Politechnika Śląska

Wydział Automatyki, Elektroniki i Informatyki

Raport z projektu PBL

Program Inicjatywa Doskonałości - Uczelnia Badawcza 2020

Pojazdy autonomiczne klasy UGV

Zespół projektowy:
Filip Hilgner - Informatyka, semestr 2 (S2)
Paweł Pietrzyk - Informatyka, semestr 2 (S2)
Mateusz Greń - Informatyka, semestr 2 (S2)
Sebastian Masłoń - Informatyka, semestr 2 (S2)

Opiekun główny: dr hab. inż. Adam Ziębiński, prof. PŚ
Pomocniczy opiekun naukowy: dr hab. inż. Rafał Cupek, prof. PŚ
Pomocniczy opiekun naukowy: mgr inż. Piotr Biernacki

Część prac w ramach projektu jest realizowana przez Międzywydziałowe Koło Naukowe Przemysłowych Zastosowań Informatyki „Industrum", którego członkami są wszystkie osoby z zespołu projektowego.

Gliwice, Luty 2021

Spis treści

Spis rysunków

Rozdział 1

Wstęp - Cele i założenia projektowe

Założeniem projektu PBL realizowanego przez naszą sekcję w semestrze zimowym 2020/2021 było opracowanie stanowiska laboratoryjnego jezdnej platformy mobilnej typu UGV z wykorzystaniem mikrokontrolerów Nvidia Nano, STM 32, sensorów pomiarowych oraz systemu Robot Operating System (ROS) z wykorzystaniem bibliotek ROS control, Navigation stack i Rviz. Celem projektu jest opracowanie stanowiska laboratoryjnego pozwalającego na badanie zależności pomiędzy architekturą sprzętową a metodyką implementacji wybranych zagadnień związanych z projektowaniem i realizacją pojazdów autonomicznych klasy UGV (Unmanned Ground Vehicle).

Projekt został podzielony na kilka etapów:

- Pierwszym z nich jest zapoznanie się ze sprzętem dostępnym na uczelni oraz stworzenie listy zakupów, w celu uzupełnienia braków.
- Jako kolejny krok przyjęliśmy demontaż platform, które zostały zaprojektowane i zbudowane przez studentów z poprzednich lat, a także posegregowanie poszczególnych elementów w taki sposób, aby praca nad budową platform przebiegała sprawnie.
- Trzecim etapem naszej pracy będzie zaprojektowanie sekcji zasilania dla naszej platform, a także przeprowadzenie testów, aby wszystkie elementy zostały odpowiednio zasilone.
- Czwartym krokiem jest rozplanowanie położenia poszczególnych komponentów oraz zamontowanie ich na platformie.
- Następnie stworzymy niezbędny software do komunikacji między komponentami.
- Przedostatnim etapem będzie przeprowadzenie testów sprawności naszej platformy.
- Finalnie powielimy sprawną platformę, na jej bliźniacze odpowiedniki.

Rozdział 2

Przebieg prac

Poniżej zostały przedstawione poszczególne etapy wykonywania prac związanych z realizowanym projektem, kamienie milowe przyjęte na początku realizacji projektu oraz oczekiwane wyniki realizacji omawianego projektu PBL.

2.1 Etapy realizacji projektu

2.1.1 Etap 1

W pierwszym tygodniu naszego projektu, odbyliśmy spotkanie z przedstawicielem grupy zajmującej się podobnym projektem, w celu zapoznania się z dostępnym sprzętem. Dowiedzieliśmy się, że nasza platforma będzie opierała się na procesorze STM 32 oraz NVIDI Nano. Wszystkie komponenty zamontujemy na platformie Pirate-4WD firmy DF Robot. W tym tygodniu stworzyliśmy też wstępną listę niezbędnych, do budowy, komponentów.

2.1.2 Etap 2

Drugi tydzień poświęciliśmy na przejrzenie dostępnych na uczelni komponentów i zweryfikowanie naszej listy. Wymieniliśmy się także naszymi spostrzeżeniami na temat podziału pracy oraz przemyśleniami dotyczącymi platformy.

2.1.3 Etap 3

W tym okresie spotkaliśmy się, aby zdemontować zbudowane przez studentów z ubiegłych lat platformy, w celu odzyskania potrzebnych części. Sprawdziliśmy, czy posiadane przez nas silniki są sprawne. Ponownie przejrzeliśmy naszą listę komponentów, abyśmy byli w stanie poczynić kolejne kroki w celu zamówienia brakujących elementów. W tym okresie pojawił się też pierwszy problem, którym było połączenie silników z enkoderami - zdecydowaliśmy się zatem na zaprojektowanie i wydrukowanie elementu łączącego na drukarce 3D.

2.1.4 Etap 4

W czwartym tygodniu z powodu epidemii zostaliśmy zmuszeni do pracy zdalnej. Pracowaliśmy, więc nad projektem tulei łączącej silniki z enkoderem.Także rozpoczęliśmy prace nad sprawozdaniem z projektu.

2.1.5 Etap 5

Tydzień rozpoczęliśmy od odebrania sprzętu znajdującego się na uczelni, aby móc w jakiś sposób pracować. Zaczęliśmy pracę nad łączeniem dostępnych komponentów ze sobą, a także kontynuowaliśmy pracę nad projektem tulei łączącej enkodery z silnikami.

2.1.6 Etap 6

Korzystając z możliwości przyjazdu na uczelnię, podjęliśmy próbę wydrukowania zaprojektowanej przez nas tulei na drukarce 3D.

2.1.7 Etap 7

Kolejnym etapem była próba zainstalowania posiadanego przez nas oprogramowania na Nvidię Nano, jak się okazało posiadaliśmy karty pamięci o zbyt małej pojemności, aby to zrobić. Więc została podjęta próba skompresowania oprogramowania obrazu w taki sposób, aby było możliwe zainstalowanie oprogramowania. Niestety mimo wielu prób, okazało się to niemożliwe i musieliśmy zakupić nowe karty pamięci o rozmiarze pamięci 32GB, zamiast 16GB.

2.1.8 Etap 8

Ten etap zajął nam około dwóch tygodni. W tym czasie udało nam się spotkać na uczelni i mogliśmy stacjonarnie pracować nad łączeniem posiadanych przez nas komponentów. W tym okresie złożyliśmy dwie platformy. Połączyliśmy ze sobą STM32, IMU, czujnik ultrasound, GPS, mini lidar oraz silniki. W trakcie tego etapu opracowaliśmy także sekcję zasilania.

2.1.9 Etap 9

Po złożeniu podstaw dwóch platform, powróciliśmy do prac nad sprawozdaniem, uzupełniliśmy opisy brakujących etapów, a także rozpoczęliśmy pracę nad schematem platformy. Również pojawił się problem związany z zasilaniem Nvidi Nano, gdyż przewody łączące STM32 przez przetwornicę nie dawały wystarczającego prądu aby Nvidia uruchomiła się. Również musieliśmy zmienić przetwornicę, gdyż maksymalny prąd wyjściowy był mniejszy niż minimalny prąd zasilający Nvidie.

2.1.10 Etap 10

Poza ciągłym rozwojem naszego sprawozdania, zainstalowaliśmy na kartach pamięci oprogramowanie do Nvidi Nano. Podłączyliśmy także kamerę do Nvidi i uruchomiliśmy przykładowe programy, aby sprawdzić poprawność działania. Pracowaliśmy także nad montażem poszczególnych elementów na platformie DF Robot Pirate-4WD.

2.1.11 Etap 11

Udało Nam się finalnie ukończyć sekcję zasilania. Musieliśmy wymienić kable zasilające oraz naszą przetwornicę. Gdy to uczyniliśmy Nvidia Nano bez problemu uruchomiła się oraz podłączona do niej kamera, a także RP Lidar zaczęły działać. Za pomocą środowiska ROS, RViZ wyświetliliśmy odczyt obrazu z kamery, a także odczyt danych w środowisku RViZ z RP Lidaru.

2.1.12 Etap 12

Kolejny etap naszego projektu poświęciliśmy na dokończenie sprawozdania, przeprowadzenie testów platformy oraz dokonanie drobnych poprawek w części sprzętowej tworzonej platformy.

2.1.13 Etap 13

Ostatni etap naszego projektu skupił się w głównej mierze na poprawkach w części programowej projektu, zbieraniu danych z czujników, testowaniu platformy oraz finalizacji sprawozdania.

2.2 Oczekiwane wyniki

- Opracowanie systemu kontrolno-pomiarowego umożliwiającego równoczesną akwizycje danych z podłączonych sensorów oraz sterowanie platformą UGV.
- Opracowanie metody komunikacji stacji nadzorczej z modułami pomiarowymi i platformą UGV dla potrzeb monitoringu, wizualizacji i analizy danych

2.3 Kamienie milowe

- Opracowanie stanowiska pozwalającego na akwizycję danych z wybranych sensorów przez platformę UGV.
- Opracowanie algorytmów sterowania platformą UGV.
- Opracowanie metody komunikacji stacji nadzorczej z platformą UGV i modułami pomiarowymi.
- Opracowanie architektury systemu monitoringu, wizualizacji i analizy danych z platformy UGV.

Rozdział 3

Hardware

W tym rozdziale przedstawiliśmy kwestie związane ze stroną sprzętową naszego projektu. Została tutaj przedstawiona lista elementów składowych naszej platformy jezdnej, schematy połączeń oraz poszczególne komponenty występujące w naszej platformie wraz z opisem.

Główne elementy wchodzące w skład naszej platformy to mikrokontroler STM 32 F4 oraz Nvidia Nano. Również w skład komponentów wchodzi szereg czujników, które szczegółowo zostały opisane w tym rozdziale.

3.1 Lista elementów

Tabela 3.1: Zestawienie elementów potrzebnych na jedną platformę.

