

Laboratorium 1

Duckiebot liniefollower

1 Cele ćwiczenia

Celem ćwiczenia jest implementacja dwóch rodzajów regulatorów których zadaniem realizacja algorytmu *liniefollower* t.j. sterowaniem robotem *Duckiebot* w taki sposób aby poruszał się on po wyznaczonej trasie.

2 Wymagane kwalifikacje osób realizujących ćwiczenie

2.1 Przygotowanie do zajęć

Do realizacji ćwiczenia konieczna jest znajomość następujących zagadnień

- znajomość *Robot Operation System* w tym tworzenie węzłów, publikowanie, subskrybowanie wiadomości, tworzenie plików startowych typu *.launch
- znajomość oprogramowania *docker* w tym tworzenie obrazów, uruchamianie kontenerów, podłączanie katalogów do kontenera
- znajomość biblioteki *OpenCV2*
- znajomość biblioteki *tensorflow*
- znajomość regulatora PID oraz jego dyskretnej implementacji
- znajomość środowiska *Google Colab*

3 Opis stanowiska laboratoryjnego

Do każdego ćwiczenia przygotowana jest instrukcja (konspekt) zawierająca wiadomości teoretyczne i zadania do wykonania. Konspekt musi być przeczytany przed zajęciami (w domu), tak aby na zajęciach nie tracić czasu na jego zrozumienie. Czas na zajęciach powinien być poświęcony na przeanalizowanie przykładów i zrobienie zadań do sprawozdania.

Wszystkie zadania, które należy wykonać podczas ćwiczeń laboratoryjnych są opisane w instrukcjach do tych ćwiczeń. Zadania zostały tak sformułowane, aby była możliwość ich wykonania w czasie trwania laboratorium.

4 Opis stanowisk laboratoryjnych

Do realizacji ćwiczenia potrzebny jest komputer PC z zainstalowanym systemem Ubuntu 20 wraz z zainstalowanymi aplikacjami takimi jak:

- docker, docker-compose, <https://www.docker.com/>
- Visual Studio Code, <https://code.visualstudio.com/>
- duckietown-shell,
https://docs.duckietown.com/daffy/opmanual-duckiebot/setup/setup_laptop/setup_dt_shell.html
- przeglądarka internetowa (najlepiej Chrome)
- Google Colab, <https://colab.research.google.com/>

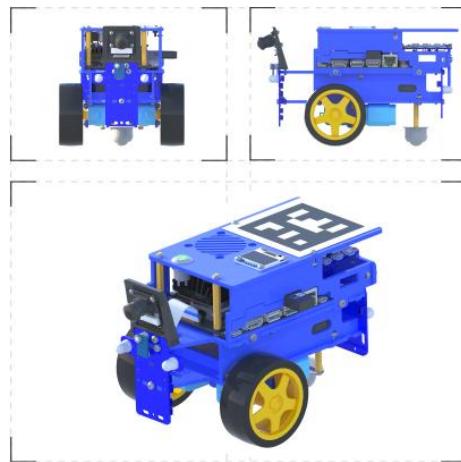
Drugim elementem niezbędnym do realizacji ćwiczenia jest robot Duckiebot wersja DB21M.

5 Wymagane informacje do realizacji ćwiczenia

5.1 Opis robota Duckiebot

Robot Duckiebot jest niewielkim, dwu kołowym robotem zdolnym do poruszania się po płaskich nawierzchniach. Do napędu robota wykorzystano dwa moduły zawierające: silnik elektryczny, przekładnię i enkoder. Robot Duckiebot wyposażony jest w następujące sensory: kamera RGB, czujnik odległości oraz czujnik bezwładnościowy IMU. Do sterowania robotem wykorzystany jest komputer JetsonNANO. Dodatkowo

robot Duckiebot wyposażony jest w cztery diody RGB które mogą być wykorzystane np. do sygnalizowania stanu w jakim aktualnie znajduje się robot. Robot Duckiebot jest zaprojektowany do autonomicznego sterowania i może poruszać się bez udziału człowieka, na podstawie danych z kamery i odpowiednich algorytmów sterowania. Rysunek 1.1 przedstawia robota Duckiebot w trzech rzutach natomiast rysunek 1.2 przedstawia jego schemat blokowy.



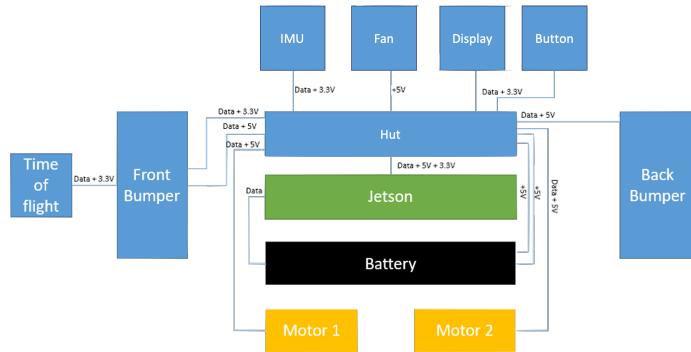
Rysunek 1.1: Robot Duckiebot (<http://duckietown.org>).

Konsekwencją zastosowania dwóch niezależnych silników elektrycznych w konstrukcji robota Duckiebot jest sposób w jaki kontroluje się kierunek poruszania się robota. Robot Duckiebot wykonuje zakręty zmieniając prędkości obrotowe silników elektrycznych w zależności od tego w jaką stronę chce zakręcić oraz jak duży ma być promień skrętu. Do sterowania robotem wykorzystywane są dwa sygnały: v -wartość prędkości liniowej oraz ω -wartość prędkości obrotowej wzduł osi pionowej Z robota, zob. rys. 1.3.

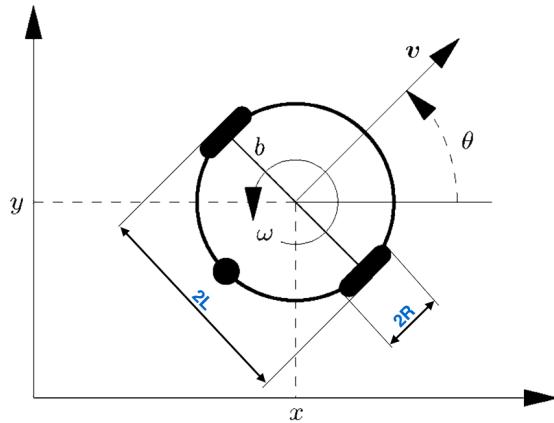
Znając wartości (v, ω) (wyznaczone np. przez regulator) możemy obliczyć prędkości obrotowe kół robota z wzorów (1.1) i (1.2):

$$\omega_L = \frac{v - L\omega}{R} \quad (1.1)$$

$$\omega_R = \frac{v + L\omega}{R} \quad (1.2)$$



Rysunek 1.2: Schemat blokowy robota Duckiebot (<http://doc.duckietown.org>).

