Politechnika Śląska
Wydział Mechaniczny Technologiczny

Projekt Inżynierski

System sterowania podwoziem robota eksploracyjnego opracowany w oparciu o środowisko Robot Operating System

Wykonał : Mariusz Lenczyk
Prowadzący projekt: dr hab. inż. Piotr PRZYSTAŁKA, prof. PŚ
Opiekun: dr inż. Wawrzyniec PANFIL
Gliwice, 2020

Agenda prezentacji

1. Cel, zakres i założenia projektu
2. Obiekt badań
3. Modernizacja układu sterowania
4. Projekt systemu sterowania
5. Oprogramowanie sterujące
6. Weryfikacja działania
7. Podsumowanie

Wstęp

- SKN AI-METH
- Silesian Phoenix
- European Rover Challenge

Rys. 1. Logo SKN AI-METH [1]

Rys. 2. Logo Silesian Phoenix [2]

Rys. 3. Zespoł Silesian Phoenix wraz z łazikiem Phoenix II na European Rover Challenge [3]

Cel, zakres i założenia projektu

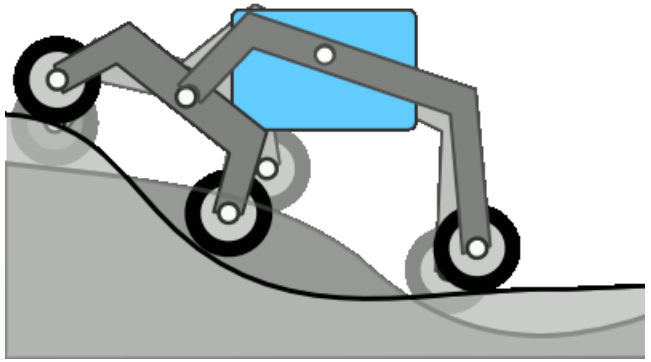
- Celem projektu było wykonanie systemu sterowania podwoziem łazika marsjańskiego
- Analiza dostępnego sprzętu, implementacja oprogramowania, wykonanie układu elektronicznego, integracja systemu
- Założenia wynikające z regulaminu ERC, założenia wynikające z istniejącego systemu, założenia dodatkowe

Założenia projektu

- Maksymalna prędkość 0.5 m/s
- Możliwość pracy w trybie manualnym, automatycznym oraz autonomicznym
- Prostota obsługi i dalszego rozwoju systemu
- Strategie jazdy: czołgowa oraz obrotowa i skrętna
- Trzy stopnie prędkości jazdy
- Sprzężenie zwrotne z systemu o stanie jego pracy
- Sterowanie układem zasilania i bezpieczeństwa oraz lampką sygnalizacyjną
- Sterowanie za pomocą wielu urządzeń i kontrola priorytetu
- Integracja z Robot Operating System

Obiekt badań

- Podwozie robota Phoenix II
  - 4 osie skrętne
  - 6 silników napędowych
  - Zawieszenie rocker-bogie
  - Układ zasilania i bezpieczeństwa



Animacja zawieszenia rocker-boogie [4]

Rys. 4. Phoenix II ze skręconymi osiami i zawieszeniem rocker-boogie [3]

Modernizacja układu sterowania

- Składowe układu sterowania podwoziem robota Phoenix II
  - Komputer Operatora
  - Główna jednostka sterująca – Raspberry Pi
  - Sterownik PLC

Rys. 5. Raspberry Pi 3b [5]

Rys. 6. Sterownik PLC - CR721S [4]

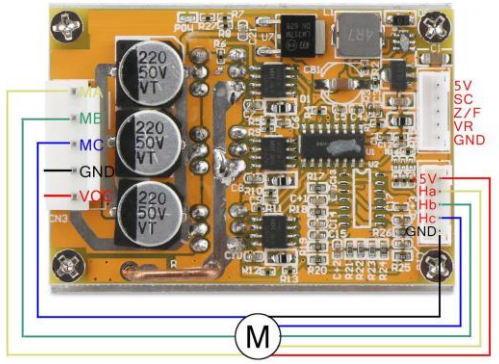
Rys. 7. Arduino Mega 2560 - rewizja 3 [6]