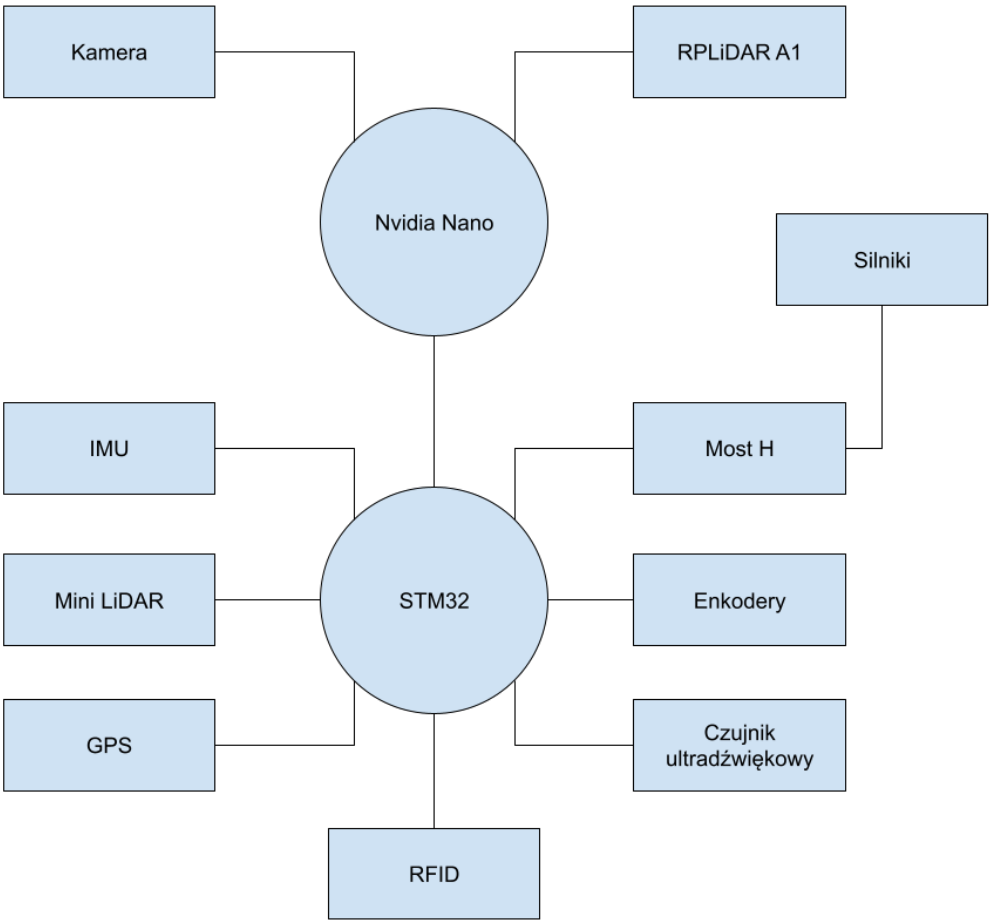
Nr	Nazwa	Ilość sztuk
1	Nucleo STM32F429ZI	1
2	Most H X-Nucleo IHM04A1	1
3	Płyta IMU X-Nucleo IKS01A3	1
4	Moduł GPS	1
5	DFRobot MiniLidar	1
6	Czujnik ultradźwiękowy URM37 (z UART)	1
7	Moduł RFID	1
8	Enkoder	2
9	Akumulator LiPo 7V 2.2Ah	2
10	Powerbank 5V 8Ah	1
11	Ładowarka LiPo	1
12	NVIDIA NANO	1
13	Kamera do Nvidii	1
14	RPLidar A1	1
15	Karta pamięci dla Nvidia Nano 32GB	1
16	Zasilacz 5V 4A dla Nvidia Nano	1
17	Zestaw DFRobot Pirate-4WD	1
18	Kabel Ethernet - crossowany	1
19	Kable jumper F-F	30
20	Kable jumper F-M	30
21	Kable jumper M-M	30
22	Grubsze przewody zasilające (dł. 10 cm)	10

Ciąg dalszy na kolejnej stronie

Tabela 3.1 – kontynuacja z poprzedniej strony

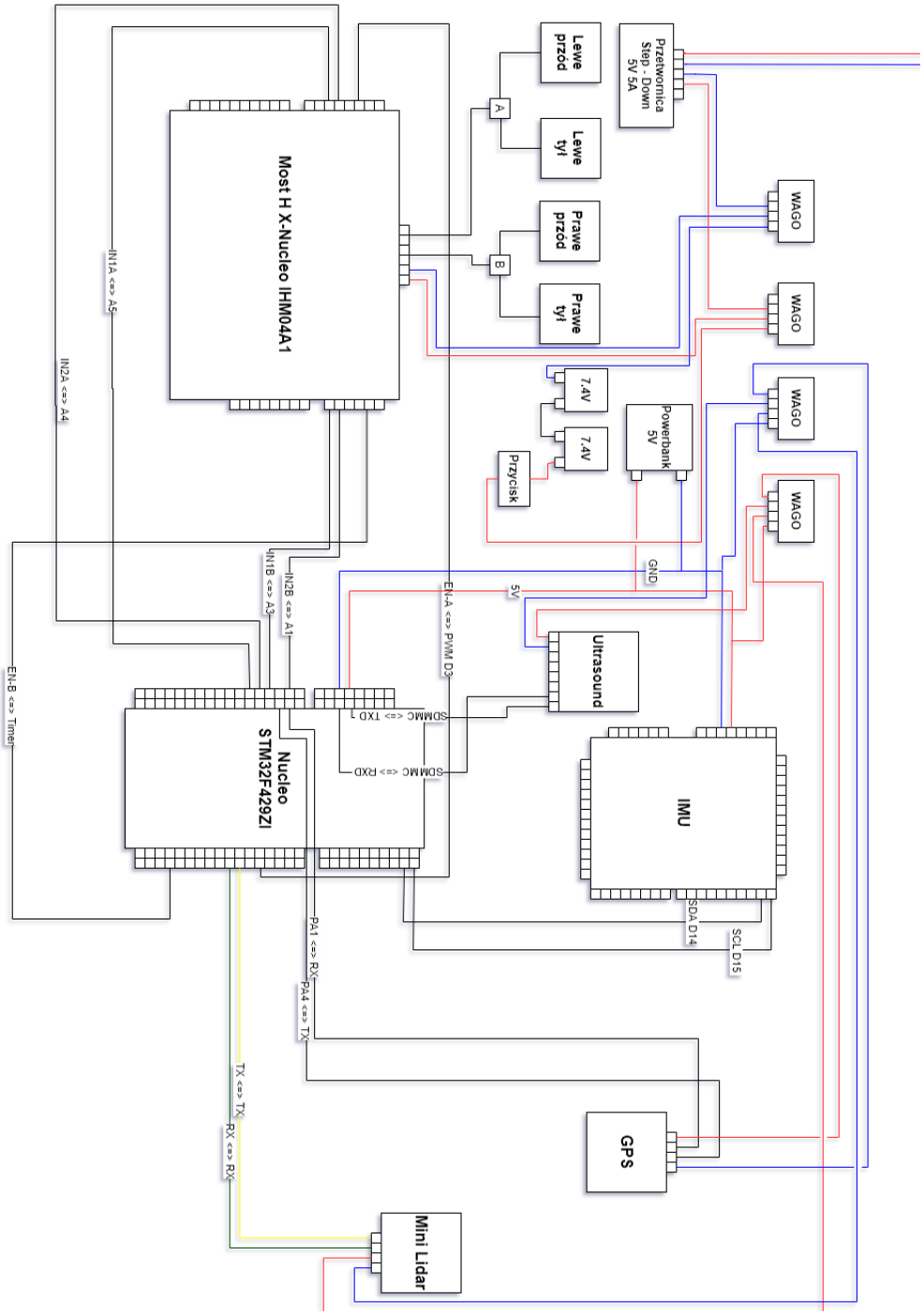
Nr	Nazwa	Ilość sztuk
23	Kabel USB - microUSB	1
24	Goldpiny	20
25	Płyta uniwersalna PCB	1
26	Złączki T-Dean	4
27	Złączki Wago	6
28	Przetwornica step-down - 5V 5A - Pololu 2851	1
29	Przełącznik on-off	1
30	Dystanse montażowe	10

3.2 Schemat ogólny

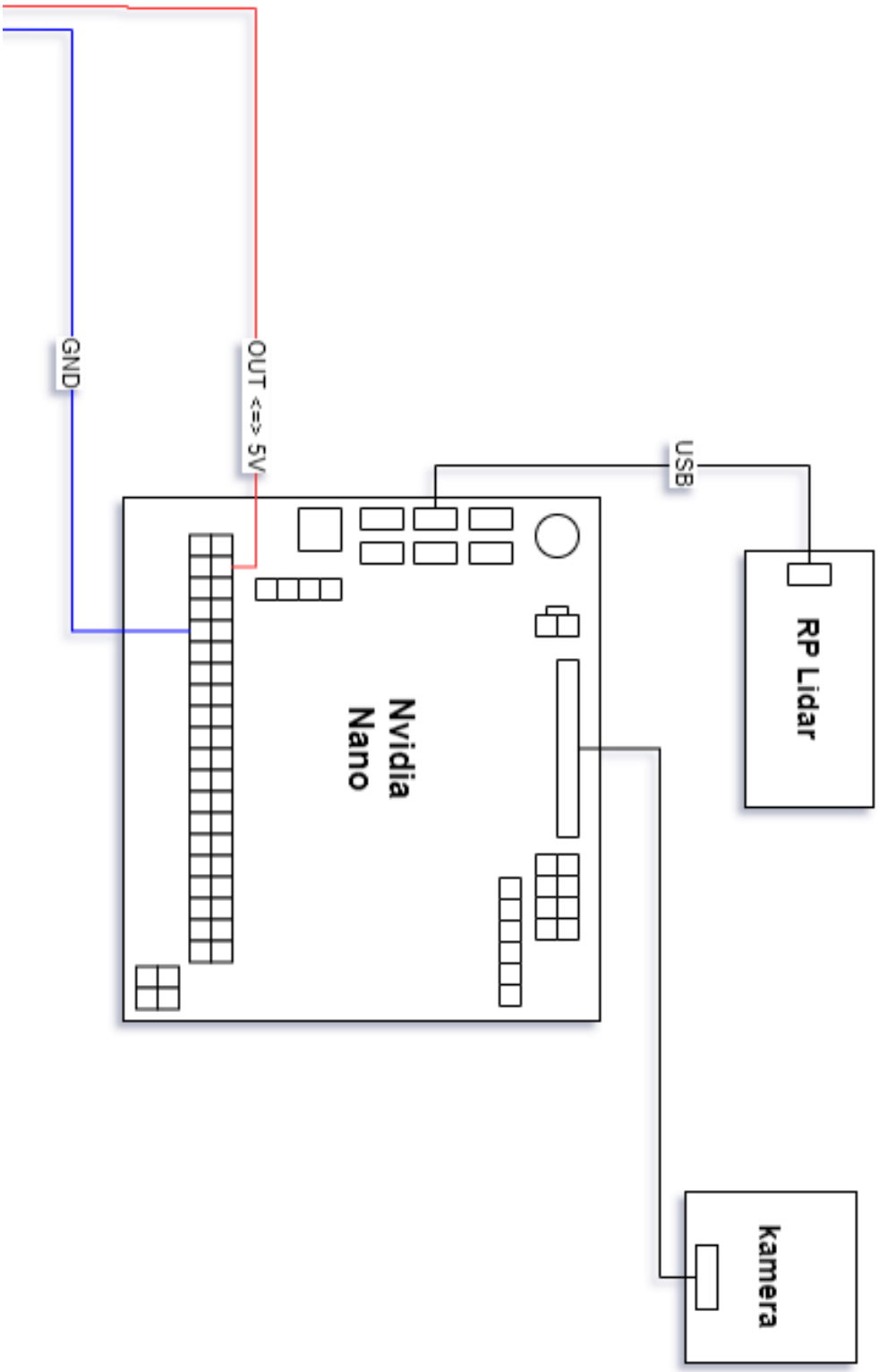


Rysunek 3.1: Schemat ogólny połączeń sygnałowych

3.3 Schemat połączeń



Rysunek 3.2: Schemat połączeń układu platformy - część 1



Rysunek 3.3: Schemat połączeń układu platformy - część 2

3.4 Platforma DFRobot Pirate-4WD

Jest to platforma z napędem na cztery koła. W zestawie również są dostępne 4 silniki prądu stałego. Platforma posiada aluminiowe płytki umożliwiające rozbudowę platformy o dodatkowe elementy oraz zamocowanie ich. W naszym projekcie wymontowaliśmy z platformy oryginalne enkodery i podjęliśmy próbę zamontowania dokładniejszych enkoderów, jednakże dedykowanych dla płytek stykowych.

