych informacji wynikowy obraz wykrywania linii odpowiednio przycięto. Zdefiniowany obszar zainteresowania przyjęto jako dolna  $\frac{1}{2}$  obrazu. Wynik tej operacji przedstawia Rys. 14.



Rysunek 14: Przycięty obraz

### 8.3 Wykrywanie linii

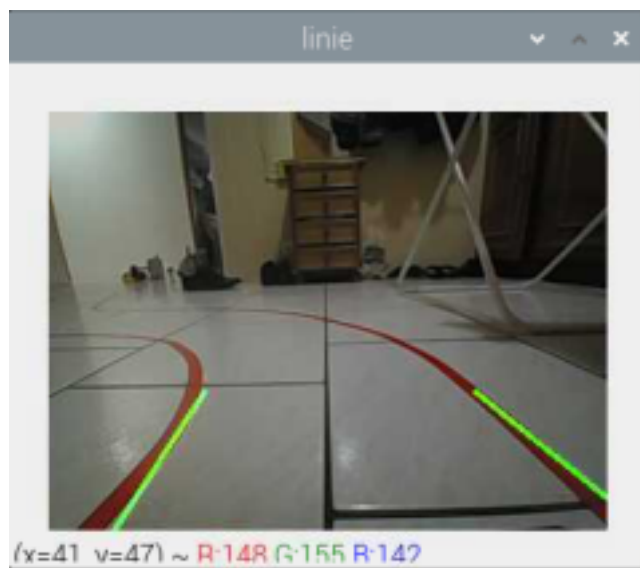
Na tym etapie można już wykryć linie na obrazie, w tym celu zastosowano transformacje Hough'a. Jej wynikiem lista wykrytych linii o zadanych parametrach. Prowadzi to do sytuacji, gdzie jedna krawędź pasa ruchu reprezentowana jest przez wiele krótszych linii. Aby rozwiązać ten problem należy uśrednić otrzymane linie.

### 8.4 Uśrednianie linii

Pierwszym krokiem jest rozróżnienie lewych linii od prawych. Można skorzystać z dwóch zależności:

- Współrzędne x-owe lewych lin powinny być w  $\frac{2}{3}$  obrazu z lewej strony i adekwatnie współrzędne x-owe prawych linii.
- Nachylenie lewych linii jest ujemne, a lin prawych dodatnie.

Zapamiętując każdą linię jako jej niechylenie i punkt przecięcia z osią OY można łatwo na tej podstawie policzyć średnie niechylenie i punkt przecięcia dla lewych i prawych linii. Następnie na podstawie tych parametrów można policzyć współrzędne końców uśrednionych linii. Na Rys. 15 pokazano wykryte linie.



Rysunek 15: Wykryte linie

## 9 Algorytm sterowania

Sterowanie robotem można podzielić na 3 przypadki:

1. Widoczne są oba linie.
2. Widoczna jest jedna lina.
3. Żadne linie nie są widoczne.

Działanie algorytmu zaprezentowane zostało na filmie pod linkiem: <https://www.youtube.com/watch?v=RXpTZvqbF10&feature=youtu.be>

### 9.1 Obie linie są widoczne

Jest to najprostszy przypadek. Zadanie sterowanie polega na określeniu kąta skrętu przedniej osi samochodu. Jako, że kamera znajduje się na środku auta kąt ten można obliczyć na podstawie nachylenia prostej poprowadzonej od środka obrazu do punktu wynikającego z średniej wartości współrzędnej x końców lewej i prawej linii.



## 9.2 Jedna linia widoczna

W tym przypadku do obliczeń brane jest nachylenie widocznej linii.

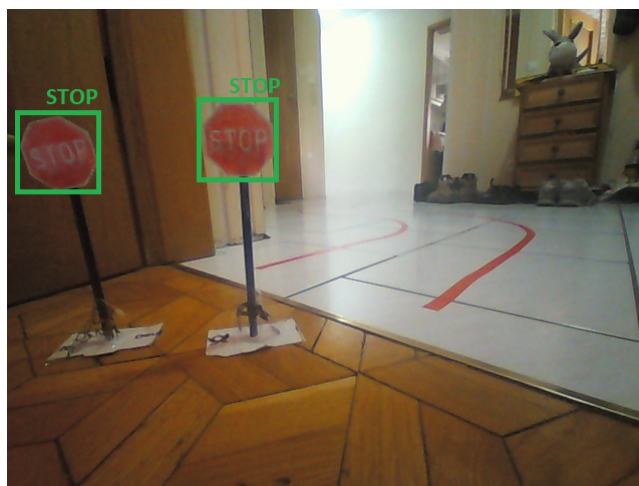
## 9.3 Brak linii

Gdy żadna linia nie jest widoczna to wówczas kąt skrętu przyjmowany jest jako zero.

# 10 Zbieranie danych treningowych

Detekcja znaków została wykonana za pomocą transfer learningu gotowej sieci `ssd_mobilenet_v2_quantized_coco`. W tym celu należało przygotować zdjęcia pod naukę.