Układ napędowy

Rys. 9. Dwukanałowy sterownik DC MC33926 [7]

Rys. 8. Oś skrętna oraz silnik napędowy

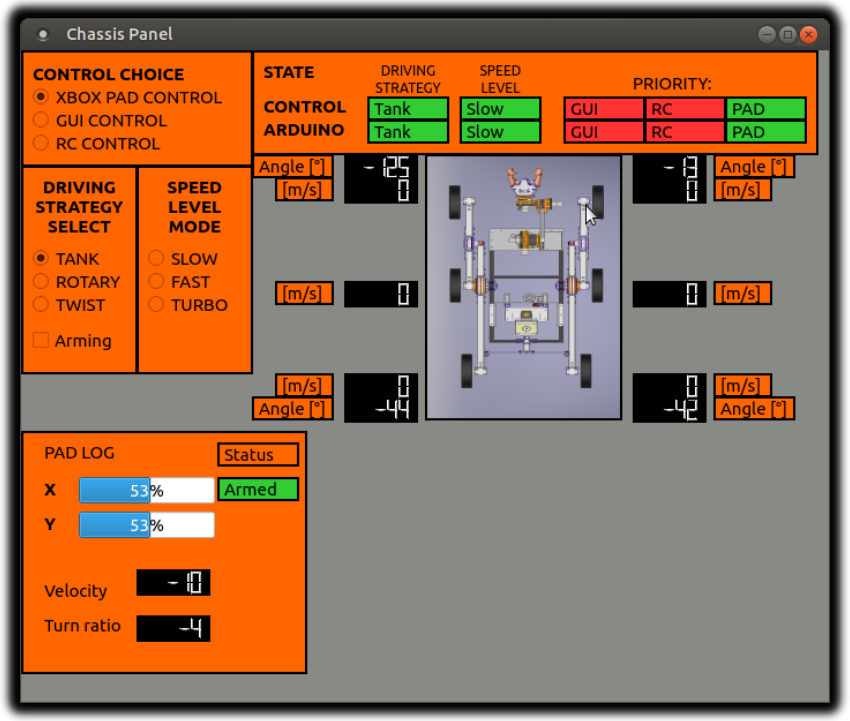
Rys. 10. Potencjometr 100k Ohm [8]



Rys. 11. Sterownik BLDC ZS-X11B [9]