- Napięcie zasilania: od 4,5 V do 6 V
- 4 silniki:
  - Zasilanie: od 3 V do 7 V
  - Pobór prądu bez obciążenia (3 V): 140 mA
  - Pobór prądu bez obciążenia (6 V): 170 mA
  - Prędkość bez obciążenia (3 V): 90 obr/min
  - Prędkość bez obciążenia (6 V) 160 obr/min
  - Maksymalny moment obrotowy: 0,8 kg*cm (0,07 Nm)
- Średnica kół: 65 mm
- Wymiary: 230 x 185 x 110 mm
- Masa: 614 g
- Maksymalne obciążenie: 800 g

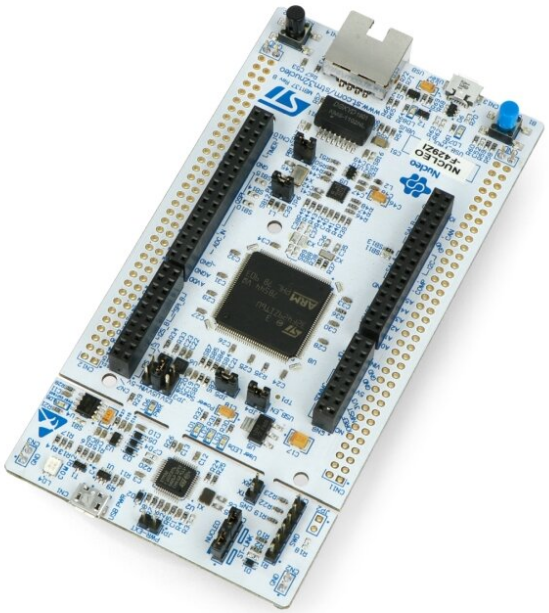
Rysunek 3.4: Zdjęcie poglądowe platformy DFRobot 4WD - bootland.com

3.5 Nucleo STM32F4

Jest to moduł z 32-bitowym mikrokontrolerem. Zestaw wyposażony został w rdzeń ARM Cortex M4. Moduł ten jest przez nas używany w celu zapewnienia komunikacji pomiędzy mostkiem H - sterującym silnikami, czujnikami, a Nvidią Nano.

- Mikrokontroler: STM32F429ZIT6
  - Rdzeń: ARM Cortex M4 32-bit
  - Częstotliwość taktowania: 180 MHz
  - Pamięć programu Flash: 2 MB
  - Pamięć RAM: 256 kB
  - Przetwornik analogowo-cyfrowy (ADC): 12-bitowy, 24-kanałowy
  - Przetwornik cyfrowo-analogowy (DAC): 12-bitowy, 2-kanałowy
  - Ilość Timerów 16-bit: 12
  - Ilość Timerów 32-bit: 2
  - Interfejsy: 3x I2C, 6x SPI, 2x I2S, 2x CAN, 4x USART, 4x UART, USB OTG
  - Złącze Ethernet
- Dwa typy złącz:
  - Złącza dla nakładek kompatybilnych z Arduino Uno Rev3
  - Standardowe piny STMicroelectronics Morpho, umożliwiające dostęp do wyprowadzeń mikrokontrolera
- Moduł zgodny z systemem mbed (mbed.org)
- Debugger ST-Link/V2 umieszczony na płytce z możliwością pracy jako oddzielne urządzenie z wyjściem SWD
- Możliwość zasilania poprzez złącze USB
- Wbudowane trzy diody LED:
  - 1 x sygnalizująca napięcia zasilania
  - 1 x sygnalizująca komunikację
  - 1 x do dyspozycji użytkownika
- Dwa przyciski:
  - 1 x RESET
  - 1 x do dyspozycji użytkownika

- Trzy różne interfejsy poprzez złącze microUSB (USB re-enumeration):
  - Wirtualny port COM
  - Pamięć masowa
  - Port do programowania/debuggowania
- Moduł wspierany przez większość popularnych środowisk, m.in: IAR, Keil oraz platformy oparte na kompilatorze GCC
- Wymiary modułu: 135 x 70 x 18 mm



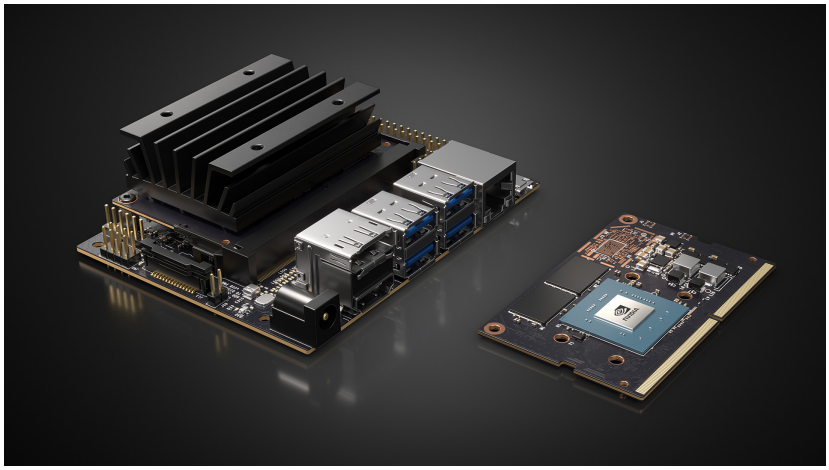
Rysunek 3.5: Zdjęcie poglądowe Nucleo STM32F4 - bootland.com

3.6 Nvidia Nano

Nvidia Jetson Nano pozwala na przetwarzanie sygnału dostarczanego przez kamery oraz inne systemy wizyjne. Dzięki temu jesteśmy w stanie odciążyć STM i uzyskujemy lepszą wydajność naszego systemu.

- GPU: 128-rdzeniowy procesor graficzny oparty na architekturze NVIDIA Maxwell™

- Procesor: czterordzeniowy ARM® A57
- Wideo: kodowanie i dekodowanie 4K przy 30 fps (H.264 / H.265) / 4K przy 60 fps (H.264 / H.265)
- Kamera: ścieżki MIPI CSI-2 DPHY, 12x (moduł) i 1x (zestaw deweloperski)
- Pamięć: 4 GB 64-bit LPDDR4; 25,6 gigabajtów / sekundę
- Łączność: Gigabit Ethernet
- Obsługa systemu operacyjnego: Linux dla Tegra®
- Rozmiar modułu: 70 mm x 45 mm
- Rozmiar zestawu deweloperskiego: 100 mm x 80 mm



Rysunek 3.6: Zdjęcie poglądowe Nvidia Jetson Nano - nvidianews.nvidia.com

3.7 Czujniki

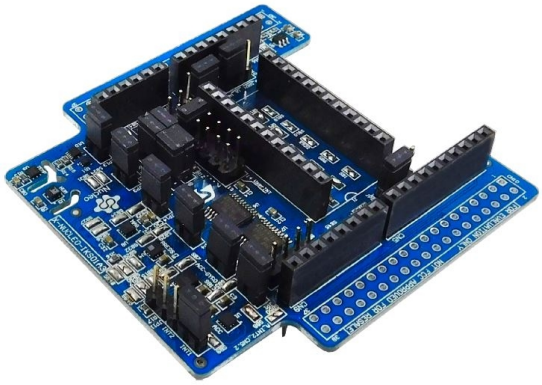
Poniżej przedstawiliśmy poszczególne czujniki, które wykorzystaliśmy w projekcie wraz z ich krótkim opisem.

3.7.1 Płyta IMU X-Nucleo

Jest to nakładka do płytki STM32. Posiada wbudowane czujniki, takie jak: akcelerometr, żyroskop, magnetometr, barometr, a także czujnik temperatury i wilgotności.

- LSM6DS0:
  - 3-osiowy akcelerometr o zakresie ±2 / ±4 / ±8 g

  - 3-osiowy żyroskop o zakresie ±245 / ±500 / ±2000 dps
- LIS3MDL: 3-osiowy magnetometr o zakresie ±4 / ±8 / ±12 / ±16 gauss
- LPS25HB: czujnik ciśnienia, barometr o zakresie od 260 hPa do 1260 hPa
- HTS221: czujnik ciśnienia i temperatury
- 24-pinowe złącze do podłączenia innych modułów MEMS
- Kompatybilny z płytkami STM32 Nucleo
- Złącze Arduino do podłączenia shieldów do Arduino
- Wymiary: 70 x 70 x 12 mm



Rysunek 3.7: Zdjęcie poglądowe IMU X-Nucleo - botland.com

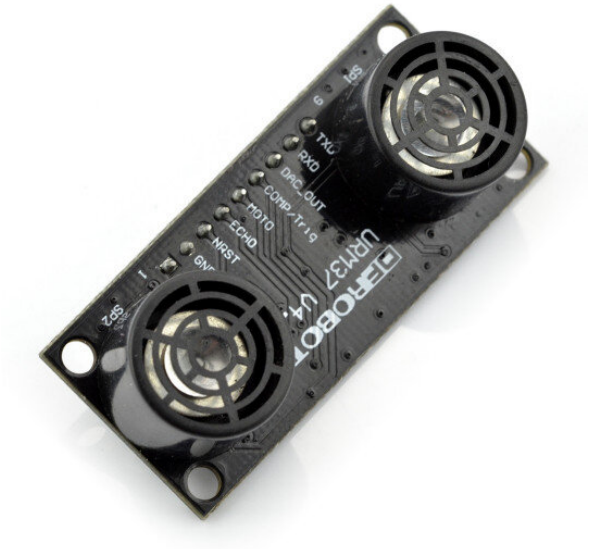
3.7.2 Czujnik ultradźwiękowy

Czujnik ultradźwiękowy URM37 został przez nas użyty do wykrywania potencjalnych przeszkód, znajdujących się na drodze naszego robota.

- Napięcie zasilania: od 3,3 V do 5,0 V
- Średni pobór prądu: poniżej 20 mA
- Zakres pomiarowy: od 5 cm do 500 cm
- Rozdzielczość: 1 cm
- Interfejs komunikacyjny:
  - RS232 (TTL - piny RX,TX)
  - PWM - od 0 do 25 000 μs, ze zmianą 1 cm / 50 μS

  - Wyjście analogowe - sygnał proporcjonalny do odległości 6,8 mV na 1 cm

- Czas pomiaru: 100 ms
- Możliwość kontrolowania serwomechanizmu modelarskiego
- Zabezpieczenie przed odwrotnym podłączeniem napięcia zasilania
- Kompensacja wpływu temperatury na wartość mierzoną
- Temperatury pracy: od 10°C do 70°C
- Wymiary: 51 x 22 mm
- Masa: 25 g



Rysunek 3.8: Zdjęcie poglądowe ultrasound URM37 - botland.com

3.7.3 Moduł GPS

Moduł GPS pozwala nam na określenie pozycji naszej platformy. W naszym projekcie skorzystaliśmy zs moduły GPS GY-GPS6MV2.