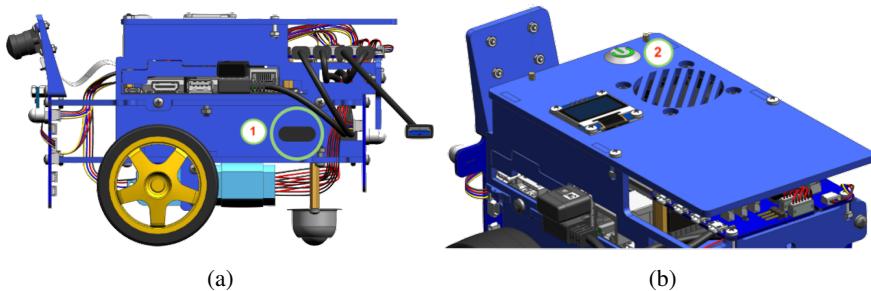


Rysunek 1.3: Układ kinematyczny robota Duckiebot.

Tego typu układ sterowania robotem Duckiebot jest bardzo często wykorzystywany w robotyce i nosi nazwę *differential drive robot* [1]. Zaletą takiego układu kinematycznego ruch robota Duckiebot jest prostota konstrukcji (brak mechanizmów osi skrętnych kół) natomiast wadą jest konieczność ciągłego kontrolowania prędkości obrotowej kół robota (ω_R, ω_L) tak aby poruszał się on po zadanej trasie.

Aby włączyć robota Duckiebot należy nacisnąć przycisk nr 1 zaznaczony na ry-

sunku 1.4a. Po kilkunastu sekundach robot Duckiebot się włączy co będzie sygnalizowane pojawiением się danych na górnym wyświetlaczu. W celu wyłączenia robota Duckiebot należy przytrzymać przycisk nr 2 zaznaczony na rysunku 1.4b przez okres około 3-5 sekund. Po upływie czasu około 10 sekund robot Duckiebot się wyłączy.



Rysunek 1.4: Włączanie i wyłączanie robota Duckiebot.

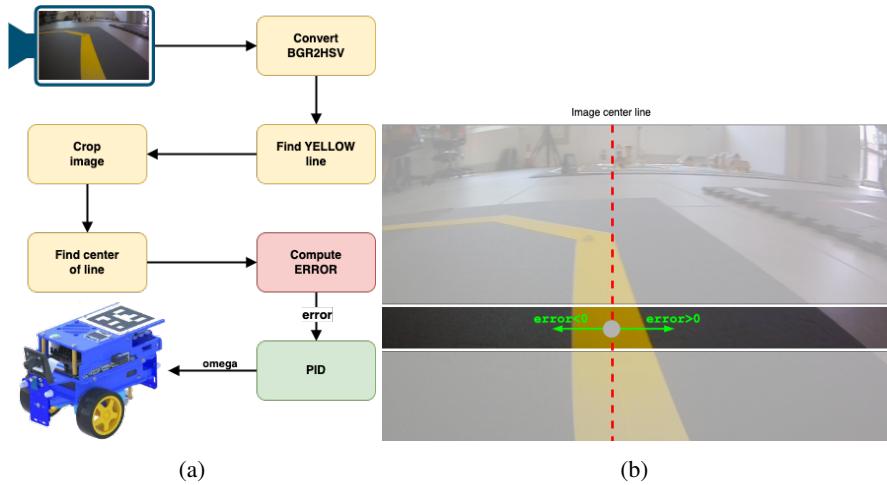
Dokładny opis budowy, sposobu użytkowania robota Duckiebot można znaleźć w dokumentacji projektu pod adresem [2].

5.2 Realizacja z regulatorem PID

Klasyczny układ sterowania robotem Duckiebot wykorzystuje regulator PID. Zadniem układu sterowania jest generowanie takiego sygnału sterowania u_i aby minimalizować wartość sygnału błędu e_i , gdzie i - kolejne chwile czasowe. Na rysunku 1.5a przedstawiona schemat blokowy układu sterowania robotem Duckiebot z wykorzystaniem regulatora PID oraz kamery robota jako sensorem.

Sygnal błędu e_i zdefiniowany jest jako różnica pomiędzy środkiem osi pionowej obrazu z kamery a punktem znajdującym się w analizowanym obszarze obrazu zawierającym linię wzdułż której powinien poruszać się robot Duckiebot, zob. rys. 1.5b.

Układ sterowania robota Duckiebot jest systemem dyskretnym (zaimplementowanym na komputerze JetsonNANO) dlatego należy wykorzystać wersję dyskretną regulatora PID. Uwzględniając rodzaj napędu robota Duckiebot (napęd różnicowy *differential drive robot*) sygnał sterowania u_i wygenerowany przez regulatora PID odpowiada prędkości obrotowej ω_i wzdułż osi Z robota.



Rysunek 1.5: Schemat układu sterowania robotem Duckiebot z wykorzystaniem regulatora PID 1.5a, Określenie błędu dla układu sterowania realizującego funkcjonalność *line follower* dla robota Duckiebot 1.5b.

Dyskretny regulator PID

Jednym z najlepiej poznanych i najczęściej stosowanych regulatorów w układach automatycznego sterowania jest regulator PID. Wynika to przede wszystkim z jego uniwersalności, łatwości realizacji zarówno w formie analogowej jak i cyfrowej oraz w miarę prostego i intuicyjnego sposobu strojenia parametrów regulatora PID. Regulator PID w wersji dyskretnej który zastosowano w układzie sterowania robotem Duckietown dany jest równaniem:

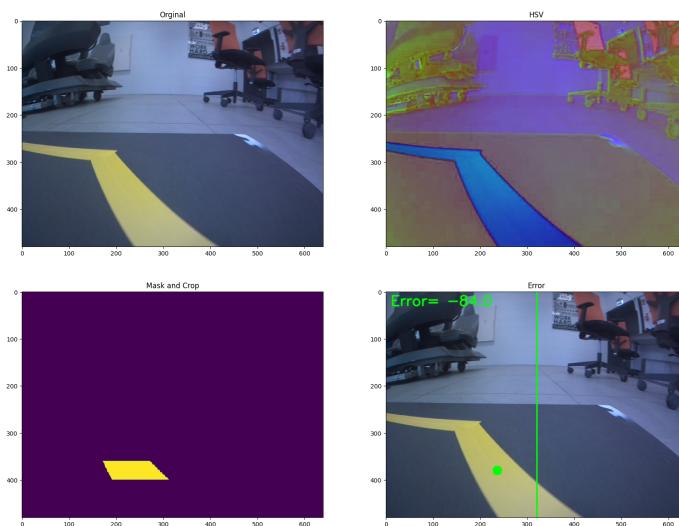
$$u_i = K_p e_i + K_i \sum_{i=0}^k e_i \Delta t + K_d \frac{e_i - e_{i-1}}{\Delta t} \quad (1.3)$$

gdzie: K_p wzmocnienie części proporcjonalnej, K_i wzmocnienie części całkującej, K_d wzmocnienie części różniczkującej, e_i błąd w i -tej chwili czasu, Δt krok czasowy [3].