Interfejs operatora

Rys. 12. Aparatura RC

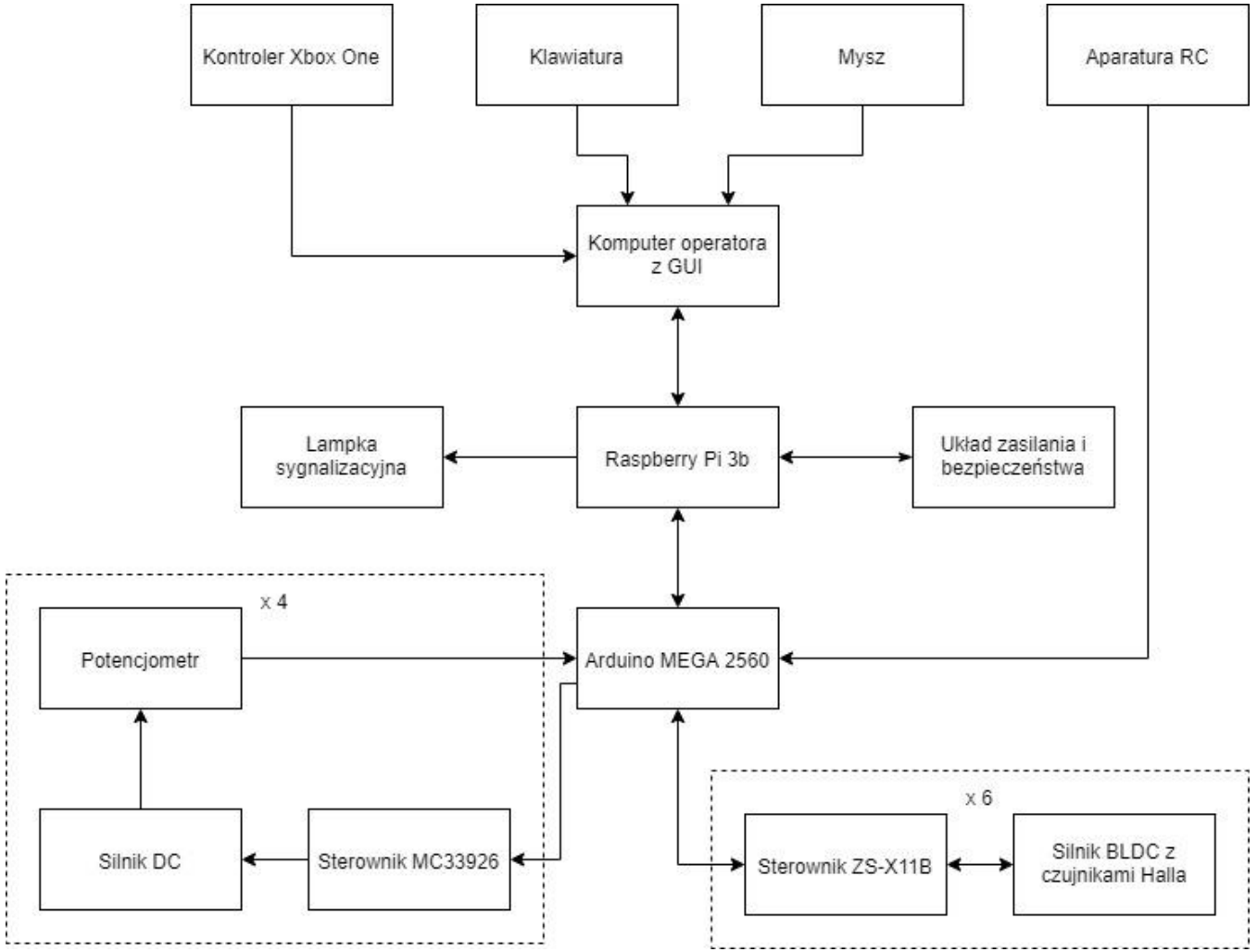


Rys. 13. Graficzny interfejs użytkownika – GUI (autor: Marcin Nagi)

Rys. 14. Kontroler Xbox One

Schemat ideowy systemu

- Rdzeń całego systemu
  - Komputer Operatora
  - Główna jednostka sterująca – Raspberry Pi
  - Arduino Mega



Rys. 15. Schemat ideowy systemu sterowania

Narzędzia wykorzystane przy tworzeniu oprogramowania

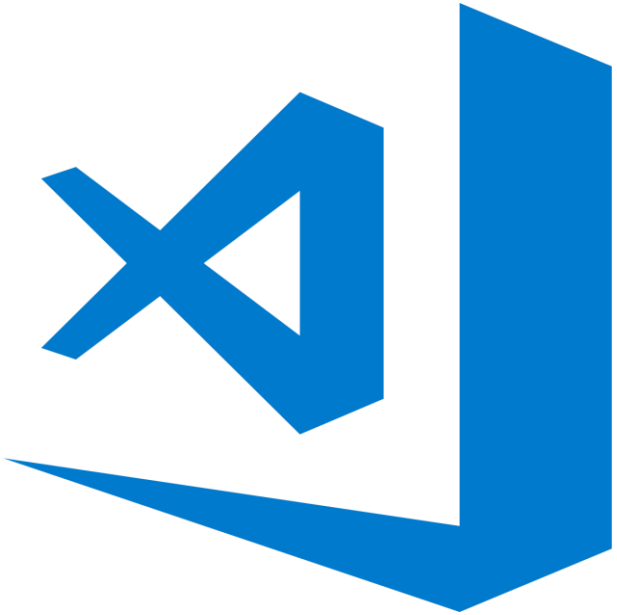
Robot Operating System (ROS) – platforma programistyczna przystosowana do programowania robotów mobilnych