- Układ GPS: U-blox NEO-6M
- Typ transmisji: UART (linie Rx, Tx)
- Pasmo odbierane: L1 (1575.42MHz)
- Skuteczność pozycjonowania: średnio 3 m

- Dryft: < 0.02 m/s
- System odniesienia: WGS-84
- Maksymalna wysokość: 18,000 m
- Maksymalna prędkość: 515 m/s
- Przyspieszenie: $< 4g$
- Czułość śledzenia: -161 dBm
- Czułość akwizycji: -148 dBm
- Średni czas "zimnego startu": 38 s
- Średni czas "ciepłego startu" 35 s
- Czas gorącego startu: 1 s
- Średni czas przywracania: 0,1 s
- Zakres temperatur pracy: od -40°C do +80°C
- Zakres napięcia zasilającego: od +3,5 V do +5,5 V
- Napięcie magistrali UART: 3,3 V
- Punkty lutownicze pozwalają na montaż anteny na PCB
- Wbudowana bateria podtrzymująca RTC
- Wbudowane złącze u.FL do anteny
- Wymiary modułu (bez anteny): 36 x 26 x 4 mm
- Wymiary anteny: 25 x 25 x 8 mm

Rysunek 3.9: Zdjęcie poglądowe GPS GY-GPSMV2 - kamami.pl

3.7.4 Mini Lidar

MiniLiDAR TF firmy DFRobot został przez nas użyty w celu wspierania pracy czujnika ultradźwiękowego. Ma on za zadanie mierzenie odległości od potencjalnych przeszkód znajdujących się na drodze naszej platformy.

- Zakres roboczy: 0,3-12 m
- Maksymalny zakres współczynnika odbicia 10
- Średnia moc: 0,6 W
- Kąt akceptacji: 2,3°
- Minimalna rozdzielczość: 5 mm
- Częstotliwość odświeżania: 100 Hz
- Dokładność pomiaru: 1% (<6 m), 2
- Jednostka zakresu: mm
- Długość fali: 850 nm
- Interfejs komunikacyjny: UART (TTL)
- Odporność na światło do 70 klux
- Zakres napięcia: 4,5-6 V

- Pobór średni prądu: 120 mA
- Rozmiar: 42 x 15 x 16 mm
- Temperatura pracy: 0-60 ° C
- Waga: 4,7 g

Rysunek 3.10: Zdjęcie poglądowe Mini LiDAR DFRobot - kamami.pl

3.7.5 RP Lidar A1

Skaner laserowy, wykorzystywany do skanowania pomieszczenia i wykrywania potencjalnych przeszkód na drodze platformy jezdnej.

- Zasięg: 0,15-12 metrów
- Widoczność pola pomiarowego: 360 stopni
- Rozdzielczość: 0,5-1% (min 0,5 mm) ¡ 1,5 m
- Częstotliwość pomiaru: do 10 Hz (zalecane 5,5 Hz)
- Interfejs: UART (napięcie logiczne 3,3 V, domyślna prędkość 115200)
- Współpraca z SLAMAIg
- Wymiary: 70 x 98,5 x 60 mm
- Waga: około 190 g

Rysunek 3.11: Zdjęcie poglądowe RP Lidar A1 - kamami.pl

3.7.6 Czujnik RFID

Czujnik RFID, który w przyszłości będzie przez nas używany do sczytywania tagów z kart rozstawionych na drodze platformy jezdnej.

- Napięcie zasilania: 2,8 V - 5 V
- Częstotliwość nośna: 125 kHz
- Standard: EM4001 ISO RFID IC
- Komunikacja: interfejs szeregowy TTL i RS232 - 9600 bps
- Raster wyprowadzeń: 2 m
- Wymiary: 21 x 19 mm

Rysunek 3.12: Zdjęcie poglądowe Czujnik RFID - kamami.pl

3.7.7 Enkodery

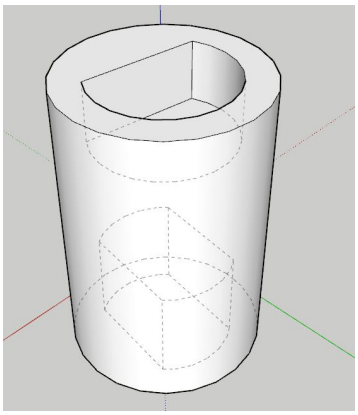
W naszej platformie zamontowaliśmy na przedniej osi dwa enkodery (jeden na prawe przednie koło, drugi na lewe), w celu mierzenie obrotu koła platformy w danej jednostce czasu. W naszym przypadku posłużyliśmy się enkoderami 24-impulsowymi dedykowanymi do płytek stykowych. Enkoder posiada 2 wyprowadzenia montażowe, 2 wyprowadzenia dotyczące wbudowanego przycisku oraz 3 wyprowadzenia związane z odczytem z enkodera (faza "A", wspólne masa oraz faza "B"). Moduł wysyła sygnał cyfrowy na dwa wyprowadzenia (faza "A" oraz "B") w zależności od kierunku obrotu.

- Wykrywa obrót zgodnie ze wskazówkami zegara oraz przeciwnie
- Ilość kroków w 360°: 24 kroki
- Wbudowany przycisk w pokrętle
- Wymiary modułu: 21 x 13 x 12 mm

Rysunek 3.13: Zdjęcie poglądowe enkodera - botland.pl

Również w celu połączenia enkodera z wałem silnika musieliśmy stworzyć model łączący obydwa elementy. W tym celu posłużyliśmy się programem SketchUp, w którym

stworzyliśmy model tulei łączącej wał silnika z elementem obrotowym enkodera. Model w środowisku SketchUp widoczny poniżej.



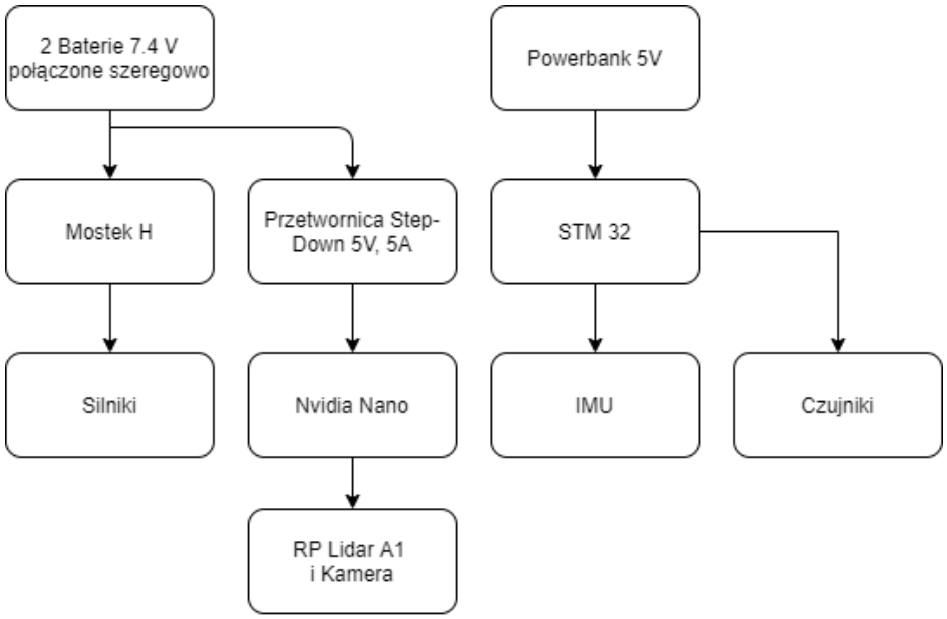
Rysunek 3.14: Wykonany model tulei w środowisku SketchUp

Kolejnym elementem tworzenia opisywanej tulei było wydrukowanie jej na drukarce 3D. Rezultat uzyskany po wydrukowaniu widoczny poniżej.

Rysunek 3.15: Wydrukowana tuleje na drukarce 3D

3.8 Zasilanie

W poniższym podrozdziale przedstawimy schemat zasilania oraz dwa źródła zasilania naszej platformy, z których aktualnie korzystamy.



Rysunek 3.16: Poglądowy diagram zasilania

3.8.1 Powerbank

W naszym projekcie skorzystaliśmy z powerbanków firmy Blow o pojemności 8Ah. Za jego pomocą zasilamy STM32 oraz czujniki.

- Kolor: Czarne
- Materiał obudowy: Tworzywo sztuczne
- Pojemność: 8000mAh
- Natężenie wejściowe: DC 5V/2A
- Natężenie wyjściowe: DC 5V/1A/2,4(MAX)
- Producent: Blow
- Rodzaj baterii: Litowo-polimerowa

Rysunek 3.17: Zdjęcie poglądowe Powerbank Blow - ceneo.pl

3.8.2 Akumulator

Akumulator, którego użyliśmy w naszym projekcie ma pojemność 2500mAh. Na naszej platformie zainstalowaliśmy dwa takie akumulatory połączone za sobą w taki sposób, aby dawały wyjściowe napięcie 14V. Za ich pomocą zasilamy most H oraz silniki.

- Li-pol - 2S - dwa ogniwa
- Producent: GPX Extreme
- Napięcie nominalne: 7,4 V
- Pojemność: 2500 mAh
- Prąd rozładowania: do 25 C (62,5 A)
- Prąd ładowania rekomendowany: od 1 do 3 C (od 2,5 A do 7,5 A)
- maksymalny prąd ładowania: 5 C (12,5 A)
- Wyjście balansera: JST-XH
- Konektor: gniazdo T-DEANS
- Wymiary: 134 x 43 x 14 mm
- Masa: 176 g

Rysunek 3.18: Zdjęcie poglądowe akumulator GPX EXTREME - botland.com

Rozdział 4

Software

W tym rozdziale przedstawiliśmy kwestie związane z oprogramowaniem wykorzystanym na naszym stanowisku badawczym. Część programową podzieliliśmy na dwa podrozdziały dotyczące każdego z mikrokontrolerów - STM 32 F4 i odpowiadającemu mu środowisku STM 32 IDE CUBE oraz Nvidia Nano, na której w głównej mierze korzystam z systemu ROS (Robot Operating System).

4.1 STM32

W środowisku STM32 CUBE IDE został zrealizowany projekt, który następnie wgraliśmy na nasz mikrokontroler STM32F4. Wszystkie dane przesyłane są za pomocą ramek.

4.1.1 Upstream (dane z STM32F4 do STalkera)

Tabela 4.1: Dane dotyczące monitorowania sterowania pojazdem.

ID	0x02
Liczba zestawów	0x01
Kierunek silników(DC - DC)	0x1 - 0x1
Lewy DC PWM (H)	0x00
Lewy DC PWM (L)	0xF0
Prawy DC PWM (H)	0x00
Prawy DC PWM (L)	0x50
Pozostały czas (H)	0x03
Pozostały czas (L)	0xC5
Liczba rozkazów kolejce	0x05

Tabela 4.2: Kierunki jazdy silników DC.

