

Temat pracy inżynierskiej

Sterowanie pojazdem autonomicznym za pomocą wybranego algorytmu uczenia maszynowego

Autor: Łukasz Kordon

Promotor: dr hab. inż. Tomasz Pajchrowski

Plan prezentacji

1. Cel pracy
2. Kluczowe pojęcia
3. Wykorzystane urządzenia
4. Koncepcja Funkcjonowania
5. Algorytm DDPG
6. Uczenie oraz implementacja modelu
7. Wykrywanie linii
8. Sterowanie napędem
9. Symulator
10. Testy funkcjonalne
11. Podsumowanie i wnioski

Cel pracy

Założeniem pracy inżynierskiej było zbudowanie autonomicznego robota podążającego za linią. Rysunek [1] obrazuje efekt końcowy. Miał on charakteryzować się sterowaniem za pomocą wybranego algorytmu uczenia maszynowego.

Problem został rozłożony na czynniki pierwsze takie jak:

- **Przegląd istniejących rozwiązań.**
- **Projekt części mechanicznej.**
- **Projekt części elektrycznej.**
- **Część programistyczną.**
- **Testy funkcjonalne.**
- **Możliwe drogi rozwoju.**

Rezultatem pracy powinien być nieprzerwany ruch pojazdu po trasie zbudowanej z taśmy.



Rysunek 1: Robot.

Kluczowe pojęcia



Rysunek 2: Gym [6].

- Język programowania Python.
- Biblioteka OpenCV.
- Uczenie maszynowe.
- Gym jako zestaw narzędzi do testowania algorytmów uczenia maszynowego.



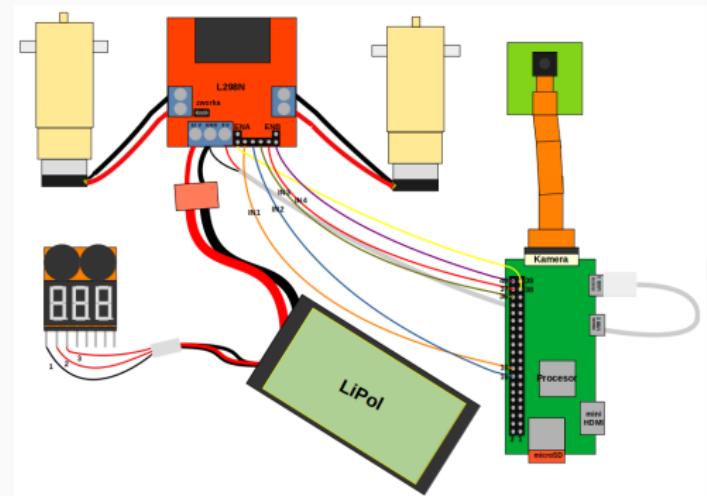
Rysunek 3: Uczenie maszynowe jako obszar sztucznej inteligencji.



Rysunek 5: Logo Python [3].

Rysunek 4: Logo OpenCV [2].

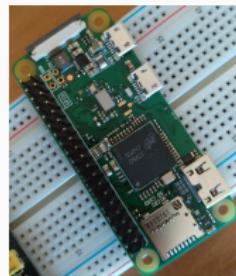
Wykorzystane urządzenia



Rysunek 6: Schemat połączeń.



Rysunek 7: Kamera do Raspberry Pi Rev 1.3.



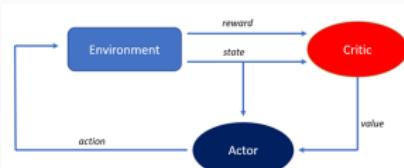
Rysunek 8: Raspberry Pi Zero.

Koncepcja Funkcjonowania



Rysunek 9: Algorytm robota.

Algorytm DDPG



Rysunek 10: Algorytm DDPG [4].

Opis algorytmu:

- Dwa modele aktor oraz krytyk [10].
- Q-learning oraz Policy gradient(Strategia gradientu) [12].
- Deterministyczny.
- DDPG jako DQN dla przestrzeni ciągły.
- Forma pseudokodu [11].

