

POLITECHNIKA POZNAŃSKA

WYDZIAŁ AUTOMATYKI, ROBOTYKI I ELEKTROTECHNIKI

INSTYTUT ROBOTYKI I INTELIGENCJI MASZYNOWEJ

ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ



POJAZD JEŻDŻĄCY

PROJEKT PRZEJŚCIOWY

DOKUMENTACJA PROJEKTOWA

PAWEŁ CHUMSKI, 144392

PAWEŁ.CHUMSKI@STUDENT.PUT.POZNAN.PL

KONSTANTY ODWAŻNY, 144514

KONSTANTY.ODWAZNY@STUDENT.PUT.POZNAN.PL

JĘDRZEJ SZCZERBAL, 144510

JEDRZEJ.SZCZERBAL@STUDENT.PUT.POZNAN.PL

PROWADZĄCY:

MGR INŻ. PIOTR SAUER

PIOTR.SAUERO@PUT.POZNAN.PL

14.06.2022

Spis treści

Wstęp	3
1 Zadania	3
1.1 Platforma	3
1.2 Software	5
1.3 Implementacja sterowania	5
1.4 Przesył obrazu	7
2 Test platformy	9
3 Wnioski	10
Bibliografia	10

WSTĘP

Tematem pracy inżynierskiej jest zaprojektowanie systemu sterującego dla pojazdów rolnych bądź ulicznych. Planowane mechanizmy obejmują sterowanie zdalne pojazdem, transmisję obrazu oraz jazdę autonomiczną z rozpoznawaniem poszczególnych obiektów na trasie. Powyższe zadania zrealizowane zostaną na platformie jeżdżącej JetRacer opartej o mikrokomputer Nvidia Jetson. Na ten moment na potrzeby projektu przejściowego zrealizowaliśmy konfigurację platformy, zainstalowanie systemu Linux wraz z środowiskiem ROS oraz sterowanie pojazdem za pomocą kontrolera od PS3 po połączeniu radiowym. Dodatkowo udało nam się zrealizować wysyłanie online obrazu na stronę internetową index.html, gdzie na każdym urządzeniu podpiętym do danej sieci można obserwować obraz z kamery pojazdu.

ZADANIA

W ramach projektu przejściowego zaplanowanymi zadaniami są:

- złożenie platformy symulacyjnej
- zaznajomienie się z możliwościami mikrokomputera oraz instalacja systemu z niezbędnym oprogramowaniem
- implementacja sterowania zdalnego pojazdu
- implementacja zdalnego przesyłania obrazu
- rozpoczęcie przygotowań do szkolenia sieci neuronowej

1.1 PLATFORMA

Obiektem sterowania jest czterokołowy pojazd jeżdżący JetRacer, którego jednostkę obliczeniową stanowi Nvidia Jetson Nano w wersji z 4GB pamięcią RAM. Mikrokomputer dzięki rozwiniętemu systemowi graficznemu jest przystosowany do zadań związanych z przetwarzaniem obrazu. Również możliwe będzie wykorzystanie pełni jego mocy obliczeniowej przy wykorzystaniu sieci neuronowych. Pierwszym postawionym przed nami zadaniem było odpowiednie złożenie platformy oraz jej elementów. [1] Elementy platformy:



Rys. 1. Części platformy JetRacer

Z ważniejszych komponentów wykorzystanych w budowie:

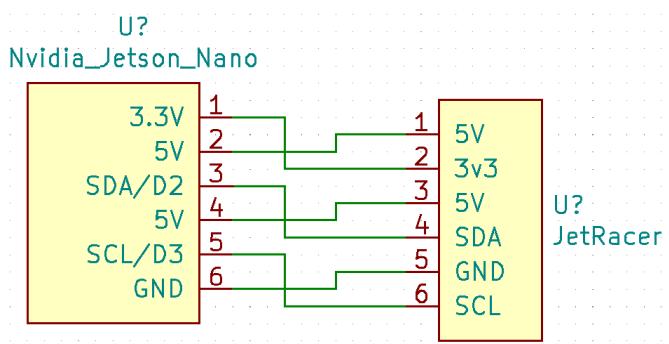
- 2x silnik DC 740rpm z przekładnią
- servomechanizm MG996R
- kamera NVIDIA Jetson Nano IMX219-160 8MP IMX219
- 3x ogniwa li-ion 18650
- zasilacz 12.6V



Rys. 2. Części platformy JetRacer



Rys. 3. Złożona platforma JetRacer



Rys. 4. Schemat połączeń mikrokomputera z platformą [2]

1.2 SOFTWARE

Na komputerze zainstalowany został system Ubuntu 18.04 LTS. [3] Do zaprogramowania zaplanowanych zadań wykorzystano język Python. Dzięki bibliotekom OpenCV oraz TensorFlow lub Torch możliwe jest zaimplementowanie sztucznej inteligencji oraz metod przetwarzania obrazów. Do komunikacji z elementami wykonawczymi sluży środowisko ROS z dystrybucją Melodic oraz worspacem Catkin.