Wartość (4 bity)	**Znaczenie**
0x0	Stop
0x1	Jazda w przód
0x2	Jazda w tył

Tabela 4.3: Dane z czujników inercyjnych.

ID
Liczba zestawów
Oś X (H)
Oś X (L)
Oś Y (H)
Oś Y (L)
Oś Z (H)
Oś Z (L)
Znacznik czasu (Bajt 3)
Znacznik czasu (Bajt 2)
Znacznik czasu (Bajt 1)
Znacznik czasu (Bajt 0)

Tabela 4.4: Znaczenie ID dla czujników inercyjnych .

Nazwa czujnika	Akcelerometr	Żyroskop	Magnetometr
ID	0x04	0x05	0x06

Tabela 4.5: Dane z enkodera.

ID	0x07
Liczba zestawów	0x01
Kierunek silników(DC - DC)	0x01 - 0x01
Prędkość lewego enkodera [cm/s] (Bajt 3)	
Prędkość lewego enkodera [cm/s] (Bajt 2)	
Prędkość lewego enkodera [cm/s] (Bajt 1)	
Prędkość lewego enkodera [cm/s] (Bajt 0)	
Pokonany dyst. Lew. Enk. [cm](Bajt 3)	
Pokonany dyst. Lew. Enk. [cm](Bajt 2)	
Pokonany dyst. Lew. Enk. [cm](Bajt 1)	
Pokonany dyst. Lew. Enk. [cm](Bajt 0)	
Prędkość prawego enkodera [cm/s] (Bajt 3)	
Prędkość prawego enkodera [cm/s] (Bajt 2)	
Prędkość prawego enkodera [cm/s] (Bajt 1)	
Prędkość prawego enkodera [cm/s] (Bajt 0)	
Pokonany dyst. Praw. Enk. [cm](Bajt 3)	
Pokonany dyst. Praw. Enk. [cm](Bajt 2)	
Pokonany dyst. Praw. Enk. [cm](Bajt 1)	
Pokonany dyst. Praw. Enk. [cm](Bajt 0)	

Dane z GPS:

- ID - 0x08 (8 bitów)
- Liczba zestawów - 0x01(8 bitów)
- Liczba satelitów (8 bitów)
- Wartość HDOP (8 bitów)
- Szerokość geograficzna (64 bity) - typ double
- Długość geograficzna (64 bity) - typ double
- Czas od pozyskania danych przez GPS (32 bity)
- Dzień miesiąca (8 bitów)
- Miesiąc (8 bitów)
- Rok (16 bitów)
- Czas - godzina (8 bitów)
- Czas - minuta (8 bitów)
- Czas - sekunda (8 bitów)
- Wysokość [m] n.p.m (32 bity) - typ float
- Azymut [deg] (32 bity) - typ float
- Prędkość [km/h] (32 bity) - typ float

Tabela 4.6: Dane z Mini LiDARu.

ID	0x09
Liczba zestawów	0x01
Dystans [cm] (H)	
Dystans [cm] (L)	
Siła sygnału (H)	
Siła sygnału (L)	
Tryb lidaru	

Tryb Mini LiDARu:

- 0x00 - dystans od 0m do 0.3m
- 0x03 - dystans od 0.3m do 6m
- 0x07 - dystans od 6m do 12m

Tabela 4.7: Dane na temat konfiguracji timerów.

ID	0x03
Liczba zestawów	0x01
Rejestr PSC IMU (H)	
Rejestr PSC IMU (L)	
Rejestr ARR IMU (H)	
Rejestr ARR IMU (L)	
Częstotliwość IMU [Hz] (Bajt 3)	
Częstotliwość IMU [Hz] (Bajt 2)	
Częstotliwość IMU [Hz] (Bajt 1)	
Częstotliwość IMU [Hz] (Bajt 0)	
Rejestr PSC Data send (H)	
Rejestr PSC Data send (L)	
Rejestr ARR Data send (H)	
Rejestr ARR Data send (L)	
Rej. Clock Div. Data send	
Częstotliwość Data send [Hz] (Bajt 3)	
Częstotliwość Data send [Hz] (Bajt 2)	
Częstotliwość Data send [Hz] (Bajt 1)	
Częstotliwość Data send [Hz] (Bajt 0)	

Tabela 4.8: Dane z czujnika ultradźwiękowego.

ID	**Liczba zestawów**	**Dystans[cm] (H)**	**Dystans[cm] (L)**
0x0F	0x01		

Tabela 4.9: Dane z czujnika RFID.

ID	0x0E
Liczba zestawów	
ID karty (Bajt 4)	
ID karty (Bajt 3)	
ID karty (Bajt 2)	
ID karty (Bajt 1)	
ID karty (Bajt 0)	

Tabela 4.10: Dane na temat czasu obsługi poszczególnych operacji.

ID	0x11
Liczba zestawów	0x01
Czas IMU (H)	
Czas IMU (L)	
Czas Data send (H)	
Czas Data send (L)	
Czas Enkoder (H)	
Czas Enkoder (L)	
Czas PWM (H)	
Czas PWM (L)	
Czas LiDAR Continental (H)	
Czas LiDAR Continental (L)	
Czas Czujnik ultradźwiękowy (H)	
Czas Czujnik ultradźwiękowy (L)	

Tabela 4.11: Dane na temat pakietów.

ID	0xFE
Liczba zestawów	0x01
Liczba przesłanych pakietów (Bajt 3)	
Liczba przesłanych pakietów (Bajt 2)	
Liczba przesłanych pakietów (Bajt 1)	
Liczba przesłanych pakietów (Bajt 0)	
Liczba błędnych pakietów (Bajt 3)	
Liczba błędnych pakietów (Bajt 2)	
Liczba błędnych pakietów (Bajt 1)	
Liczba błędnych pakietów (Bajt 0)	

Tabela 4.12: Dane na temat pakietów.

ID	0xFF
Liczba zestawów	0x01
Znacznik czasu (Bajt 3)	
Znacznik czasu (Bajt 2)	
Znacznik czasu (Bajt 1)	
Znacznik czasu (Bajt 0)	

4.1.2 Downstream (dane z STalkera do STM32F4)

Tabela 4.13: Rozkaz dotyczący sterowania pojazdem.

ID	0x02
Liczba zestawów	0x01
Kierunek silników(DC - DC)	0x1 - 0x1
Lewy DC PWM (H)	0x00
Lewy DC PWM (H)	0xF0
Prawy DC PWM (H)	0x00
Prawy DC PWM (L)	0x50
Czas rozkazu (H)	0x03
Czas rozkazu (L)	0xC5
Czy kolejkować	0x01

Tabela 4.14: Kierunki jazdy silników DC.

Wartość (4 bity)	**Znaczenie**
0x0	Stop
0x1	Jazda w przód
0x2	Jazda w tył

4.2 ROS

Rozdział ten dotyczy oprogramowania ROS. Od przygotowania obrazu do uruchomienia na Nvidii Nano, po praktyczne wykorzystanie dostępnych w nim funkcji.

4.2.1 Przygotowanie obrazu

W celu przygotowania obrazu z systemem na Nvidię Nano , należało zmniejszyć jego rozmiar do rozmiaru karty pamięci. Najprostszym sposobem okrojenia obrazu jest użycie skryptu Pishrink dostępnego pod adresem: `https://github.com/Drewsif/`

`PiShrink` . Aby zainstalować skrypt na komputerze z systemem linux należy wykonać ciąg polecń:

```
$ wget https://raw.githubusercontent.com/Drewsif/PiShrink/master/
   pishrink.sh
$ chmod +x pishrink.sh
$ sudo mv pishrink.sh /usr/local/bin
```

Następnie należy dokonać zmniejszenia obrazu za pomocą polecenia:

```
$ sudo pishrink.sh obraz.img
```

W tym przypadku obraz nosi nazwę "obraz.img". Jeśli zmniejszenie obrazu się powiodło naszym oczom powinno się ukazać następujący widok:

```
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/loop1: 88262/1929536 files (0.2% non-contiguous),
   842728/7717632 blocks
resize2fs 1.42.9 (28-Dec-2013)
resize2fs 1.42.9 (28-Dec-2013)
Resizing the filesystem on /dev/loop1 to 773603 (4k) blocks.
Begin pass 2 (max = 100387)
Relocating blocks XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Begin pass 3 (max = 236)
Scanning inode table XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Begin pass 4 (max = 7348)
Updating inode references XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
The filesystem on /dev/loop1 is now 773603 blocks long.

Shrunk pi.img from 30G to 3.1G
```

Niestety skrypt ten nie działa za każdym razem. W naszym przypadku skrypt nie zadziałał poprawnie i plik po zmniejszeniu nie nadawał się do użytku, dlatego należało zmniejszyć obraz za pomocą innej metody. Pierwszą czynnością było zamontowanie obrazu, aby tego dokonać należy wykonać następujące polecenia:

```
$ sudo modprobe loop
$ sudo losetup -f
```

Po wykonaniu drugiego polecenia konsola zwróci informacje o adresie wirtualnego urządzenia. Następnie należy zamontować obraz pod tym adresem.

```
$ sudo losetup /dev/loop0 ~/Desktop/obraz.img
$ sudo partprobe /dev/loop0
```

Następnym krokiem jest zmniejszenie rozmiaru partycji. Wykorzystany do tego zostanie program GParted, za jego pomocą można usunąć nieużywane miejsce na partycji. Do konsoli wpisujemy:

```
$ sudo gparted /dev/loop0
```

Ukaże nam się okno programu GParted. Należy wybrać partycję na której zainstalowany jest system, a następnie zmniejszyć jej rozmiar do najmniejszego jaki można. Następnie należy akceptować zmiany za pomocą zielonego ptaszka. Kiedy program GParted zakończy pracę, można go wyłączyć i następnie wpisać polecenie w konsoli:

```
$ sudo losetup -d /dev/loop0
```

Ostatnim krokiem jest przycięcie obrazu. Aby otrzymać dane potrzebne do obcięcia obrazu należy wpisać w konsoli:

```
$ fdisk -l ~/Desktop/obraz.img
```

Następnie przycinamy obraz za pomocą polecenia:

```
$ truncate --size=$[(OSTATNI_BLOK+1)*ROZMIAR_BLOKU] ~/Desktop/obraz
    .img
```

, gdzie OSTATNI_BLOK to liczba w kolumnie koniec ostatniego bloku zmniejszonej partycji, a ROZMIAR_BLOKU to 512.

Po wykonaniu tych operacji obraz zostanie maksymalnie zmniejszony. Teraz wystarczy go wgrać na kartę pamięci oraz zainstalować system na Nvidii Nano.