Rys. 16. Logo ROS Melodic [10]

Git – rozproszony system kontroli wersji

Rys. 17. Logo Git [11]

Visual Studio Code – darmowy edytor tekstu, wspierający wiele języków programowania



Rys. 18. Logo VS Code [12]

Oprogramowanie

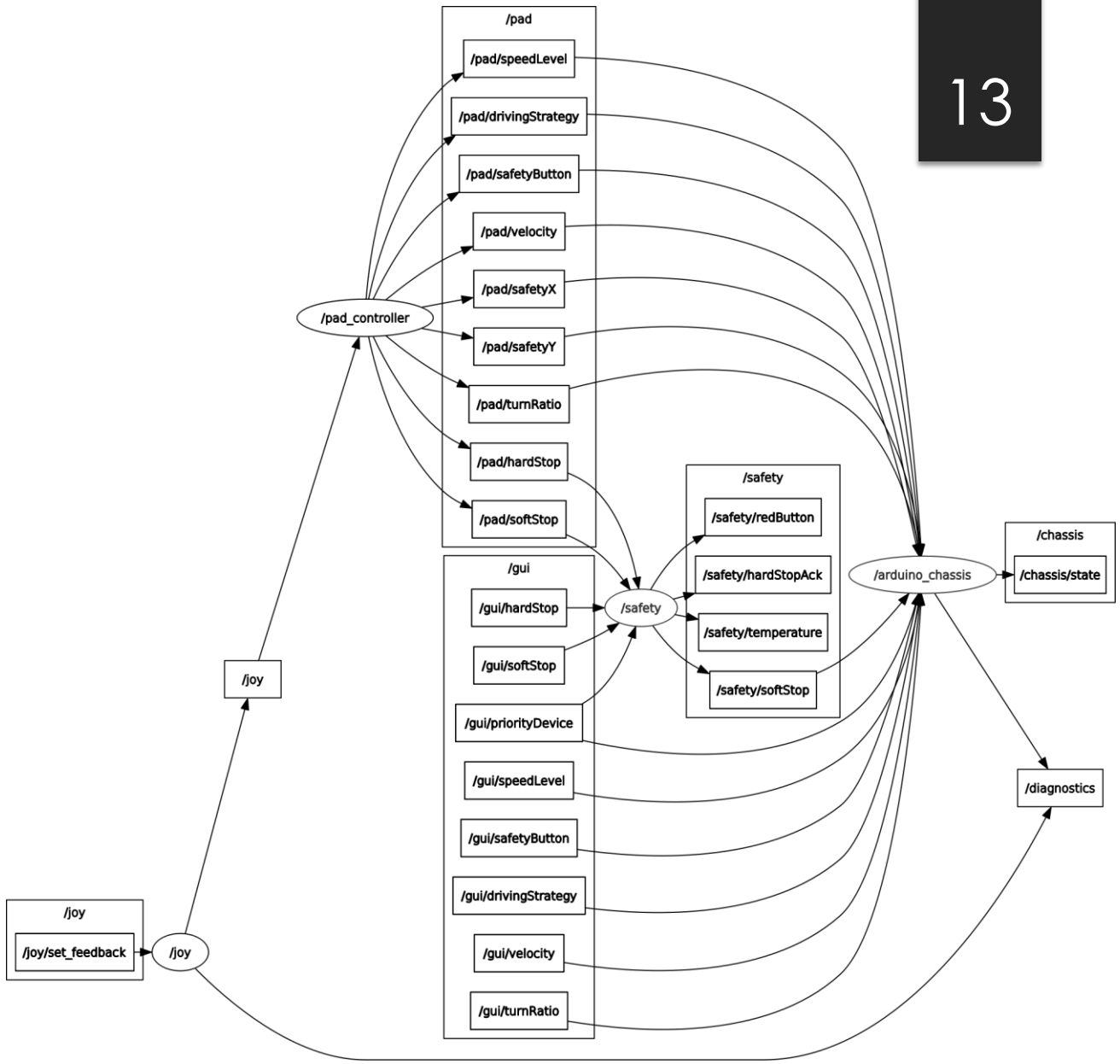
- Sterowanie układem zasilania i bezpieczeństwa
- Sterowanie lampką sygnalizacyjną
- Sterowanie podwoziem robota w pętli zamkniętej z wyróżnieniem kilku strategii jazdy
- 3 stopnie prędkości jazdy
- Procedura uzbrajania urządzeń sterujących
- Obsługa priorytetu sterowania dla wielu urządzeń sterujących
- Generowanie informacji zwrotnej o stanie pracy systemu