Zdjęcia wykonywano za pomocą użytej w robocie kamery w różnych miejscach mieszkania, przy różnych warunkach oświetleniowych. Łącznie zebrano ok 500 zdjęć znaku stop, świateł drogowych i pustej przestrzeni. Następnie należało opatrzyć je w etykiety i w tym celu posłużono się programem `LabelImg`. Przykładowe zdjęcie znajduje się na Rys. 16. Projekt



Rysunek 16: Etykieta znaku stop

# 11 Detekcja obiektów na drodze

W celu wykrywania obiektów na drodze (czerwone światło, zielone światło, znak STOP) skorzystano z splotowych sieci neuronowych oraz TPU. Pierwszym zadaniem było wybór odpowiedniej sieci do transfer learning'u. Użyte TPU ma ograniczoną liczbę gotowych modeli udostępnionych przez firmę Google, z których można korzystać [6]. Nic oczywiście nie stoi na przeszkodzie, aby samemu dokonać kwantyzacji własnego modelu. Co wymaga jednak nieco więcej pracy.

Postanowiono skorzystać z sieci `ssd_mobilenet_v2_quantized_coco`, ponieważ jest całkiem szybkie (czas inferencji = 29 ms) i dokładne (COCO mAP = 22), dodatkowym atutem jest to, że sieć jest już skwantyzowana.

## 11.1 Transfer learning

W celu osiągnięcia dobrych rezultatów trenowania, bez potrzeby żmudnego zbierania tysięcy zdjęć wykrywanych obiektów skorzystano z techniki Transfer Learningu. Wykorzystuje ona wcześniej wytrenowane wagi dla modeli, zakładając, że przystosowały się one już do wyciągania z obrazów szczegółów, które determinują przynależności obiektów do danych klas. Jak już wcześniej wspomniano skorzystano z gotowego modelu sieci `ssd_mobilenet_v2_quantized_coco` wraz z wcześniej wytrenowanymi wagami. Na podstawie zebranych wcześniej dokonano 6000 iteracji nauki, dla rozmiaru batch'a 12. Osiągnięto 97.9% skuteczność trenowania według metryki:

- AP for IoU from 0.50 : 0.05 : 0.95 : 97.9%

Na podstawie wytrenowanego modelu dokonano kwantyzacji grafu.

## 11.2 Kwantyzacja modelu

Graf wyuczonej sieci należało następnie przygotować do formatu obsługiwanego przez TPU. Dokonano tego za pomocą kilku skryptów przygotowanych przez tensorflow:

- `export_inference_graph.py` – tworzy zamrożony graf modelu
- `export_tflite_ssd_graph.py` – tworzy graf przystosowany dla TensorFlow lite
- `tflite_convert` – konwertuje zamrożony graf do skwantyzowanego obsługiwanego przez TensorFlow lite.

Tak otrzymany model należało jeszcze skompilować dla używanego TPU za pomocą programu `Edge_TPU Compiler`.

## 12 System kontroli wersji

W celu realizacji projektu członkowie zespołu wykorzystują system kontroli wersji Git. Link do publicznego zdalnego repozytorium zamieszczonego w serwisie GitHub: <https://github.com/DSLiwowski1/TOFIC>. W repozytorium zamieszczane na bieżąco są postępy w pracach nad projektem dokonywane przez członków zespołu.

## 13 Podsumowanie

Udało się pomyślnie wykonać projekt. Auto porusza się wewnątrz pasa ruchu oraz reaguje na znak stopu i światła drogowe, co widać na filmie pod linkiem: <https://www.youtube.com/watch?v=RXpTZvqbF10&feature=youtu.be>.

Projekt można dalej rozwijać konstruując algorytm sterowania w oparciu o splotowe sieci neuronowe. Spróbowano zrealizować taki sposób sterowania, ale auto niepoprawnie jeździło. Podejrzewa się, że nie potrafiła się ona wyuczyć przez niedostateczną ilość danych (Zebrano ok. 800 zdjęć).

Kolejnym aspektem, w którym można dalej rozwijać projekt jest dodanie wykrywania więcej ilości znaków oraz obiektów, np. samochodów, czy ludzi.

W trakcie realizacji projektu nabyto umiejętności z zakresu projektowania i wykonywania konstrukcji mechanicznej robota wykorzystując druk 3D oraz elementy wycięte z materiału plexa. Zdobyto wiedzę z praktycznego zastosowania przetwarzania obrazów oraz transfer learningu sieci przystosowanej do pracy na TPU. Rozwinięto również umiejętności z zakresu tworzenia sieci neuronowych oraz zbierania i przetwarzania danych niezbędnych do ich wytrenowania oraz skutecznego działania.

## Literatura

- [1] F. Chollet. *Deep Learning. Praca z językiem Python i biblioteką Keras*. Helion, 2019.
- [2] Coral. Instrukcja instalacji oprogramowania do obsługi Edge TPU. <https://coral.ai/docs/accelerator/get-started/>, 2020.
- [3] Google Brain Team. Dokumentacja API Tensorflow. [https://www.tensorflow.org/api\\_docs](https://www.tensorflow.org/api_docs), 2020.
- [4] A. Jaskulski. *Autodesk Inventor 2020 PL / 2020+*. Helion, 2019.
- [5] Raspberry Pi Foundation. Dokumentacja Raspberry Pi. <https://www.raspberrypi.org/documentation/>, 2020.
- [6] Tensorflow. Dostępne modele sieci na TPU. [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/detection\\_model\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md), 2020.