4.2.2 Komunikacja z Nvidą Nano

Jeśli nawiązaliśmy połączenie z Nvidią Nano możemy dokonać odczytu m.in. z RP Lidaru oraz z kamery. Gdy wejdziemy do okna konsoli musimy początkowo wpisać poniższą komendę.

```
$ sudo chmod 666 /dev/ttyUSB0
```

Pozwoli ona na zmianę trybu portu USB, pod który podpięty jest RP Lidar A1, w celu jego poprawnego działania.
Następnie musimy pobrać potrzebne pliki obsługujące RP Lidar A1.

```
$ cd catkin_ws/src
$ git clone https://github.com/robopeak/rplidar_ros
```

W kolejny korku musimy zbudować pakiet. Wpisujemy więc poniższe komendy.

```
$ cd ..
$ catkin_make
$ source devel/setup.bash
```

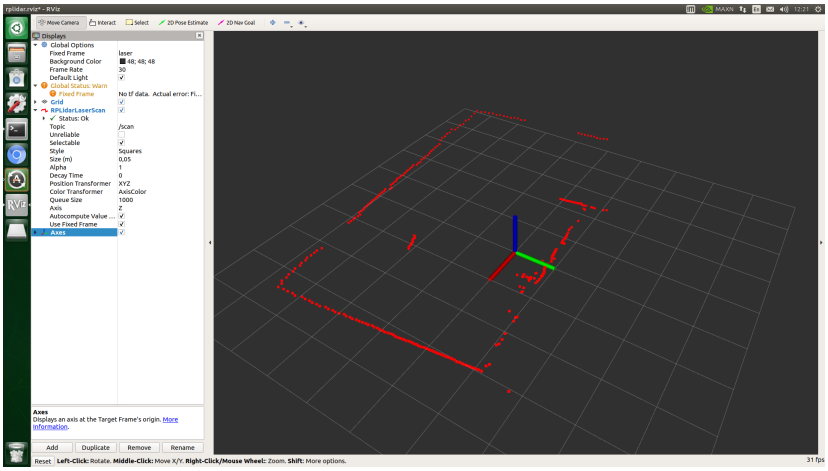
Następnie w celu uruchomienia odczytów z RP Lidaru w środowisku RViZ musimy uruchomić najpierw w terminalu roscore'a poprzez poniższą komendę.

```
$ roscore
```

Następnie w drugim oknie terminala wpisać poniższe polecenie, które uruchomi środowisko RViZ

```
$ roslaunch rplidar_ros view_rplidar.launch
```

Rezultat odczytów z RP Lidaru A1 w środowiska RViZ widoczny na poniższym zdjęciu

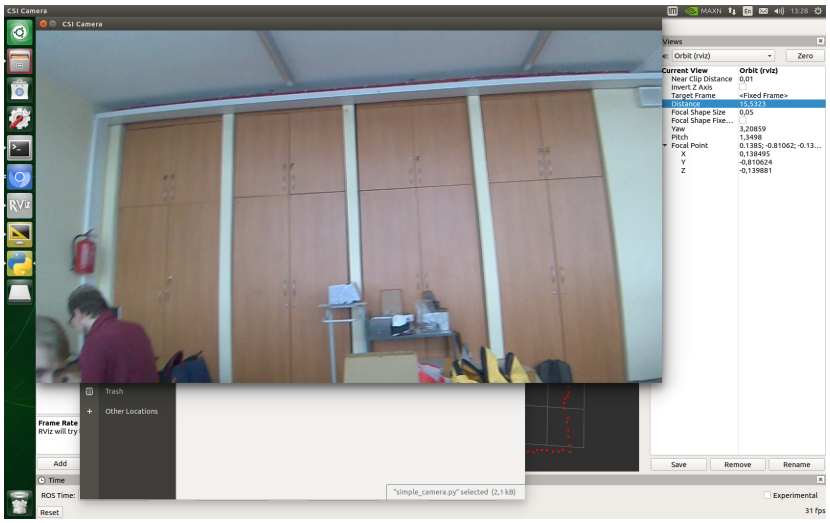


Rysunek 4.1: Widok środowiska RViZ ukazujący odczyty z RP Lidaru A1

W celu uruchomienia widoku z kamery zamontowanej na naszej platformie musimy uruchomić terminal i wpisać poniższe polecenia.

```
$ cd camera
$ python simple_camera.py
```

Po wpisaniu tych dwóch poleceń zostanie uruchomione nowe okno, w którym będziemy mogli obejrzeć obraz rejestrowany w czasie rzeczywistym przez zamontowaną na platformie kamerę. Rezultat widoczny poniżej.



Rysunek 4.2: Widok obrazu z kamery

Kolejnym kluczowym elementem realizowanego projektu było nawiązanie komunikacji między STM 32 F4, a Nvidą Nano. Mikrokontrolery zostały połączone scrossowanym przewodem ethernetowym. W pierwszej kolejności uruchomiliśmy okno terminala i musieliśmy zmienić dane karty sieciowej Nvidii z domyślnych na przedstawione poniżej:
IP: 192.168.1.120
Maska: 255.255.255.0
Bramka domyślna: 192.168.1.1
Aby to zrealizować należy umieścić w oknie konsoli poniższe polecenia.

```
$ sudo ifconfig eth0 192.168.1.120 netmask 255.255.255.0
$ sudo route add default gw 192.168.1.1 eth0
```

Kolejną krokiem była instalacja potrzebnych pakietów. Pierwszy z nich do ROS Control. Pakiety ros control przyjmuje jako dane wejściowe dane z enkoderów silnika platformy. W celu instalacji tego pakietu w uruchomionym oknie konsoli należy wpisać poniższe polecenie.

```
$ sudo apt-get install ros-melodic-ros-control ros-melodic-ros-
   controllers
$ catkin_make
$ source devel/setup.bash
```

Następnie musimy zainstalować pakiet RQT wraz z potrzebnymi pluginami. RQT jest to graficzny interfejs użytkownika, który implementuje różne narzędzia i interfejsy w postaci wtyczek. W celu instalacji potrzebnych pakietów konieczne jest wykonanie poniższych poleceń.

```
$ sudo apt-get install ros-melodic-rqt
$ sudo apt-get install ros-melodic-rqt-common-plugins
```

```
$ sudo apt-get install rqt_robot_plugins
$ catkin_make
$ source devel/setup.bash
```

W przypadku pojawienia się problemów / błędów konieczna jest instalacja pojedynczych wtyczek (pluginów) za pośrednictwem githuba. W naszym przypadku pojawił się taki problem dla paru pluginów. Rozwiązanie zaprezentujemy poniżej dla jednego z pluginów - *four wheel steering msgs*. W celu instalacji tej pojedynczej wtyczki należy wpisać poniższe polecenia w oknie terminala.

```
$ git clone https://github.com/ros-drivers/four_wheel_steering_msgs
$ cd four_wheel_steering_msgs
$ git checkout melodic-devel
$ cd ..
$ cd ..
$ catkin_make
$ source devel/setup.bash
```

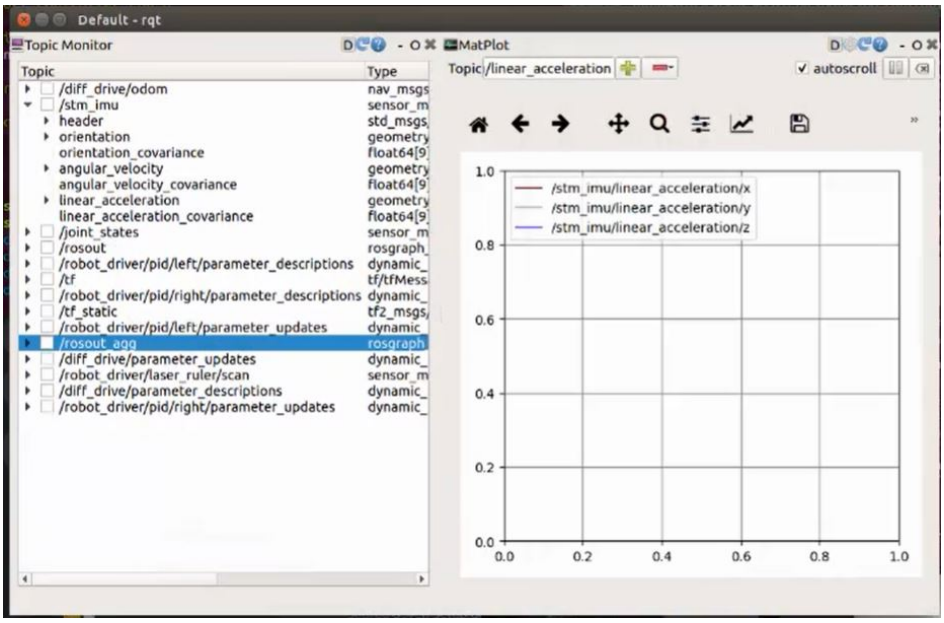
Użycie polecenia *git checkout melodic-devel* jest konieczne w przypadku gdy zainstalowany pakiet nie jest dedykowany dla używanej wersji ROSa, ale posiada możliwość migracji na inne wersje. Jeśli wszystkie pakiety poprawnie zostały poprawne zbudowane, możemy przejść do nawiązania połączenia z STM32 i rozpoczęcia przesyłania danych. W pierwszej kolejności musimy się upewnić czy mikrokontoler STM jest zasilony i posiada wgraną poprawną konfigurację wraz z całym projektem. W kolejnym kroku należy wpisać w konsoli poniższe polecenie.

```
$ roslaunch vc200_controller vc200_controller.launch
```

Polecenie to uruchomi roscore'a oraz przygotowany projekt, który nawiąże łączność z STM 32 F4. Po chwili w oknie konsoli ukażą się komunikaty o nawiązaniu połączenia z poszczególnymi elementami platformy UGV. Gdy to nastąpi w nowym oknie konsoli możemy uruchomić graficzny interfejs użytkownika RQT. W tym celu w oknie konsoli należy wpisać poniższe polecenie.

```
$ rqt
```

Polecenie to uruchomi nowe okno programu. Aby dodać elementy (wtyczki) do naszego okna, konieczne jest uruchomienie programu w trybie pełnoekranowym, a następnie z rozwijanej zakładki *Plugins* wybranie początkowo dwóch wtyczek: *Monitor Topic, Plot*. Po wybraniu tych dwóch elementów okno GUI powinno przyjąć taką postać jak poniżej.

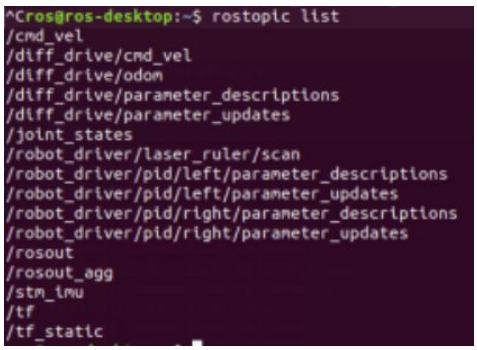


Rysunek 4.3: Widok okna środowiska RQT z uruchomionymi wtyczkami

W lewej części okna znajdują się wszystkie tematy wraz z poszczególnymi wartościami odpowiadającymi poszczególnym parametrom mierzonym w danych elementach naszej platformy i przesyłanych w ramkach do Nvidii Nano. W prawej części okno znajduje się okno, w któym możemy wyświetlić przebiegi w czasie dowolnych sygnałów przesyłanych do i wysyłanych z Nvidii. Przykładowo możemy wyświetlić dane dotyczące prędkości liniowej, prędkości kątowej, zorientowania naszej platformy, przyśpieszeniu, pozycji i innych. Również z poziomu konsoli możemy wyświetlić wszystkie tematy aktualnie realizowane w ROSie. Możemy tego dokonać poprzez wpisanie w konsoli poniższego polecenia.

```
$ rostopic list
```

Po wpisaniu polecenia w terminalu ukażą się wszystkie tematy. Przykładowy rezultat widoczny poniżej.