Nadawca	Odbiorcy	Temat	Typ wiadomości
joy	pad_controller	joy	sensor_msgs/Joy
pad_controller	gui, safety	/pad/drivingStrategy	std_msgs/UInt8
pad_controller	safety	/pad/softStop	std_msgs/Bool
pad_controller	safety	/pad/hardStop	std_msgs/Bool
pad_controller	gui, arduino_chassis	/pad/speedLevel	std_msgs/UInt8
pad_controller	gui, arduino_chassis	/pad/velocity	std_msgs/Int16
pad_controller	gui, arduino_chassis	/pad/turnRatio	std_msgs/Int16
pad_controller	gui, arduino_chassis	/pad/safetyX	std_msgs/UInt8
pad_controller	gui, arduino_chassis	/pad/safetyY	std_msgs/UInt8
pad_controller	gui, arduino_chassis	/pad/safetyButton	std_msgs/Bool
safety	gui, arduino_chassis	/safety/softStop	std_msgs/Bool
safety	gui	/safety/redButton	std_msgs/Bool
safety	gui	/safety/temperature	std_msgs/Float64
safety	gui	/safety/hardStopAck	std_msgs/Bool
gui	arduino_chassis, safety	/gui/priorityDevice	std_msgs/UInt8
gui	safety	/gui/drivingStrategy	std_msgs/UInt8
gui	safety	/gui/softStop	std_msgs/Bool
gui	safety	/gui/hardStop	std_msgs/Bool
gui	arduino_chassis	/gui/speedLevel	std_msgs/UInt8
gui	arduino_chassis	/gui/velocity	std_msgs/Int16
gui	arduino_chassis	/gui/turnRatio	std_msgs/Int16
gui	arduino_chassis	/gui/safetyButton	std_msgs/Bool
arduino_chassis	gui	/chassis/state	custom_msgs/ChassisState