Pomiar błędu

Wartość sygnału błędu e_i wyznaczana jest na podstawie obrazów przesyłanych przez kamerę robota Duckiebot. W celu określenia wartości sygnału e_i konieczne jest wy-

konanie kilku operacji na przesłanych obrazach (zob. rys. 1.5a). Pierwszą operacją jest konwersja obrazu z przestrzeni barw RGB do przestrzeni HSV (*Hue, Saturation, Value*). Takie przekształcenie przestrzeni barw obrazów ma na celu przede wszystkim lepsze rozróżnianie i rozpoznawanie kolorów. Kolejną operacją jest znalezienie na obrazie linii określonym kolorze wzdłuż której ma poruszać się robot Duckiebot. Taka operacja ma na celu pozostawienia na obrazie tylko linii wyznaczającej trasę robotą i usunięcie z obrazu wszystkich pozostałych elementów (np. tło). Kolejna operacją jest ograniczenie obszaru analizowanego poprzez wykonanie operacji przycięcia. Operacja ta ma na celu dokładne znalezienie środka lini wzdłuż której ma poruszać się robot Duckiebot. Środek lini znajdywany jest jako współrzędne środka ciężkości otrzymanego, w wyniku poprzednich przekształceń, lini wyznaczającej trasę robota Duckiebot. Na rysunku 1.6 pokazano rezultat poszczególnych przekształceń wykonywanych na obrazu z kamery robota w celu wyznaczenia wartości błędu położenia e_i .

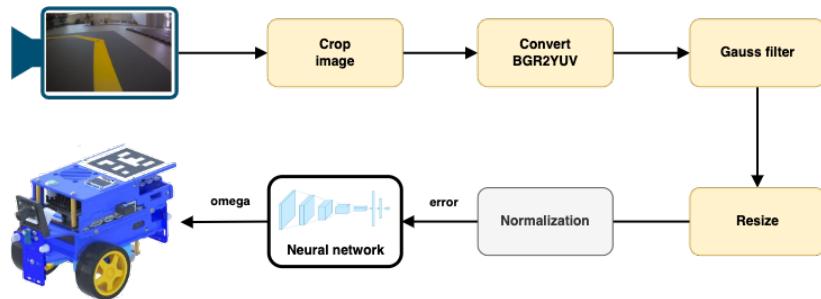


Rysunek 1.6: Operacje wykonywane na obrazach z kamery robota w celu wyznaczenie błędu położenia e_i .

Na końcu wartość błędu położenia e_i jest normalizowana do przedziału $e_i \in [-1; 1]$. Ma to na celu uniezależnieniu się od rozmiarów (szerokość, wysokość) obrazów przesyłanych z kamery.

5.3 Realizacja z regulatorem CNN

Drugi ze sposobów generowania sygnału sterowania u_i wykorzystuje konwolucyjną sieć neuronową (*Convolutional Neural Network CNN*). Schemat blokowy układu regulacji wykorzystującego sieci neuronowe przedstawiono na rys. 1.7.



Rysunek 1.7: Schemat układu sterowania robotem Duckiebot z wykorzystaniem konwolucyjnych sieci neuronowych.

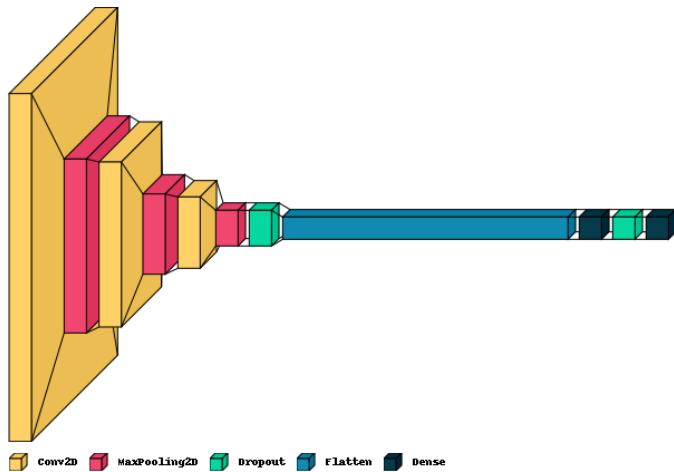
Konwolucyjne sieci neuronowe są bardzo skuteczne szczególnie w zadaniach rozpoznawania wzorców i przetwarzania obrazów. Sieci CNN wykorzystują wielokrotnie operację konwolucji której użycie ma na celu wydobywanie z obrazów wejściowych charakterystycznych cech (krawędzie, tekstury, kształty) na podstawie których klasyczna sieć neuronowa może przeprowadzić wnioskowanie "co znajduje się na obrazie". Konwolucyjne sieci neuronowe są stosowane w różnych dziedzinach, takich jak rozpoznawanie obrazów, analiza tekstu, przetwarzanie mowy, rozpoznawaniem twarzy, samochodów, diagnozowaniem chorób na podstawie obrazów medycznych, analizą obrazów satelitarnych itp. [4], [5], [6].

W przypadku wykorzystania do sterowania robotem Duckiebot sieci CNN, sieć taka na wejściu otrzymuje obrazy z kamery (po wykonaniu kilku przekształceń) na podstawie których generuje sygnał sterowania u_i który jest interpretowany jako ω_i prędkość obrotowa wzdłuż osi Z robota, zob. rys. 1.7.

Regulator CNN

Proponowana struktura konwolucyjnej sieci neuronowej przedstawiona jest na rys. 1.8 i należy zaznaczyć iż jest to jedna z możliwych do wykorzystania struktur sieci.

Sieć składa się z pierwszej warstwy konwolucyjnej, następną warstwa jest warstwa *MaxPooling*. Jej zdaniem jest zredukowanie rozmiaru danych wejściowych. Kolejne



Rysunek 1.8: Struktura konwolucyjnej sieci neuronowej wykorzystanej do sterowania robotem Duckiebot.