1.3 IMPLEMENTACJA STEROWANIA

Implementację sterowania zaczęliśmy od nauki i zastosowanie prostych algorytmów sterowania pojazdem zawartych w Jupiter Notebook wraz z platformą. Można dostać się do notatnika będąc w tej samej sieci. W tym celu należy w przeglądarce adres IP przydzielony urządzeniu oraz port "8888": **http://<ip>:8888** (aby poznać adres IP należy w konsoli terminala wpisać polecenie **ifconfig**). Za pomocą JupiterNotebook można również zobaczyć jak uzyskać obraz z kamery. Następnie po nauce podstaw użycia sterowania na naszej platformie przeszliśmy do implementacji swoich kodów sterowania za pomocą języka Python i środowiska ROS Melodic i workspaceu Catkin. Za pomocą skryptu **racecar.py** stworzyliśmy klasę naszej platformy której użyjemy do sterowania za pomocą kontrolera bezprzewodowego. Następnie za pomocą skryptów **teleop_gamepad.py** oraz **camera.py** implementujemy sterowanie bezprzewodowe oraz odpowiednio Livestreaming z kamery. Aby odpowiednio uruchomić skrypty należy najpierw uruchomić platformę ROS co dodaliśmy do pliku **.bashrc**, linijką : **source ~/catkin_ws/devel/setup.bash**. Dzięki temu workspace ROSa uruchamia się automatycznie. Następnie aby móc sterować pojazdem za pomocą kontrolera należy otworzyć terminal i uruchomić najpierw:

roscore

Następnie uruchamiamy w nowym terminalu plik z klasą obiektu :

racecar.py

Ostatnim krokiem jest uruchomienie w nowym terminalu skryptu do kontroli pada: (opcjonalnie można również sterować za pomocą klawiatury w skrypcie **teleop.py**)

teleop_gamepad.py



Rys. 5. Sterowanie kontrolerem od PS3

```
01. #!/usr/bin/env python3
02.
03. import rospy
04. from jetracer.nvidia_racecar import NvidiaRacecar
05. from std_msgs.msg import Float32
06.
07. #Initialize car variable and tune settings
08. car = NvidiaRacecar()
09. car.steering_gain = 0.75
10. car.steering_offset = 0
11. car.throttle_gain = 0.8
12. car.steering = 0.0
13. car.throttle = 0.0
14.
15. #Throttle
16. def callback_throttle(throt):
17.     car.throttle = throt.data
18.     rospy.loginfo("Throttle: %s", str(throt.data))
19.
20. #Steering
21. def callback_steering(steer):
22.     car.steering = steer.data
23.     rospy.loginfo("Steering: %s", str(steer.data))
24.
25. #Setup node and topics subscription
26. def racecar():
27.     rospy.init_node('racecar', anonymous=True)
28.     rospy.Subscriber("throttle", Float32, callback_throttle)
29.     rospy.Subscriber("steering", Float32, callback_steering)
30.
31.     rospy.spin()
32.
33. if __name__ == '__main__':
34.     print("Running_racecar.py")
35.     racecar()
```

Listing 1. Skrypt racecar.py

```
01. #!/usr/bin/env python3
02.
03. import rospy
04. import pygame
05. import time
```

```
06. from std_msgs.msg import Float32
07.
08. #Initialize pygame and gamepad
09. pygame.init()
10. j = pygame.joystick.Joystick(0)
11. j.init()
12. print ('Initialized Joystick %s' % j.get_name())
13.
14. def teleop_gamepad():
15.     #Setup topics publishing and nodes
16.     pub_throttle = rospy.Publisher('throttle', Float32, queue_size=8)
17.     pub_steering = rospy.Publisher('steering', Float32, queue_size=8)
18.     rospy.init_node('teleop_gamepad', anonymous=True)
19.     rate = rospy.Rate(10) # 10hz
20.
21.     while not rospy.is_shutdown():
22.         pygame.event.pump()
23.
24.         #Obtain gamepad values
25.         throttle = j.get_axis(1) #Left thumbstick Y
26.         steering = j.get_axis(2) #Right thumbstick X
27.         print("Throttle:", throttle)
28.         print("Steering:", steering)
29.
30.         #Pubblish gamepad values
31.         pub_throttle.publish(throttle)
32.         pub_steering.publish(steering)
33.
34.         rate.sleep()
35.
36. if __name__ == '__main__':
37.     try:
38.         teleop_gamepad()
39.     except rospy.ROSInterruptException:
40.         pass
```