Rys. 20. Tabela tematów sieci ROS

Oprogramowanie

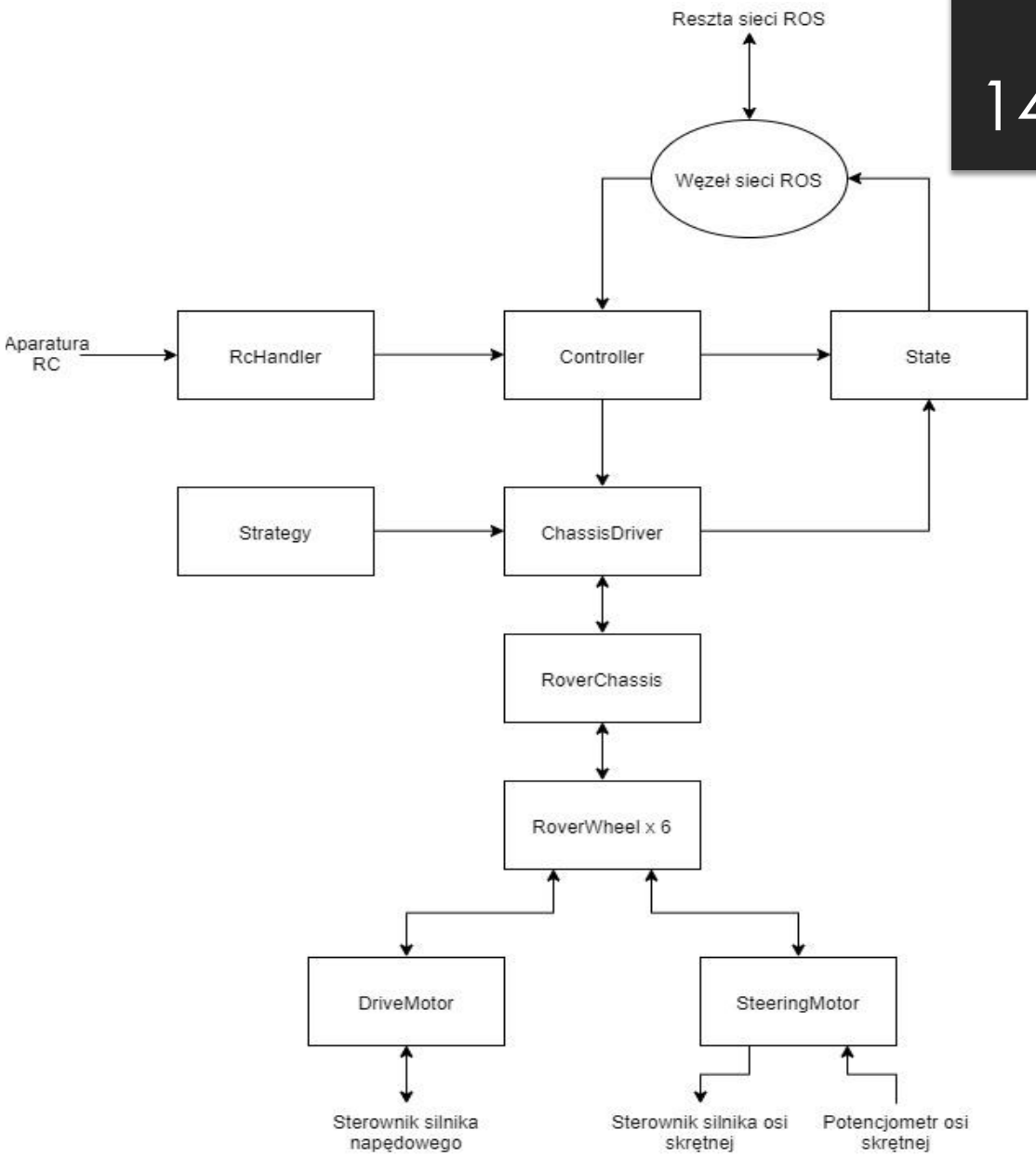
- Węzły w sieci ROS
  - joy
  - pad_controller
  - safety
  - arduino_chassis



Rys. 21. Sieć ROS

Oprogramowanie

- Najważniejsze byty:
  - Controller
  - State
  - ChassisDriver
- Komunikacja z ROS'em
- Wzorzec projektowy - Strategia
- Język programowania: C++