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network  $Q(s, a | \theta^Q)$  and actor  $\mu(s | \theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .  
Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q$ ,  $\theta^{\mu'} \leftarrow \theta^\mu$   
Initialize replay buffer  $R$   
for episode = 1, M do  
    Initialize a random process  $\mathcal{N}$  for action exploration  
    Receive initial observation state  $s_1$   
    for i = 1, T do  
        Select action  $a_i = \mu(s_i | \theta^\mu) + \mathcal{N}_i$  according to the current policy and exploration noise  
        Execute action  $a_i$  and observe reward  $r_i$  and observe new state  $s_{i+1}$   
        Store transition  $(s_i, a_i, r_i, s_{i+1})$  in  $R$   
        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$   
        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'})$   
        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$   
        Update the actor policy using the sampled policy gradient:
```

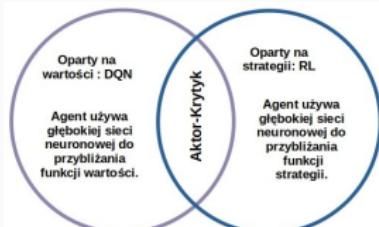
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a | \theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu)|_{s_i}$$

Update the target networks:

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}\end{aligned}$$

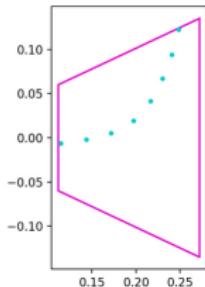
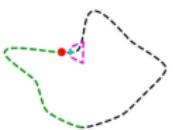
end for
end for

Rysunek 11: Algorytm w formie pseudokodu [5].



Rysunek 12: Algorytm aktor-krytyk.

Uczenie oraz implementacja modelu



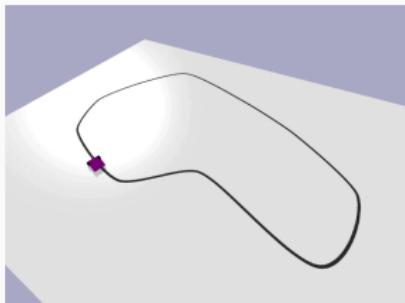
Rysunek 13: Dane wejściowe dla agenta [7].

Przebieg procesu:

- Dane wejściowe w formie punktów [13].
- Budowa sieci neuronowych oraz konfiguracja [14].
- Proces uczenia [15].
- Ewaluacja agenta.
- Implementacja agenta.

Model: "sequential_1"			
Layer (type)	Output Shape	Param #	
flatten_1 (Flatten)	(None, 80)	0	
dense_1 (Dense)	(None, 128)	10368	
dense_2 (Dense)	(None, 128)	16512	
dense_3 (Dense)	(None, 64)	8256	
dense_4 (Dense)	(None, 2)	130	
Total params: 35,266			
Trainable params: 35,266			
Non-trainable params: 0			
Model: "model_1"			
Layer (type)	Output Shape	Param #	Connected to
observation_input (InputLayer)	(None, 5, 16)	0	
action_input (InputLayer)	(None, 2)	0	
flatten_2 (Flatten)	(None, 80)	0	observation_input[0][0]
concatenate_1 (Concatenate)	(None, 82)	0	action_input[0][0]; flatten_2[0][0]

Rysunek 14: Proces uczenia agenta.

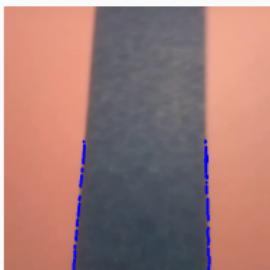