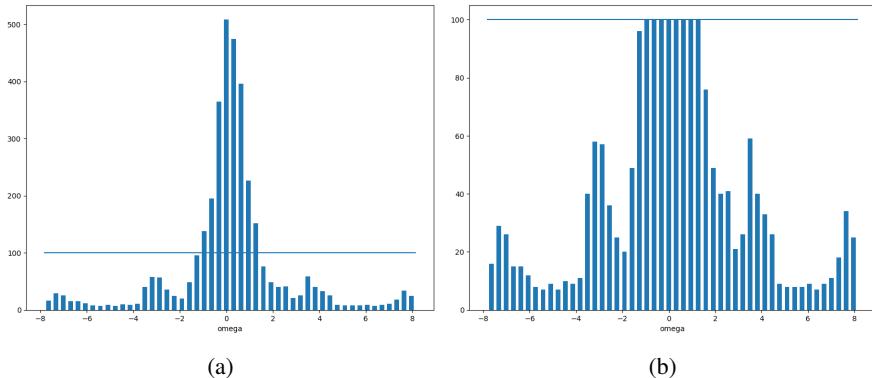
warstwy sieci to warstwa konwolucyjnej i warstwa *MaxPooling* - taki układ warstw jest powtórzony dwukrotnie. Za każdym razem warstwy konwolucyjne ekstrahują kolejne cechy obrazu wejściowego istotnie z punktu wiedzenia sterowania robotem Duckiebot, natomiast warstwy *MaxPooling* redukują rozmiar danych. Zadaniem następnej warstwy *Dropout* losowe wyłączanie poszczególnych neuronów sieci, tak aby zapobiec zjawisku *overfitting* oraz wspomóc zdolność generalizacji sieci neuronowej. Przed przesłaniem danych na wejście właściwej sieci neuronowej muszą one zostać spłaszczone co jest zadaniem warstwy *Flatten*. Przekształca ona dane wielowymiarowe w jednowymiarowy wektor wejściowy do następnej warstwy *Dense*. Warstwa *Dense* jest odpowiedzialna za wykonywanie klasyfikacji, regresji lub innych zadań uczenia maszynowego. Wyjściem z sieci CNN jest wartość sygnału ω z przedziału $[-8 ; 8]$ dlatego funkcją aktywacjii jest z wyjścia warstwy *Dense* jest funkcja *tanh*.

5.4 Przygotowanie danych

Każda sieć neuronowa jest tak dobra jak dobrymi danymi została wytrenowana. Stąd też bardzo ważnym etap pracy z sieciami neuronowymi jest przygotowanie dobrego zbioru danych uczących. W przypadku robota Duckiebot i zadania które ma realizować (*line follower*) dane uczące dla sieci składają się z par: obraz z kamery i odpowia-

dająca mu prawidłową wartość ω , zob. rys. 1.7. Dane konieczne do trenowania sieci można uzyskać sterując robotem Duckiebot ręcznie (np. za pomocą gamepad) lub wykorzystać opracowany wcześniej układ sterowania z regulatorem PID dodając skrypt zapisujący dane.

Oprócz odpowiedniej ilości zalogowanych danych do uczenia sieci neuronowej, ważne jest aby dane te były odpowiednio różnorodne - czyli aby obejmowały jak największą liczbę przypadków, sytuacji, zdarzeń w jakich może znaleźć się robot Duckiebot. Na rysunku 1.9a przedstawiono histogram przykładowego zbioru danych jaki został zalogowany.

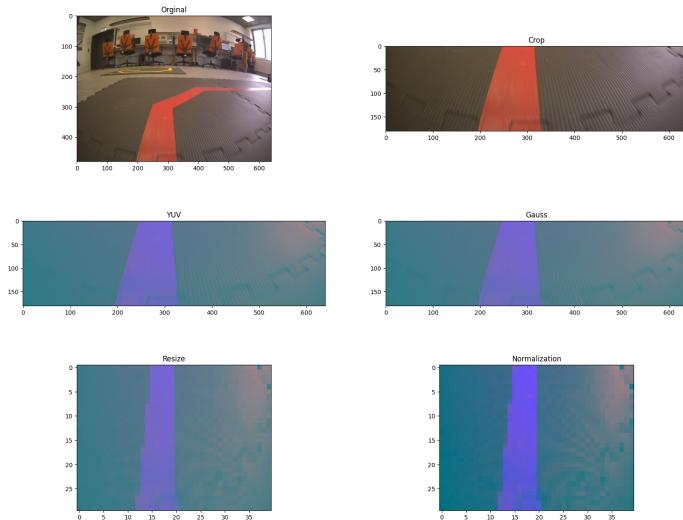


Rysunek 1.9: Histogram zalogowanych danych przed analizą a) i po analizie b).

Wyraźnie widać, że dominującą wartością w tym zbiorze danych dla zmiennej ω są wartości w bliskie wartości 0 i jeśli użyć takich danych do trenowania sieci CNN która ma sterować robotem Duckiebot to prawdopodobnie robot bardzo dobrze jeździł by prosto, natomiast miał by problemy z zakrętami. W takim przypadku, po analizie danych należy usunąć ze zbioru uczącego dane których wartości znaczco dominują nad innymi wartościami otrzymując bardziej reprezentatywny zbiór danych uczących, zob. rys. 1.9b.

Podobnie jak w przypadku sterowania robotem Duckiebot z wykorzystaniem regulatora PID tak i w przypadku sterowania robotem Duckiebot z wykorzystaniem sieci neuronowych konieczne jest odpowiednie przygotowania obrazów z kamery. Poszczególne etapy przetwarzania obrazów z kamery robota Duckiebot przedstawiono na rysunku 1.7 (bloki o kolorze żółtym) i są to: przycinanie obrazu do interesującego nas obszaru, konwersja do przestrzeni barw YUV (Y-luminacja, UV-barwa kolorów, chrominancja), filtrowanie z wykorzystaniem filtra Gauus-a, zmiana rozmiaru obrazu i nor-

malizacja wartości poszczególnych kanałów obraz do przedziału [0 1]. Rezultat poszczególnych operacji można zobaczyć na rysunku 1.10.

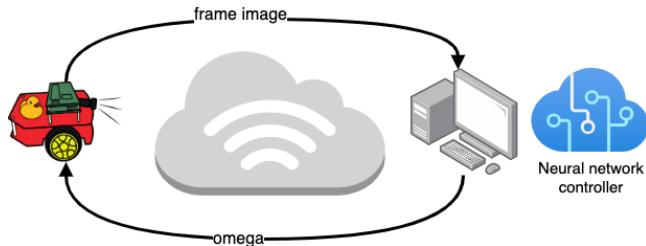


Rysunek 1.10: Operacje wykonywane na obrazach z kamery robota Duckiebot sterowanym przed zastosowaniem regulatora CNN.