Rys. 22. Schemat blokowy oprogramowania na Arduino

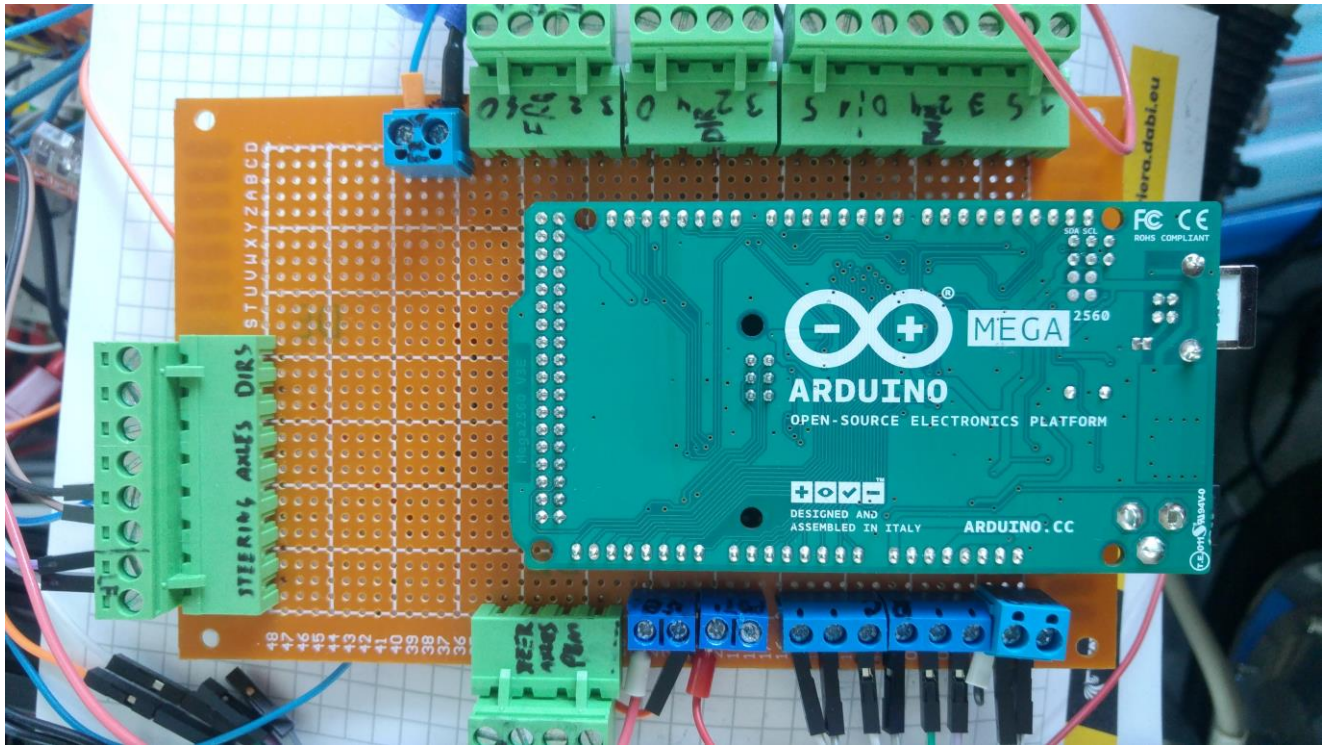
Oprogramowanie

Opis	**Informacja zwrotna - *ChassisState***
Priorytet sterowania - aparatura RC	priorityDevice = 0
Faza uzbrajania - faza rozbrojenia	armingState = 1
Strategia jazdy – czołgowa	drivingStrategy = 0
Stopień prędkości jazdy – wolny	speedLevel = 0
Prędkości obrotowe	driveSpeed = [0,0,0,0,0,0]
Orientacje kątowe	steerAngle = [0,0,0,0]

Rys. 19. Przykładowa informacja zwrotna – stan początkowy systemu

Układ elektroniczny

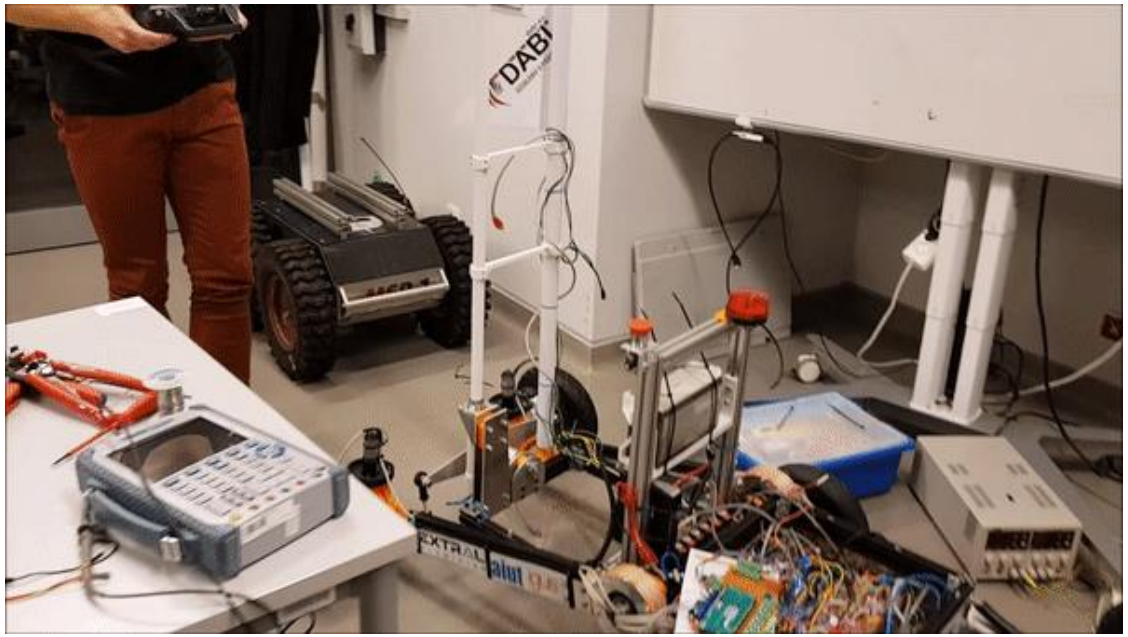
Autorski shield na Arduino integrujący go z pozostałą częścią układu sterowania