Rysunek 15: Klatka z procesu uczenia.

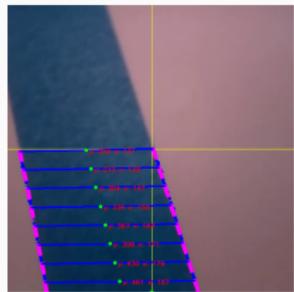
Wykrywanie linii

Etapy przetwarzania obrazu:

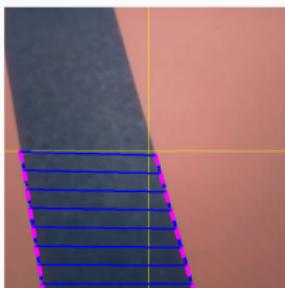
- Przeprowadzenie probabilistycznej transformacji Hough [16].
- Wyznaczenie linii prostopadłych [17].
- Obliczenie punktów centralnych [18].
- Naniesienie informacji dodatkowych [19].



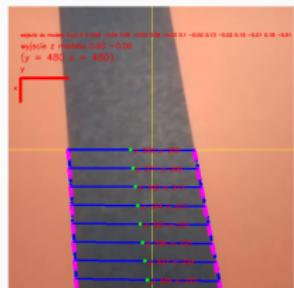
Rysunek 16: Etap 1.



Rysunek 18: Etap 3.



Rysunek 17: Etap 2.



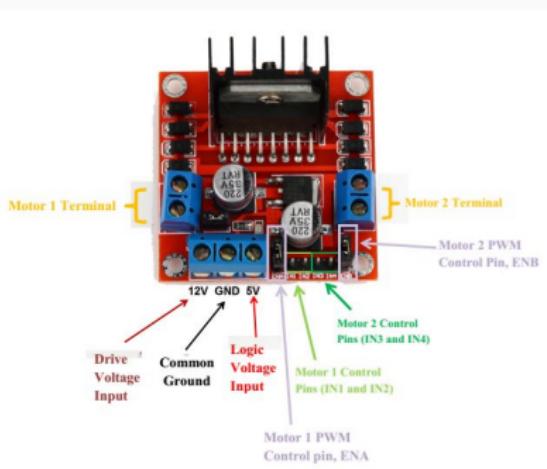
Rysunek 19: Etap 4.

Sterowanie napędem

- Konfiguracja układu sterowania.
- Konwersja danych z modelu DDPG na zakres PWM od 0 do 100.
- Wydanie poleceń silnikom.



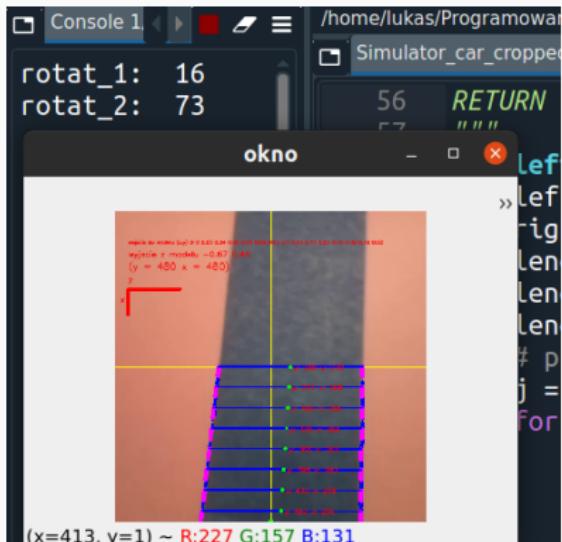
Rysunek 20: Silnik z przekładnią.



Rysunek 21: Dwukanałowy sterownik silników L298N [1].

Symulator

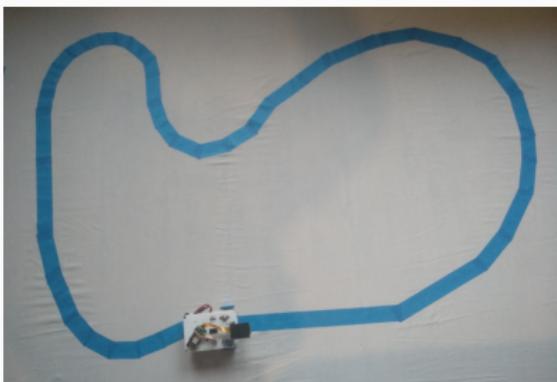
- Opracowanie symulatora w celu przeprowadzanie testów oraz implementacji zmian programu sterującego robotem.
 - Brak konieczności ciągłego przesyłania kodu do Raspberry Pi.
 - Skrócenie czasu wykonania skryptu.



Rysunek 22: Obraz z symulatora.

Testy funkcjonalne

- Rysunek numer [23] ilustruje rzut z góry trasy oraz pojazdu podczas przejazdu.
- Trasa wykonana z tkaniny oraz taśmy malarskiej.
- Test poprawności poruszania się robota.



Rysunek 23: Obraz z symulatora.

- Pojazd spełnia założenia projektowe.
- Wytypowano możliwe drogi rozwoju robota.
- Projekt przyczynił się do zdobycia nowych umiejętności przez autora.

Dziękuję za uwagę!

Źródła zdjęć

- [1] L298n motor driver IC pinout, features, applications and example, . URL
<https://microcontrollerslab.com/l298n-motor-driver-circuit/>.
- [2] Media kit, . URL <https://opencv.org/resources/media-kit/>.
- [3] Python logo, . URL
By www.python.org-<http://www.python.org/community/logos/>, GPL,
<https://commons.wikimedia.org/w/index.php?curid=34991637>.
- [4] M. Buchholz. Deep reinforcement learning. deep deterministic policy gradient (DDPG) algorithm. URL <https://medium.com/@markus.x.buchholz/deep-reinforcement-learning-deep-deterministic-policy-gradient-ddpg>
- [5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. URL
<http://arxiv.org/abs/1509.02971>.
- [6] OpenAI. Gym: A toolkit for developing and comparing reinforcement learning algorithms. URL <https://gym.openai.com>.
- [7] N. Planinšek. nplan/gym-line-follower. URL
<https://github.com/nplan/gym-line-follower>. original-date:
2018-11-25T20:24:15Z.