Rysunek 4.4: Widok okna terminala z realizowanymi tematami

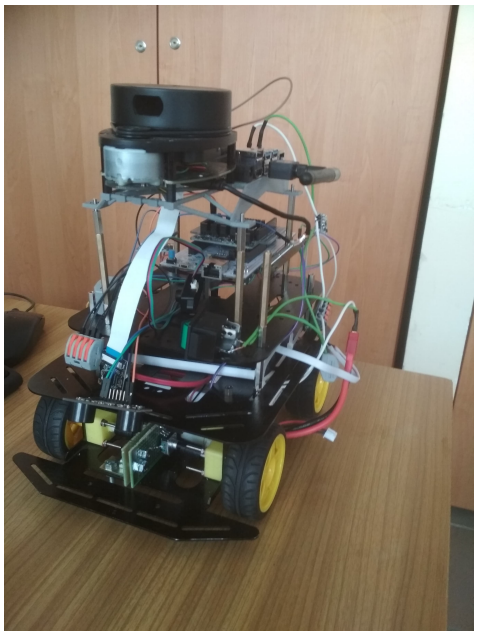
Rozdział 5

Rezultaty

W tym rozdziale przedstawiliśmy uzyskane przez nas rezultaty. Zarówno te związane z sekcją hardware'ową jak i software'ową.

5.1 Platforma

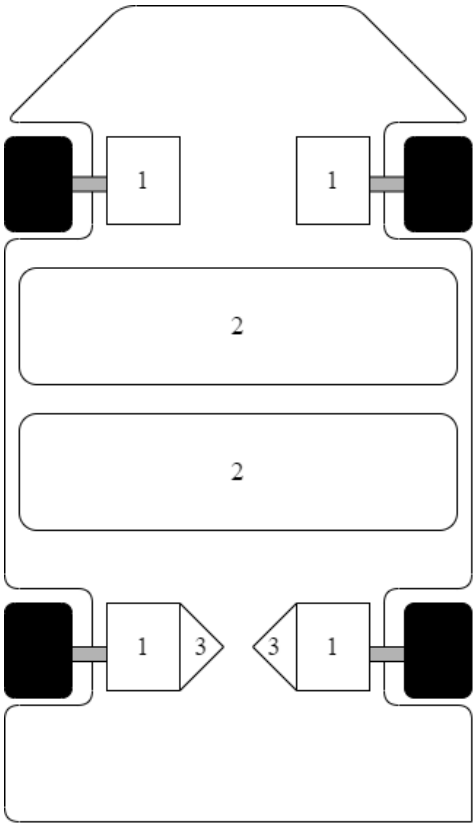
Wygląd finalny naszej platformy jest ukazany na poniższy zdjęciu:



Rysunek 5.1: Widok platformy

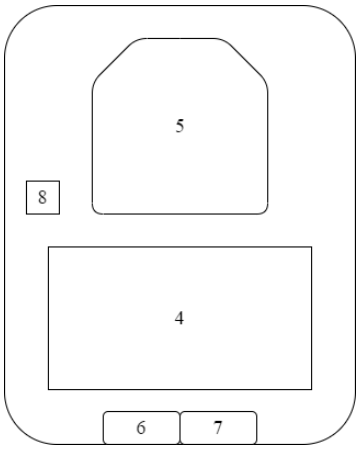
Można zauważyć iż nasza platforma składa się z paru poziomów. Było konieczne utworzenie kolejnych pięter na platformie, gdyż na jednym, bądź też dwóch nie bylibyśmy w stanie pomieścić wszystkich elementów przy takim rozmiarze platformy. Jeśli patrzeć na rozmieszczenie poszczególnych elementów części sprzętowej na poszczególnych poziomach wygląda to następująco (patrząc od podłoża):

- **Poziom 0** - oprócz silników (1) na tym poziomie znajdują się enkodery (3) oraz dwa akumulatory 7.4V (2) zasilające silniki oraz Nvidie Nano.



Rysunek 5.2: Widok poziomu 0 platformy

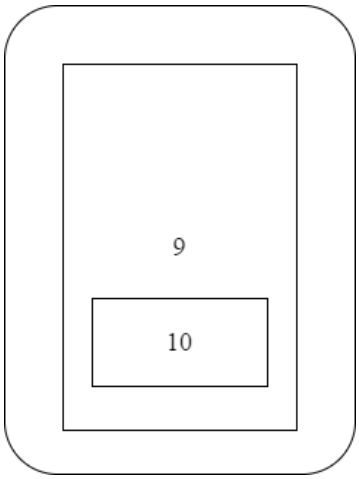
- **Poziom 1** - poziom wyżej został umieszczony powerbank (4) zasilający STM32 F4 oraz mostek H X-Nucleo IHM04A1 (5). Także umieściliśmy kilka czujników takich jak: czujnik ultradźwiękowy (6), mini lidar (7) oraz GPS (8).



Rysunek 5.3: Widok poziomu 1 platformy

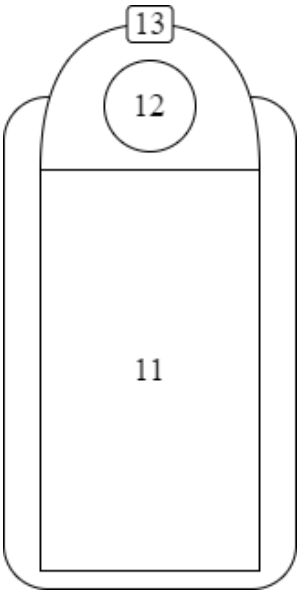
- **Poziom 2** - na tym poziomie umieściliśmy STM 32 F4 (9) oraz umieszczoną na

nim płytkę IMU X-Nucleo IKS01A3 (10).



Rysunek 5.4: Widok poziomu 2 platformy

- **Poziom 3** - na ostatnim - najwyższym poziomie znajduje się Nvidia Nano (11) połączona z RP Lidarem A1 (12) oraz kamerą (13). RP Lidar jest najwyższym punktem naszej platformy, gdyż jak wiadomo podczas obrotu lidaru, żadne elementy platformy nie powinny zasłaniać obszaru wysyłania wiązki przez lidar.



Rysunek 5.5: Widok poziomu 3 platformy

5.2 Odczyty danych z platformy

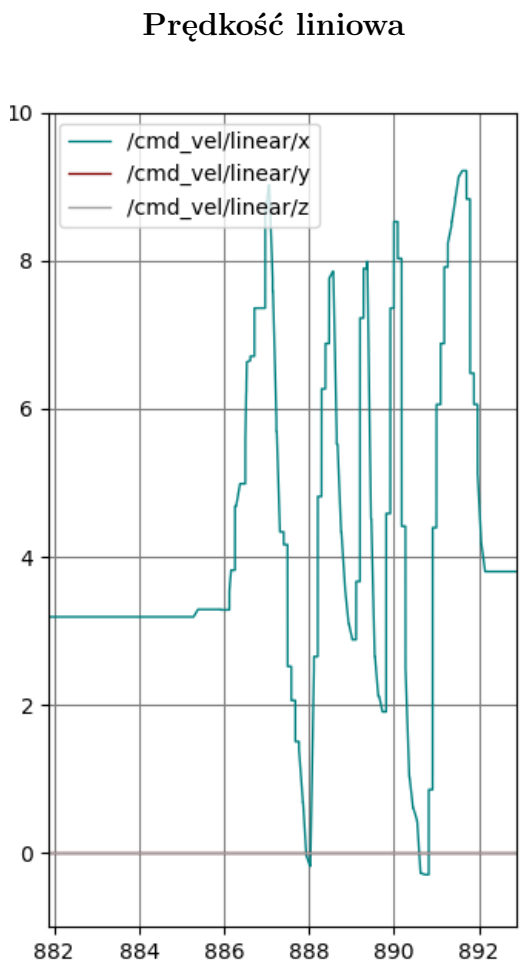
Badania których dokonaliśmy dotyczyły m.in sprawdzenia dokładności pomiaru dokonywanego RP Lidar A1. Wyniki przedstawione są w poniższej tabeli.

Tabela 5.1: Rezultaty pomiaru odległości przez RP Lidar A1.

Nr	**Zmierzona odległość przez RP Lidar A1**	**Rzeczywista odległość**
1	90 cm	85 cm
2	309 cm	297 cm
3	404 cm	401 cm
4	485 cm	489 cm

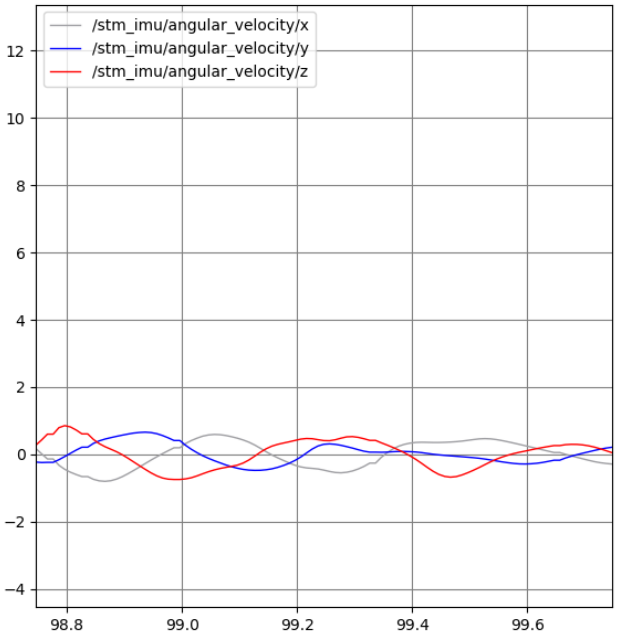
Możemy zauważyć iż dokładność lidaru jest dość wysoka. Maksymalny błąd lidaru wynosi 12 cm.

Kolejne testy jakie zostały przeprowadzone dotyczyły sprawdzenia działania płytki IMU oraz Enkoderów. Gdy uzyskaliśmy połączenie pomiędzy STM32 F4, a Nvidią Nano byliśmy w stanie wykreślić charakterystyki przy użyciu graficznego interfejsu użytkownika RQT, a dokładniej wtyczki *Plot*. Uzyskane rezultaty widocznie są na poniższych rysunkach.