Listing 2. Skrypt teleop_gamepad.py

1.4 PRZESYŁ OBRAZU

Przesyłanie obrazu na stronę internetową również odbywa się za pomocą dystrybucji ROS. Najpierw tak jak wcześniej dla sterowania zaznajomiliśmy się z użyciem kamery na JupiterNotebook. Włączenie odbywa się analogicznie jak dla sterowania. Aby uruchomić kamerę na innym urządzeniu niż lokalnie połączony monitor do mikrokomputera należy w pliku **camera.py** wpisać odpowiedni adres IP naszego urządzenia(polecenie **ifconfig**):

```
if __name__ == "__main__":
    show_camera()
    app.run(host='192.168.1.7') #Host ip address
```

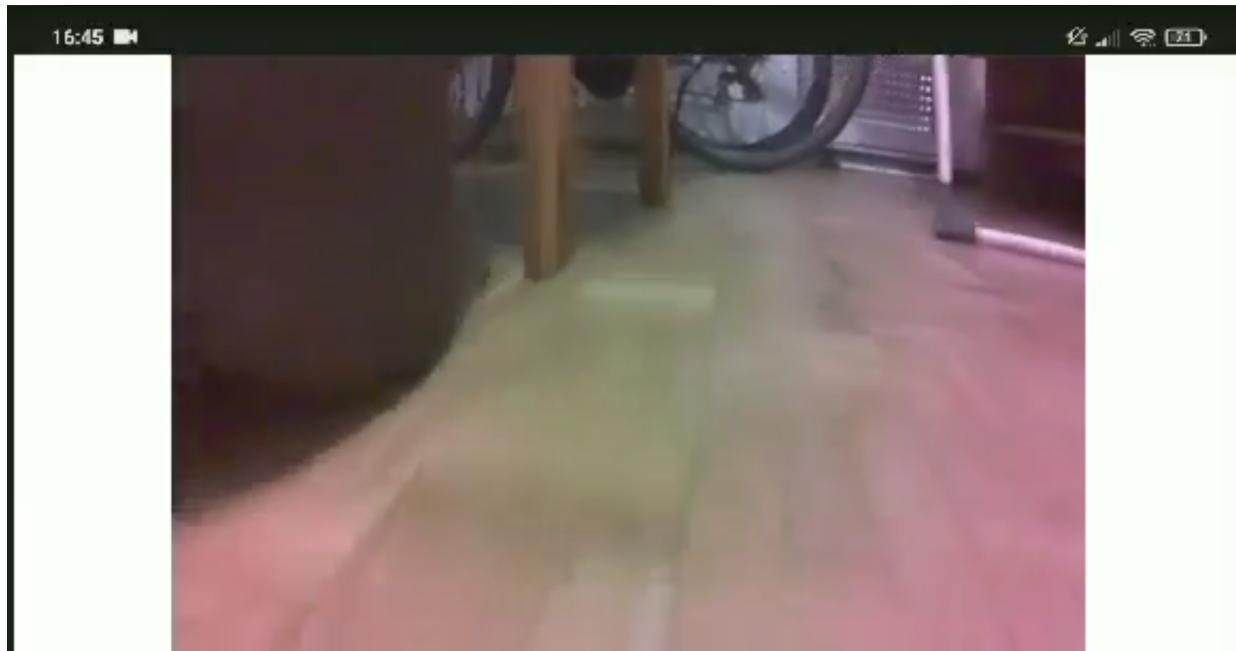
Rys. 6. Ustawienie adresu IP dla strony z obrazem kamery

Kiedy mamy już ustawiony odpowiedni adres IP możemy zacząć odpalać obraz z kamery. Pierwsze kroki są analogiczne jak dla sterowanie odpowiednio w osobnych terminalach: **roscore**, **racecar.py**. Następnie w nowym terminalu uruchamiamy obraz z kamery odpalając skrypt:

camera.py

Teraz możemy obserwować obraz z kamery na stronie, którą włącza się wpisując w przeglądarkę :