6 Przebieg ćwiczenia

Oprogramowanie robotów Duckietown zostało zaimplementowane z wykorzystaniem framework-a Robot Operation System (*ROS*) [7]. Jest on obecnie jednym z najpopularniejszych framework-ów do programowania różnego rodzaju robotów. Konsekwencją wybrania framework-a ROS jest wybór języka programowania i w przypadku robotów Duckiebot jest to Python. Projekt Duckietown definiuje również sposób tworzenia własnego oprogramowania dla robotów i wymusza stosowanie technologii konteneryzacji. Jest to bardzo dobra praktyka zgodna z ówczesnymi trendami tworzenia oprogramowania. Dokładny opis w raz z przykładami tworzenia oprogramowania dla robotów Duckiebot jest bardzo dobrze opisana w dokumentacji projektu [8]. Wykorzystanie framework-a ROS umożliwia również efektywnie tworzenie i testowania oprogramowania bez konieczności ciągłego wgrywania kolejnych wersji do robota Duckiebot. Konfigurując odpowiednio komputer host-a, tak aby łączy się z serwerem ROS-na uru-

chomionym na wybranym robocie Duckiebot można w sposób wygodny i efektywny implementować i testować własne oprogramowanie, zob. rys. 1.11.



Rysunek 1.11: Konfiguracja środowiska deweloperskiego dla robota Duckiebot.

Proponowane regulatory należy zaimplementować jako węzły ROS-a. Węzły powinny odczytywać obrazy z kamery robota (przesyłane jako wiadomości ROS) a jako wyjście generować sygnał sterowania ω . Dla uproszczenia sterowania robotem Duckiebot można przyjąć, że porusza się on ze stałą prędkością liniową $v = const$.

Konwolucyjne sieci neuronowe można tworzyć, trenować oraz testować wykorzystując moduły *keras* i *tensorflow* języka Python. Zaletą modułów *keras* i *tensorflow* jest również bardzo duża ilość przykładów.

Należy zwrócić również uwagę prawidłowe przygotowanie danych do uczenia sieci neuronowych, każdemu obrazowi z kamery musi odpowiadać jedna, poprawna wartość sygnału ω .

6.1 Przygotowanie środowiska

Przed przystąpieniem do realizacji zadań należy przygotować środowisko pracy. Należy pobrać plik DTF.zip i rozpakować go w dowolnej lokalizacji katalogu użytkownika. Po rozpakowaniu archiwum należy przejść do katalogu DTF. W celu utworzenie obrazu docker-a którego zadaniem będzie realizacja zadania należy wydać komendę:

```
dts devel build -f
```

Wszystkie komendy powłoki muszą być wykonywane w katalogu DTF /

Wywołanie powyższej instrukcji spowoduje utworzenie obrazu dockera

```
duckietown/dtf:latest-amd64
```

oraz zainstalowanie w nim wszystkich niezbędnych bibliotek wyspecyfikowanych w plikach `dependecies-apt.txt` oraz `dependecies-py3.txt`, oraz utworzenie wewnątrz obrazu katalogu `DTF/` w miejsce którego następnie będzie montowany katalog `DTF/` komputera hosta w trakcie uruchamiania kontenera.

Jeśli chcemy uruchomić kontener z przygotowanym oprogramowaniem na komputerze robota Duckiebot to należy wcześniej przygotować odpowiedni obraz dockera który odpowiada architekturze procesora na jakim będzie on wykonywany. Ponieważ roboty Duckiebot wersji DB21M wykorzystują komputery JetsonNANO z architekturą arm to instrukcja przygotowująca obraz dokcera gotowy do uruchomienia na robocie Duckiebot mam postać:

```
dts devel build -H ROBOT_NAME -f
```

 Wyrażenie `ROBOT_NAME` zawsze trzeba zastąpić nazwą robota Duckiebot, można ją znaleźć na górnjej części robota (np. D1, D2, ...)

6.2 Uruchomienie oprogramowania

Uruchomienie kontenera wykorzystującego utworzony obraz wykonywane jest wywołanie w oknie terminala skryptu:

```
./scripts/container_local_start.sh ROBOT_NAME
```

Po wywołaniu powyższego skryptu uruchamiany jest kontener z zamontowanym w kontenerze katalogu `DTF/` i oczekuje na dalsze komendy. Chcąc uruchomić zaimplementowany kod należy w uruchomionym kontenerze wywołać komendę:

```
roslaunch [launch-file]
```

gdzie `[launch-file]` to pliku `*.launch` wraz z ścieżką, np.:

```
roslaunch package/sensing/launch/  
lateral_position_error_node.launch
```

uruchamia węzeł `lateral_position_error_node` pakietu `sensig`.

Każdy plików `*.launch` w projekcie Duckietown zawiera na początku następujące dane:

```
<arg name="veh" value="$(env VEHICLE_NAME)" />  
<arg name="pkg_name" value="..."/>  
<arg name="node_name" default="..." />
```

W argumencie `veh` zapisana jest nazwa robota np. `d1`, `d2` itp.. Nazwa ta jest ustawiana automatycznie w momencie tworzenia i uruchamiania kontenera z naszym oprogramowaniem. Argument `pkg_name` zawiera nazwę pakietu a argument `node_name` zawiera nazwę węzła do uruchomienia. Zgodnie z zasadami tworzenia kodu w projekcie Duckietown, każdy węzeł musi posiadać swój własny plik typu `*.launch` który konfiguruje parametry pracy węzła.

6.3 Testowania działania oprogramowania

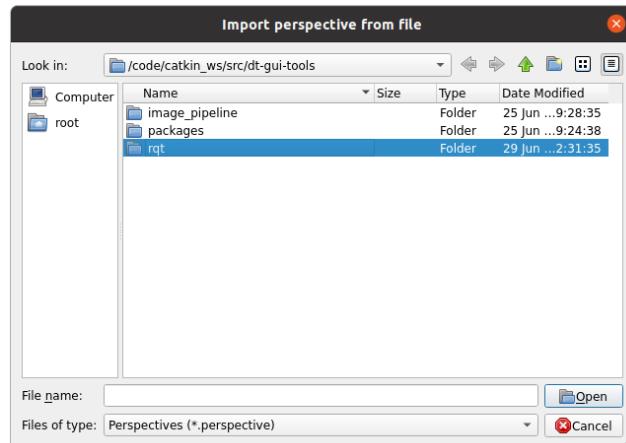
Chcąc sprawdzić poprawność działania oprogramowania najelpiej skorzystać z wbudowanego narzędzia `rqt` pakietu ROS. W celu uruchomienia programu `rqt` należy wykonać instrukcję (w nowym oknie terminala):

```
./scripts/container_gui_tools.sh ROBOT_NAME
```

i w uruchomionym kontenerze wykonać instrukcję `rqt`. Po uruchomieniu się programu `rqt` należy zimportować konfigurację wyświetlanych danych na ekranie. Można to zrobić wybierając opcje z menu