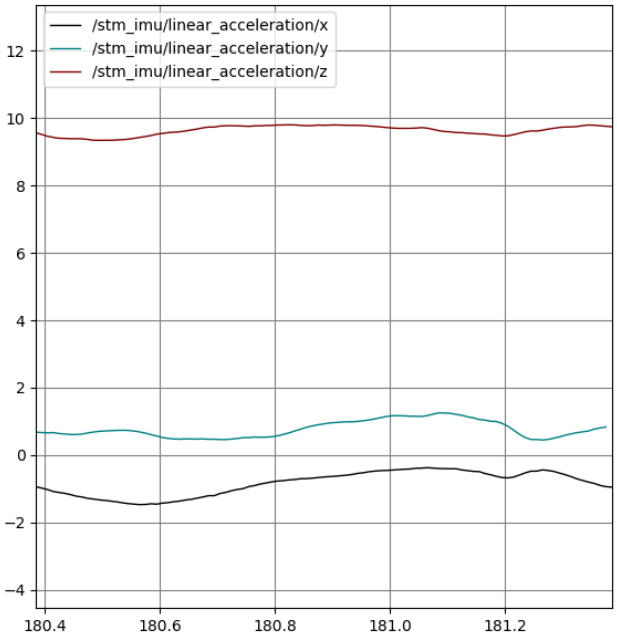
Rysunek 5.6: Przebieg prędkości liniowej w czasie względem osi x,y i z w m/s

Prędkość kątowa



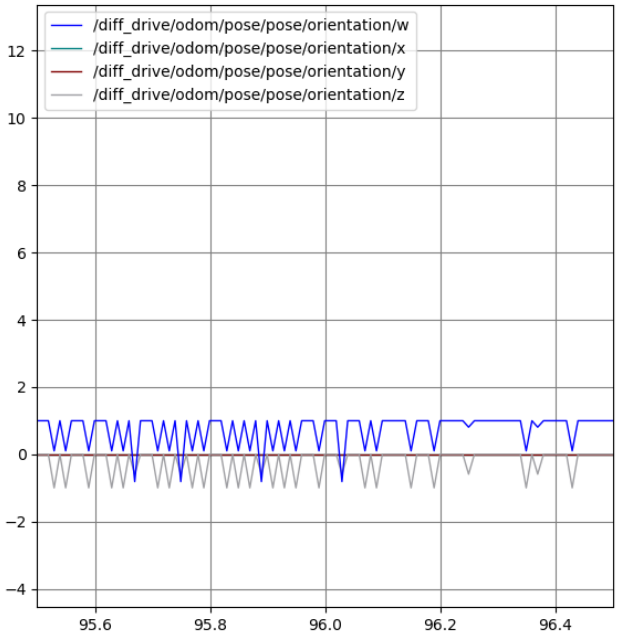
Rysunek 5.7: Przebieg prędkości kątowej w czasie względem osi x,y i z w rad/s

Przyśpieszenie liniowe



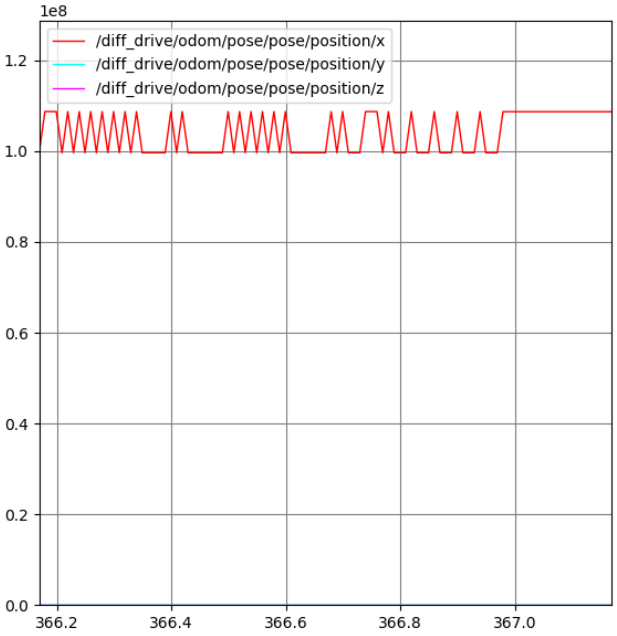
Rysunek 5.8: Przebieg przyśpieszenia liniowego w czasie względem osi x,y i z w m/s^2

Orientacja



Rysunek 5.9: Orientacja platformy względem poszczególnych osi w czasie

Pozycja



Rysunek 5.10: Pozycja platformy względem poszczególnych osi w czasie

Rozdział 6

Podsumowanie

Podsumowując, podczas prac wykonywanych w semestrze zimowym 2020 / 2021 udało nam się zrealizować stanowisko laboratoryjne pozwalające na badanie zależności pomiędzy architekturą sprzętową a metodyką implementacji wybranych zagadnień związanych z projektowaniem i realizacją pojazdów autonomicznych klasy UGV. W tym celu skorzystaliśmy z platformy napędzaną czterema silnikami DC. Jest ona zasilana dwoma akumulatorami 7.4V o pojemności 2.2Ah każdy oraz jednym powerbankiem 5V o pojemności 8Ah. Platforma jest wyposażona w szereg sensorów takich jak: czujnik ultradźwiękowy, GPS, akcelerometr, żyroskop, magnetometr, lidar, enkodery i inne. Wszystkie dane z czujników oraz silników są odbierane przez mikrokontroler STM32 F4 a następnie kierowane w postaci odpowiednich ramek do Nvidii Nano, na której jest umieszczony obraz z oprogramowaniem ROS. W systemie Robot Operating System (ROS) wykorzystujemy m.in. biblioteki ROS control i Rviz. Sterowanie platformą początkowo było realizowane z poziomu STM32 poprzez wgranie konkretnych ramek. Finalnie wysyłamy rozkazy z poziomu graficznego interfejsu użytkownika RQT i wtyczki *Robot Steering*. Dane wyświetlane są również w graficznym interfejsie użytkownika RQT natomiast w tym przypadku używamy wtyczki *Plot*.

Ze względu na pandemią SARS-CoV-2 udało nam się zrealizować oczekiwane wyniki tylko w zakresie podstawowym. Opracowaliśmy system kontrolno pomiarowy umożliwiający akwizycje danych z podłączonych sensorów oraz sterowanie platformą UGV, co możemy zauważyć na wykresach odczytów danych z sensorów w rozdziale piątym. Opracowaliśmy metody komunikacji stacji nadzorczej z modułami pomiarowymi i platformą UGV dla potrzeb monitoringu, wizualizacji i analizy danych. Zostało to uzyskane za pomocą systemu ROS, dzięki któremu odczytywaliśmy dane z sensorów, enkoderów oraz mieliśmy dostęp do systemu wizyjnego w postaci kamery.

Jeśli rozpatrzeć realizację kamieni milowych udało nam się zrealizować 3 z 4 przyjętych punktów. Udało nam się opracować stanowisko pozwalającego na akwizycje danych z wybranych sensorów przez skonstruowaną platformę UGV. Aktualnie pobieramy dane tylko z płytki IMU oraz z enkoderów. W przyszłości jednak z pewnością dane z wszytkich czujników będą transportowane i analizowane. Udało nam się również opracować metody komunikacji stacji nadzorczej z platformą UGV i modułami pomiarowymi. Z drobnymi problemami ale udało nam się nawiązać połączenie początkowo pomiędzy mikrokontrolerem STM32 F4, a sensorami oraz silnikami, następnie utworzyć połączenie pomiędzy mikrokontrolerem Nvidą Nano, a RP Lidarem A1 oraz kamerą aż w końcu finalnie utworzenie połączenia i działającej komunikacji między wyżej wymienionymi mikrokontrolerami. Monitorowanie, wizualizacja i analiza danych z platformy UGV odbywa się aktualnie w systemie ROS z wykorzystaniem graficznego interfejsu użytkownika RQT oraz dołączonego do nich wtyczek *Topic monitor*, *Plot* oraz *robot*

Steering oraz pakietu *RViZ*.

Znaczną część naszego projektu poświęciliśmy na część sprzętową opisywanego zagadnienia. Było to związane z wieloma problemami, które napotkaliśmy oraz z ograniczeniami w komunikacji, czasie i sprzęcie, którym nie dysponowaliśmy w domu. Napotkane problemy dotyczyły m.in. enkoderów, gdyż zdecydowaliśmy się nie używać dedykowanych enkoderów do platform DF Robot Pirate-4WD, tylko bardziej dokładnych enkoderów dedykowanych do płytek stykowych. Kolejny problem jaki napotkaliśmy i nad którym musieliśmy poświęcić więcej czasu to problem z wgraniem obrazu na Nvidie Nano oraz zasilenie jej. Początkowo próbowaliśmy wieloma metodami skompresować obraz o rozmiarze ok. 30GB na kartę SD o pojemności 16 GB. Żaden z zastosowanych przez nas sposobów niestety nie dał nam oczekiwanego efektu. Finalnie udało się wgrać skompresowany obraz na kartę SD o pojemności 32GB. Jednak kolejny problem pojawił się przy próbie zasilenia Nvidii Nano. Początkowo zasilaliśmy Nvidie dwoma akumulatorami 7.4V połączonymi w sposób szeregowy poprzez przetwornicę 5V. Jednak po poprawnym podłączeniu Nvidia załączała się i po chwili wyłączała. Problem tkwił w przewodach zasilających oraz w przetwornicy. Pierwszy element - przewody. Były one za cienkie i przez to nie mógł przez nie przepłynąć wystarczająco duży prąd. Drugi element - przetwornica. O ile napięcie wyjściowe zgadzało się tj. wynosiło 5V. Natomiast maksymalny prąd wyjściowy przetwornicy wynosił 2A, a było to niewystarczająco na nasze potrzeby. Po zmianie przetwornicy problem zniknął. Zasililiśmy Nvidie oraz Kamerę, a także RP Lidar A1, które zaczęły poprawnie działać. Problem również wystąpił podczas próby komunikacji pomiędzy Nvidą Nano, a STM32. Był on związany z brakiem komunikacji pomiędzy wspomnianymi mikrokontrolerami. W pierwszej kolejności musieliśmy doinstalować parę pakietów za pośrednictwem githuba. W kolejnym kroku przy pomocy programu *Wireshark* zdiagnozowaliśmy gdzie leży problem. Otóż niestety nie mieliśmy poprawnie skonfigurowanych adresów IP oraz maski. Gdy dokonaliśmy tego przy użyciu odpowiednich komend połączenie i wymiana danych pomiędzy mikrokontrolerarami została dokonana.

W przyszłości platforma może być rozwinięta o część programistyczną: sprowadzenie sterowania platformy z poziomu Nvidii do poziomu autonomicznego przy użyciu techniki gmappingu, biblioteki którą oferuje ROS Navigation Stack. Również może zostać zrealizowane odczytywanie danych z pozostałych czujników takich jak czujnik ultradźwiękowy, czy też mini-lidar. W tym semestrze niestety nie udało nam się zrealizować odczytu wszystkich danych z powodu ograniczeń obowiązujących z występującą pandemią SARS-CoV-2.

Bibliografia

[1] https://www.ros.org/history/

[2] https://botland.com.pl

[3] https://kamami.pl/

[4] https://www.ceneo.pl/