<IP>:5000



Rys. 7. Obraz z kamery

```
01. #!/usr/bin/env python3
02.
03. #To view the output stream
04. #Open web browser
05. #http://IP_ADDRESS:5000
06.
07. from flask import Flask, render_template, Response
08. import cv2
09.
10. # gstreamer_pipeline returns a GStreamer pipeline for capturing from the CSI camera
11. # Defaults to 1280x720 @ 60fps
12. # Flip the image by setting the flip_method (most common values: 0 and 2)
13. # display_width and display_height determine the size of the window on the screen
14.
15. app = Flask(__name__) #Flask web server
16.
17. # Gstreamer pipeline settings
18. def gstreamer_pipeline(
19.     capture_width=600,
20.     capture_height=400,
21.     display_width=600,
22.     display_height=400,
23.     framerate=30,
24.     flip_method=0,
25. ):
26.     return (
27.         "nvarguscamerasrc ! "
28.         "video/x-raw(memory:NVMM), "
29.         "width=(int)%d, height=(int)%d, "
30.         "format=(string)NV12, framerate=(fraction)%d/1! "
31.         "nvvidconv flip-method=%d ! "
32.         "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
33.         "videoconvert ! "
```

```
34.         "video/x-raw, format=(string)BGR ! appsink"
35.         % (
36.             capture_width,
37.             capture_height,
38.             framerate,
39.             flip_method,
40.             display_width,
41.             display_height,
42.         )
43.     )
44.
45.
46. #Capture video frame
47. def show_camera():
48.     # To flip the image, modify the flip_method parameter (0 and 2 are the most
49.     # common)
50.     print(gststreamer_pipeline(flip_method=0))
51.     cap = cv2.VideoCapture(gststreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)
52.     if cap.isOpened():
53.         window_handle = cv2.namedWindow("CSI_Camera", cv2.WINDOW_AUTOSIZE)
54.         # Window
55.         while cv2.getWindowProperty("CSI_Camera", 0) >= 0:
56.             ret_val, img = cap.read()
57.             #cv2.imshow("CSI Camera", img)
58.             ret, buffer = cv2.imencode('.jpg', img)
59.             frame = buffer.tobytes()
60.             yield (b'--frame\r\n'
61.                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n') # concat
62.                   # frame one by one and show result
63.
64.             keyCode = cv2.waitKey(30) & 0xFF
65.             # Stop the program on the ESC key
66.             if keyCode == 27:
67.                 break
68.             cap.release()
69.             cv2.destroyAllWindows()
70.         else:
71.             print("Unable to open camera")
72.
73. @app.route('/video_feed')
74. def video_feed():
75.     #Video streaming route. Put this in the src attribute of an img tag
76.     return Response(show_camera(), mimetype='multipart/x-mixed-replace; boundary=frame')
77.
78. @app.route('/')
79. def index():
80.     """Video streaming home page."""
81.     return render_template('index.html')
82.
83. if __name__ == "__main__":
84.     show_camera()
85.     app.run(host='192.168.1.13') #Host ip address
```

Listing 3. Skrypt camera.py

TEST PLATFORMY

W celu ukazania działania naszej platformy przeszliśmy do testu pojazdu co umieściliśmy na filmie. [4] Filmik pokazuje wyniki naszej pracy oraz etap, który udało nam się na ten moment osiągnąć.



WNIOSKI

Zaplanowane zadania na ten semestr zostały zrealizowane. Z rzeczy wymagających poprawy kolejnym krokiem będzie dodatnie offsetu do systemu skręcania ze względu na lekkie przekrzywienie w lewym kierunku. Kolejnym krokiem w realizacji projektu pozostaje wykorzystanie sieci neuronowej do identyfikacji obiektów rejestrowanych przez kamerę.

BIBLIOGRAFIA

1. *Film jak złożyć pojazd.* Dostępne także z: <https://youtu.be/Mn2QYPFADdo>.
2. *Home Page /www.kicad.org.* Dostępne także z: <https://www.kicad.org/>.
3. *Jetracer Image.* Dostępne także z: <https://www.balena.io/etcher/>.
4. *Film z jazdą testową :)* dostępne także z: <https://drive.google.com/file/d/158aSpdD03zHkLfEMy2sxmON-qa4N-M8S/view?usp=sharing>.