Perspectives->Import

a następnie z lokalizacji pokazanej na rysunku 1.12 wybrać odpowiedni plik konfigu-

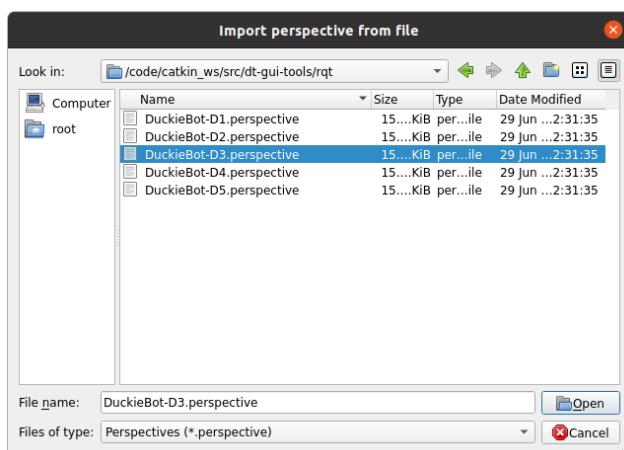


Rysunek 1.12: Importowanie ustawień w `rqt`.

racyjny zob. 1.13, np.:

DuckieBot-D3.perspective

 Uwaga! Dla każdego robota Duckiebot jest dostępna dedykowana konfiguracja programu rqt.



Rysunek 1.13: Lista dostępnych konfiguracji programu rqt.

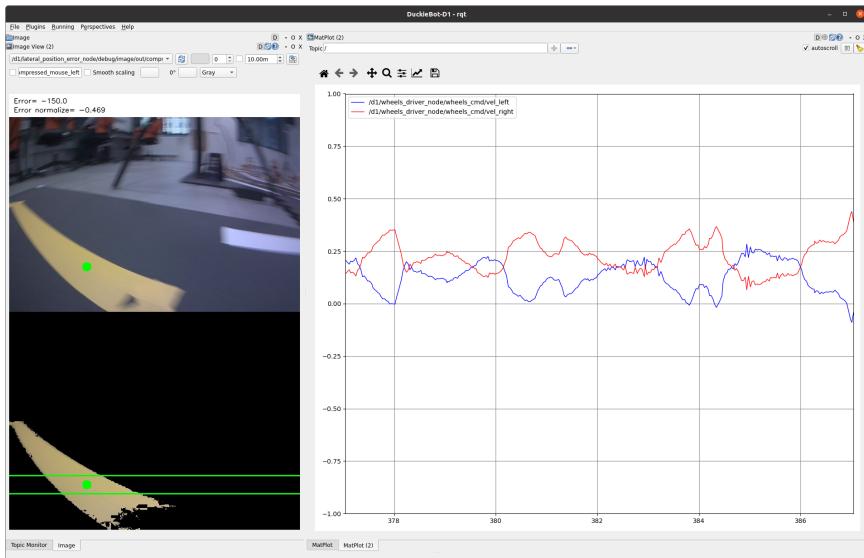
Po poprawnym załadowaniu wskazanej konfiguracji ekran programu rqt powinien wyglądać jak na rysunku 1.14:

6.4 Implementacja pakietu `sensing`

Pakiet `sensing` zawiera implementację węzła `lateral_position_error_node` który subskrybuje skompresowane obrazy z kamery `~image/in/compressed` i wykonując odpowiednie przekształcenia obrazu, patrz rys 1.5a wyznacza wartość błędu która następnie jest publikowana na dwóch tematach jako:

- `~error/raw/lateral` - wartość błędu
- `~error/norm/lateral` - znormalizowana wartość błędu do przedziału [-1.0, 1.0]

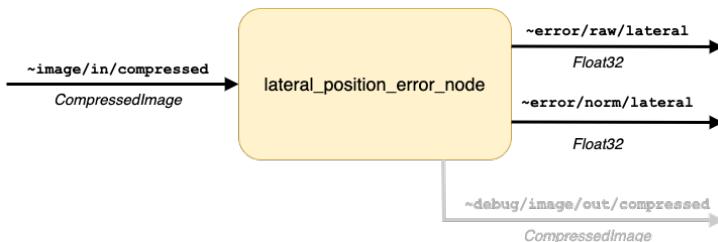
Kod węzła `lateral_position_error_node` który należy zaimplementować znajduje się w pliku:



Rysunek 1.14: Ekran programu `rqt` po załadowaniu konfiguracji.

`DTF/packages/sensing/lateral_position_error_node.py`

Rysunek 1.15 przedstawia węzeł `lateral_position_error_node` wraz z sygnałami wejściowymi i wyjściowymi.



Rysunek 1.15: Węzeł `lateral_position_error_node` pakietu `sensing`.

W celu zrealizowania zadania należy w pliku `lateral_position_error_node.py` w miejscach wskazanych komentarzem `# [A-Z] - Place your code here` (gdzie A-Z kolejne litery) wstawić swój kod:

- # A – Place... - "rozpakowanie" skompresowanego obrazu wejściowego z wykorzystaniem metody:

```
self.cvbridge.compressed_imgmsg_to_cv2
```

- # B Place... - konwersja rozpakowanego obrazu do przestrzeni barw HSV z wykorzystaniem metody:

```
cv2.cvtColor
```

pakietu OpenCV2.

- # C Place... – usunięcie z obrazu w przestrzeni kolorów HSV wszystkich elementów których kolor jest różny od zdefiniowanych za pomocą zmiennych

```
self.color_line_mask['lower1']
self.color_line_mask['upper1']
self.color_line_mask['lower2']
self.color_line_mask['upper2']
```

z wykorzystaniem metody:

```
cv2.inRange
```

pakietu OpenCV2. Wynik operacji dla przedziału kolorów 1 i 2 powinnien być zapisany w zmiennych lower_mask i upper_mask

