Painters

<niegdysiejszy tytuł roboczy>

Autorzy:  
Bartłomiej Strózik  
Adam Szreter

# Założenia i cele projektu

Celem projektu było zaprogramowanie pary robotów mobilnych tak, aby zdołały przejechać narysowaną w programie graficznym i wgraną do ich pamięci trasę, skutecznie unikając kolizji. Podczas realizacji zadania roboty miały opierać się na znajomości własnej pozycji (na podstawie pomiaru obrotu kół za pomocą enkoderów) i komunikacji radiowej (w technologii Bluetooth Low Energy).

# Zasoby

Nasze roboty oparliśmy o następujące zasoby sprzętowe:

* płytka ESP32 DevKitC V4 (z wbudowanym modułem bluetooth) (dokumentacja: [ESP32](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf), [DevKitC-V4](https://www.espressif.com/sites/default/files/documentation/esp32-devkitc-v4_reference_design.zip))
* podwozie Magician Chassis z silnikami Dagu DG01D-A130 GearMotor (dokumentacja: [silniki](https://botland.com.pl/pl/index.php?controller=attachment&id_attachment=363))
* sterownik silników Pololu TB6612FNG ([dokumentacja](https://botland.com.pl/pl/index.php?controller=attachment&id_attachment=83))
* moduł enkoderów z robota SparkFun RedBot ([dokumentacja](https://botland.com.pl/pl/index.php?controller=attachment&id_attachment=520))

Do programowania robotów wykorzystaliśmy środowisko Arduino IDE 1.8.9.

# Repozytorium

## Adres

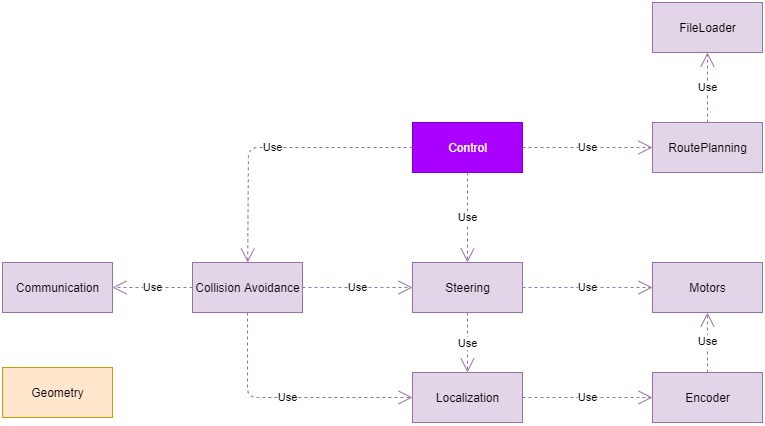
Projekt został upubliczniony przez umieszczenie go w publicznym repozytorium na GitHubie, dostępnym pod adresem: <https://github.com/tyrrr-aj/Painters.git>

## Struktura repozytorium

W głównym katalogu projektu znajdują się następujące pliki i katalogi:

* plik *Painters.ino* – jest to standardowy plik źródłowy platformy Arduino, zawierający główny kod wykonania programu (funkcje *setup* i *loop*).
* plik *Pins.h* – opisuje on odwzorowanie logicznych funkcji pinów w kodzie na ich fizyczne numery (zależne od zestawienia okablowania konkretnego egzemplarza robota)
* katalog *src* - zostały w nim umieszczone wszystkie biblioteki specyficzne dla projektu
* katalog *data* – przechowuje on pliki z trasami, jakie może przejechać robot

# Struktura projektu



Projekt został zbudowany modułowo. Głównym komponentem jest klasa **Control**, pełniąca zasadniczy nadzór nad robotem. Jej zadaniem jest prowadzenie robota po zadanej trasie (otrzymanej z modułu **RoutePlanning**) – do czego wykorzystuje moduł **Steering**, i utrzymywanie poprawnego działania modułu **CollisionAvoidance**. Moduł ten, jak wskazuje nazwa, odpowiada za monitorowanie, czy robotowi nie grozi kolizja, i reagowanie, jeśli dojdzie do takiej sytuacji. Wykorzystuje w tym celu znajomość lokalizacji robota (**Localization**) i komunikację z partnerem (**Communication**).

Do kontroli nad ruchem robota zarówno klasa **Control**, jak i **CollisionAvoidance** wykorzystują moduł **Steering**. Udostępnia on funkcjonalności takie, jak jazda do określonego punktu albo zatrzymanie robota i późniejsze wznowienie ruchu. Wykorzystuje on informacje o bieżącej lokalizacji i kursie (**Localization**) i wydaje polecenia silnikom (**Motors**).

Klasa Localization monitoruje aktualną względną pozycję i obrót robota – punkt odniesienia stanowi ustawienie początkowe. Posługuje się ona odczytami z enkoderów (**Encoder**). Moduł **Encoder** odpowiada za monitorowanie sygnałów z fizycznych enkoderów i udostępnianie liczby zliczonych ticków. Do realizacji tego zadania potrzebuje informacji o aktualnym kierunku ruchu, którą otrzymuje bezpośrednio od modułu **Motors** – odpowiedzialnego za kontrolę nad silnikami.

Moduł **Geometry** jest użytkowym modułem, udostępniającym różnorodne funkcje związane z planimetrią, niecharakterystyczne dla projektu, ale niezbędne w wielu miejscach programu. Wiele z modułów projektu korzysta z różnych składowych modułu **Geometry** – powiązania te zostały pominięte na schemacie, aby nie zaciemniać obrazu.

# Omówienie poszczególnych modułów

## Control

## Steering

## Motors

## Encoder

## Localization

## FileLoader

## RoutePlanning

## CollisionAvoidance

## Communication