Rys. 23. Układ elektroniczny połączeń z Arduino

Weryfikacja

- Szereg testów jednostkowych
- Testy integracyjne
- Oględziny wizualne zachowania robota



Wnioski

- Projekt można określić jako udany. Większość wymagań została w pełni spełniona - z małymi wyjątkami.
- Projekt wykrył kilka niedoskonałości obecnego układu elektrycznego jak i konstrukcji mechanicznej oraz dostarczył informacji na temat koniecznych modernizacji.
- Projekt jest bardzo rozwojowy, a wraz z wymaganymi modernizacjami konieczna będzie ciągła dbałość o utrzymanie kodu i przeprowadzenie testów końcowych.

Podsumowanie wymagań

Wymaganie	Opis realizacji
Maksymalna prędkość 0.5 m/s	Nie można jednoznacznie określić, jednak oprogramowanie wyróżnia kilka stopni prędkości jazdy, więc istnieje możliwość konfiguracji prędkości maksymalnej
System sterowania powinien uwzględniać możliwość pracy w trybie manualnym, automatycznym oraz autonomicznym	Oprogramowanie zostało zaprojektowane tak, aby dodanie nowej strategii ruchu oraz nowego trybu dla lampki sygnalizacyjnej było bardzo proste

Podsumowanie wymagań

Wymaganie	Opis realizacji
Prostota obsługi i dalszego rozwoju systemu	Ocena realizacji tego wymagania jest bardzo subiektywna. Sama idea obsługi została dobrze odebrana przez osoby które się z nią zapoznały, lecz do tej pory żadna osoba nie pracowała z powstałym systemem i kodem źródłowym. Jednak ze względu na użycie powszechnie wykorzystywanych elementów i technologii, uważam, że można uznać za zrealizowane
Strategie jazdy: czołgowa oraz obrotowa i skrętna wykorzystujące skręcanie przy pomocy osi skrętnych	Zrealizowano i przetestowano na stanowisku testowym jednak nie ma gwarancji czy założone ustawienie kół w strategii obrotowej jest odpowiednie, a konstrukcja mechaniczna wręcz uniemożliwia przetestowanie trybu skrętnego

Źródła

[1] https://sknaimeth.polsl.pl/

[2] https://sknaimeth.polsl.pl/lazik-marsjanski/

[3] Materiały Silesian Phoenix

[4] https://www.ifm.com/pl/pl/product/CR721S

[5] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[6] https://store.arduino.cc/arduino-mega-2560-rev3

[7] https://www.pololu.com/product/2503

[8] https://botland.com.pl/pl/potencjometry/6263-potencjometr-obrotowy-100k-liniowy-18w-z-wylacznikiem.html

[9] https://www.gotronik.pl/sterownik-silnika-bldc-5v-36v-350w-p-4987.html

[10] http://wiki.ros.org/melodic

[11] https://pl.wikipedia.org/wiki/Git_(oprogramowanie)

[12] https://pl.wikipedia.org/wiki/Visual_Studio_Code

Dziękuję za uwagę