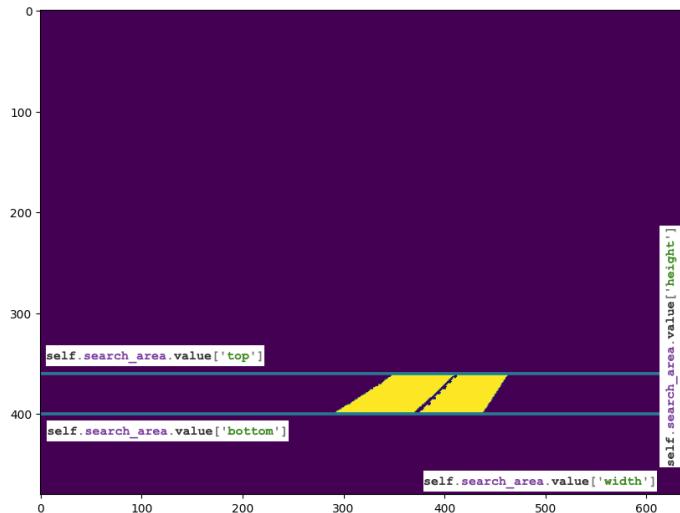
- # D Place... – usunięcie (poprzez wstawienie wartości 0 do każdego z kanałów obrazu) z obrazu wszystkich elementów które znajdują się poza zdefiniowanym obszarem za pomocą zmiennych:

```
self.search_area.value['top']
self.search_area.value['bottom']
self.image_param.value['height']
self.image_param.value['width']
```

Na rysunku 1.16 wskazano znaczenie zmiennych definiujących dopuszczalny obszar.

- # E Place... – wyznaczenie współrzędnych środka ciężkości otrzymanego obszaru z wykorzystaniem metody:

```
cv2.moments
```



Rysunek 1.16: Znaczenie zmiennych definiujących dopuszczalny obszar.

- `# F Place...` – wyznaczenie wartości błędu oraz znormalizowanej wartości błędu, zob. rys. 1.5b
- `# G Place...` – opublikowanie na odpowiednich tematach węzła wartości błędu i znormalizowanej wartości błędu z wykorzystaniem metody:

```
self.pub_error['raw'].publish
self.pub_error['norm'].publish
```

Spodziewane wyniki Prawidłowa implementacja węzła `lateral_position_error_node` powinna publikować na dwóch oddzielnych tematach wartość błędu i wartość błędu znormalizowaną. Wezel `lateral_position_error_node` można uruchomić wykonując instrukcję w kontenerze `duckietown/dtf:latest-amd64`

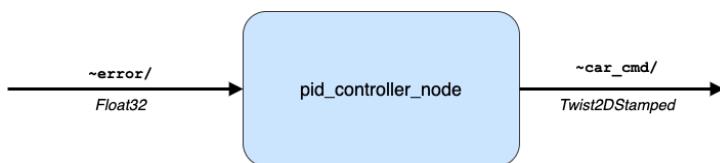
```
roslanuch packages/sensing/launch/
lateral_position_error_node.lanuch
```

Efekt działania można również zaobserwować w programie `rqt` subskrybując temat
`~debug/image/out/compressed`
który zawiera obraz oryginalny, przetworzony oraz wartości wyznaczonych błędów.

 Uwaga! Znak ~ w nazwie tematu oznacza, że końcowa nazwa tematu zależy od tego w jakiej przestrzeni nazw zostanie uruchomiony węzeł. Przestrzeń nazw w której uruchamiane są węzły jest definiowana w plikach *.launch.

6.5 Implementacja dyskretnego regulatora PID

Regulator PID do poprawnego działania wymaga informacji o tym w jakim położeniu w stosunku do linii wyznaczającej ścieżkę po której ma się poruszać robot znajduje się Duckiebot. Na rysunku 1.17 przedstawiono węzeł pid_controller_node wraz sygnałami wejściowymi i wyjściowymi.



Rysunek 1.17: Węzeł pid_controller_node pakietu controllers.

Węzeł ten powinien implementować dyskretny regulator PID dany równaniem (1.3). W celu zrealizowania zadania należy w pliku pid_controller_node.py w miejscach wskazanych komentarzem `# [A-Z] - Place your code here` (gdzie A-Z kolejne litery) wstawić swój kod:

- `# A - Place...` – implementacja dyskretnej wersji kontrolera PID danej równaniem (1.3),
- `# B - Place...` – wyliczenie wartości sterowania,
- `# C - Place...` – opublikowanie na temacie `~car_cmd` wiadomości ROSowej typu `Twist2DStamped` zawierające prędkość postępową v i kątową ω robota Duckiebot,
- `# D - Place...` – w pliku `packages/controllers/lanuch/pid_controller_node.launch` skonfigurować mapowanie tematów wejściowych dla regulatora

Spodziewane wyniki Poprawne działanie regulatora PID powinno skutkować publikowaniem dany (v, ω) na temacie `~car_cmd`. Do poprawnej pracy węzła `pid_controller_node` należy go uruchomić równocześnie z z węzłem `lateral_position_error_node` pakiety `sensing`. Można to zrobić wykonując instrukcje w utworzonym kontenerze `duckietown/dtf:latest-amd64`:

```
roslaunch package/startPID.launch
```

 **Uwaga!** Przed wykonaniem tej instrukcji należy się upewnić, czy robot Duckiebot znajduje się w bezpiecznej lokalizacji (najlepiej plansza po której ma jeździć).

Po uruchomieniu w drugim oknie terminala kontenera z narzędziami do monitorowania pracy robota Duckiebot, patrz [6.3](#), na ekranie programu `rqt` można zweryfikować poprawność pracy zaimplementowanego kodu. Oczywiście ostatecznym testem poprawności działania jest poruszenia się robota Duckiebot po planszy wzduł w wyznaczonej linii.

7 Sprawozdanie z realizacji ćwiczenia

7.1 Wymagania dotyczące sprawozdania

Sprawozdanie z ćwiczenia laboratoryjnego powinno zawierać następujące elementy:

- A. Informacje o zespole realizującym ćwiczenie;
- B. Sformułowanie problemu;
- C. Sposób rozwiązania problemu;
- D. Wyniki przeprowadzonych analiz, symulacji, testów i eksperymentów;
- E. Wnioski.

Do wykonania sprawozdania należy wykorzystać szablon, który jest umieszczony na uczelnianej platformie e-learningowej. Informacje związane z każdym z elementów sprawozdania A-E powinny znaleźć się na tylko na jednej stronie zgodnie z przygotowanym szablonem. Przebieg i rezultaty ćwiczenia należy przedstawić w sposób jednocześnie zwarty ale na tyle bogaty w informacje, aby osoba przeglądająca sprawozdanie

mogła na jego podstawie odtworzyć przebieg ćwiczenia. Każde sprawozdanie powinno łącznie zawierać pięć stron.

Sprawozdanie z ćwiczeń laboratoryjnych (jedno na grupę laboratoryjną) należy wysłać w postaci pliku PDF poprzez uczelnianą platformę e-learningową najpóźniej przed rozpoczęciem kolejnych zajęć. Plik ze sprawozdaniem należy nazwać zgodnie z poniższym schematem:

$$AP_LXX_RRRRMMDD_HHMM_GYY.pdf , \quad (1.4)$$

gdzie *AP* jest skrótem od nazwy przedmiotu Automatyki Pojazdowej, *LXX* oznacza numer ćwiczenia laboratoryjnego, np. *L01*, *L02* itd., *RRRRMMDD* jest datą wykonania ćwiczenia, np. *20190301* co oznacza, że ćwiczenie zostało wykonane 1 marca 2019 roku, *HHMM* jest czasem rozpoczęcia ćwiczenia laboratoryjnego, np. *1030* co oznacza, że ćwiczenia laboratoryjne rozpoczęły się o godzinie 10:30, *GYY* oznacza numer grupy laboratoryjnej, np. *G01*, *G02*, *G03* lub *G04* – numery grup nadaje prowadzący zajęcia.

7.2 Kryteria zaliczenia ćwiczeń

Tabela 1.1 przedstawia elementy składające się na kryterium zaliczenia ćwiczenia. W ramach każdego elementu kryterium można uzyskać 0, 1 lub 2 punkty. W sumie za w pełni poprawnie wykonane ćwiczenie laboratoryjne grupa (a tym samym każda osoba obecna i biorąca czynny udział w realizacji ćwiczenia) możetrzymać 10 punktów.

Przed rozpoczęciem każdego laboratorium odbędzie się wstępna weryfikacja niezbędnych umiejętności, które student powinien opanować, aby poprawnie i w zadanym czasie wykonać ćwiczenie. Test weryfikujący będzie dotyczył zagadnień wyszczególnionych w instrukcji do danego ćwiczenia. Weryfikacja ma formę 5 pytań (zob. tab. 1.2). Brak poprawnej odpowiedzi przez grupę na zadane pytanie oznacza 0 punktów, częściowo poprawna lub niepełna odpowiedź oznacza 1 punkt, pełna i poprawna odpowiedź to 2 punkty. Warunkiem koniecznym dopuszczenia grupy do laboratorium jest uzyskanie przez nią 5 punktów na 10 możliwych. W przypadku niedopuszczenia grupy do zajęć, grupa otrzymuje 0 punktów za dane ćwiczenie.

Pierwsze laboratorium nie będzie brane pod uwagę w kryteriach oceniania zarówno pod względem obecności na zajęciach jak i uzyskanych rezultatów z ćwiczenia.

Tabela 1.1: Elementy kryterium zaliczenia ćwiczenia oraz stosowana punktacja.

Element kryterium	Punktacja
Punkty z testu weryfikacyjnego	0 – w przypadku uzyskania przez grupę 5 lub 6 punktów w teście weryfikacyjnym; 1 – w przypadku uzyskania przez grupę 7 lub 8 punktów w teście weryfikacyjnym; 2 – w przypadku uzyskania przez grupę 9 lub 10 punktów w teście weryfikacyjnym
Pytanie sprawdzające nr 1	0 – w przypadku braku lub niepoprawnej odpowiedzi; 1 – w przypadku częściowo poprawnej lub niepełnej odpowiedzi; 2 – w przypadku pełnej i poprawnej odpowiedzi
Pytanie sprawdzające nr 2	0 – w przypadku braku lub niepoprawnej odpowiedzi; 1 – w przypadku częściowo poprawnej lub niepełnej odpowiedzi; 2 – w przypadku pełnej i poprawnej odpowiedzi
Pytanie sprawdzające nr 3	0 – w przypadku braku lub niepoprawnej odpowiedzi; 1 – w przypadku częściowo poprawnej lub niepełnej odpowiedzi; 2 – w przypadku pełnej i poprawnej odpowiedzi
Sprawozdanie z ćwiczenia	0 – brak sprawozdania w wyznaczonym terminie lub całkowicie błędne sprawozdanie pod względem redakcyjnym i merytorycznym; 1 – sprawozdanie jest częściowo poprawnie zredagowane lub zawiera niepełne wyniki; 2 – sprawozdanie jest poprawne pod względem redakcyjnym i zawiera poprawne wyniki

Bibliografia

- [1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.
- [2] “The duckiebot operation manual,” 2022. [Online]. Available: <https://docs.duckietown.org/en/stable/duckiebot/operation/manual.html>

Tabela 1.2: Elementy testu weryfikacyjnego oraz stosowana punktacja.

Element testu weryfikacyjnego	Punktacja
Pytanie sprawdzające nr 1	0, 1 lub 2
Pytanie sprawdzające nr 2	0, 1 lub 2
Pytanie sprawdzające nr 3	0, 1 lub 2
Pytanie sprawdzające nr 4	0, 1 lub 2
Pytanie sprawdzające nr 5	0, 1 lub 2

duckietown.com/daffy/opmanual-duckiebot/intro.html

- [3] K. J. Aström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.
- [4] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A survey of convolutional neural networks: analysis, applications, and prospects,” *IEEE transactions on neural networks and learning systems*, 2021.
- [5] W. Rawat and Z. Wang, “Deep convolutional neural networks for image classification: A comprehensive review,” *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.
- [6] P. Almási, R. Moni, and B. Gyires-Tóth, “Robust reinforcement learning-based autonomous driving agent for simulation and real world,” in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.
- [7] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System*. "O'Reilly Media, Inc.", 2015.
- [8] A. F. Daniele. Developer manual. [Online]. Available: <https://docs.duckietown.com/daffy/devmanual-software/intro.html>
- [9] “Duckietown developer manual,” 2022. [Online]. Available: <https://docs.duckietown.com/daffy/devmanual-software/intermediate/dtos/index.html>

Bibliografia

- [1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots.* MIT press, 2011.
- [2] “The duckiebot operation manual,” 2022. [Online]. Available: <https://docs.duckietown.com/daffy/opmanual-duckiebot/intro.html>
- [3] K. J. Aström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers.* Princeton university press, 2010.
- [4] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A survey of convolutional neural networks: analysis, applications, and prospects,” *IEEE transactions on neural networks and learning systems*, 2021.
- [5] W. Rawat and Z. Wang, “Deep convolutional neural networks for image classification: A comprehensive review,” *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.
- [6] P. Almási, R. Moni, and B. Gyires-Tóth, “Robust reinforcement learning-based autonomous driving agent for simulation and real world,” in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.
- [7] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System.* "O'Reilly Media, Inc.", 2015.
- [8] A. F. Daniele. Developer manual. [Online]. Available: <https://docs.duckietown.com/daffy/devmanual-software/intro.html>
- [9] “Duckietown developer manual,” 2022. [Online]. Available: <https://docs.duckietown.com/daffy/devmanual-software/intermediate/dtos/index.html>