DOI: https://doi.org/10.53052/9788367652148.24

Damian PYTKA[1]

Opiekun naukowy: Jacek RYSIŃSKI[2]

# OPTYMALIZACJA TORU RUCHU ROBOTA TYPU LINEFOLLOWER

**Streszczenie:** Przedmiotem pracy było opracowanie symulacji oraz optymalizacji toru ruchu robota typu linefollower. Wykonano projekt oraz prototyp robota. Podzespoły wykonano w technologii druku 3D. Przedstawiono kluczowe elementy projektu elektronicznego. Opisano proces optymalizacji toru ruchu robota ze względu na dwa kryteria: długość trasy oraz krzywiznę toru. Zbadano wpływ optymalizacji toru ruchu na czas przejazdu robota.

**Słowa kluczowe:** linefollower, druk 3D, projekt PCB, symulacja, optymalizacja

# OPTIMISATION OF THE LINEFOLLOWER ROBOT'S PATH

**Summary:** The aim of the study was to develop a simulation and optimisation of the path of a linefollower robot. A design and prototype of the robot was made. Components were made using 3D printing technology. Key elements of the electronic design are presented. The process of optimising the robot's path with respect to two criteria: path length and path curvature is described. The impact of path optimisation on the robot's travel time was investigated..

**Keywords:** linefollower, 3D printing, PCB project, simulation, optimization

## 1. Wstęp

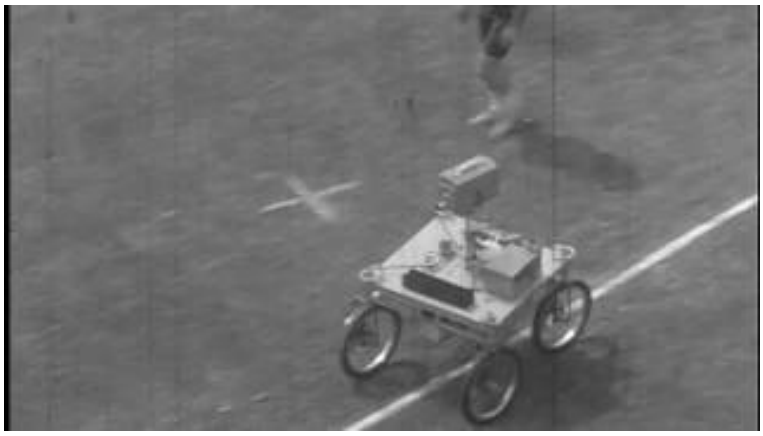
Robot linefollower (inaczej nazywany też liniowym robotem śledzącym) został zaprojektowany do śledzenia i podążania za liniami na podłodze. Najczęściej jest to czarna linia na jasnym tle, choć niektóre roboty linefollower potrafią pracować także z innymi kombinacjami kolorów.

Robot linefollower należy do podklasy robotów autonomicznych, które potrafią przemieszczać się względem swojego otoczenia (np. roboty jeżdżące, latające lub kroczące). Najpopularniejszym rodzajem robotów mobilnych są platformy samojezdne, poruszające się przy pomocy kół. Często potocznie mianem robota

[1] mgr inż., Uniwersytet Bielsko-Bialski, Wydział Budowy Maszyn i Informatyki, dpytka@ubb.edu.pl

[2] dr inż., Uniwersytet Bielsko-Bialski, Wydział Budowy Maszyn i Informatyki, jrysinski@ubb.edu.pl

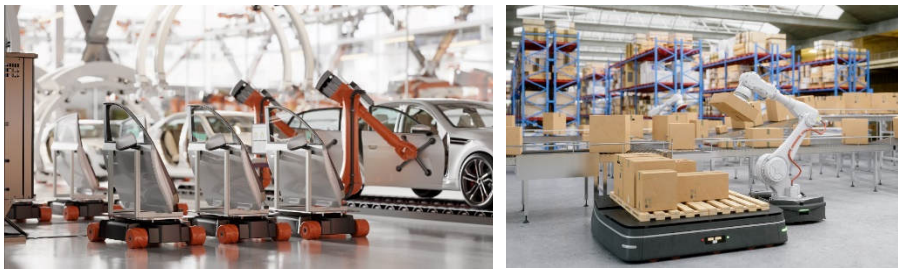
mobilnego (lub platformy mobilnej) określa się właśnie autonomiczne pojazdy kołowe [6]. Typowy robot linefollower wykorzystuje czujniki optyczne lub czujniki podczerwieni, aby "widzieć" linie na podłodze. Na podstawie odczytów z czujników podejmuje decyzje dotyczące swojego ruchu, takie jak skręcanie w lewo, w prawo lub utrzymanie prostej trasy.
Roboty linefollower są często stosowane w edukacji, w celu nauki podstaw programowania oraz robotyki. Bardzo często występują w konkursach, w których rywalizują ze sobą w śledzeniu liniii osiąganiu jak najszybszego czasu przejazdu przez trasę.
Przeglądając literaturę stwierdzono, iż pierwszym udokumentowanym robotem mobilnym typu line follower był Stanford Cart Mobile Robot. Pionierskie rozwiązanie opracowane na Uniwersytecie Stanforda w latach 60. Był to jeden z pierwszych robotów mobilnych, zdolnych do śledzenia linii na podłodze. Stanford Cart został zaprojektowany w celu badania sterowania robotami mobilnymi. Posiadał dwa koła napędowe z silnikami elektrycznymi, umieszczone na sztywnym podwoziu. Na przedniej części robota zamontowano czujnik w postaci fotokomórki, która mogła rejestrować jasność podłoża. Za pomocą tej fotokomórki robot był w stanie śledzić linie, które były rysowane na podłodze [9].



*Rysunek 1. Stanford Card, pierwszy robot śledzący linię [8]*

Sterowanie robotem odbywało się na podstawie informacji o jasności podłoża. Jeśli robot wykrył, że jest na linii, kontynuował jazdę prosto. Jeśli wykrył różnicę w jasności, wiedział, że zbliża się do skrzyżowania lub zakrętu. Na podstawie tych informacji robot podejmował decyzję o skręcie w odpowiednim kierunku.
Stanford Cart był ważnym krokiem w rozwoju robotyki mobilnej. Wykorzystanie prostego czujnika światła i algorytmu śledzenia linii pozwoliło na badanie podstawowych problemów związanych z nawigacją robotów w realnym środowisku. Choć stanowił początkowe podejście do line followerów, Stanford Cart przyczynił się do rozwoju technologii i algorytmów, które były później wykorzystywane w bardziej zaawansowanych robotach linefollower [10].
Konstrukcja robota typu linefollower stanowi pole badań, które łączy w sobie elementy elektroniki, programowania i mechaniki. Roboty konstruowane do udziału w zawodach robotów obarczone są dodatkowo szeregiem ograniczeń, zawartych w regulaminie konkurencji. Robot nie może być konstrukcją komercyjną, jego zarys

musi mieścić się w obszarze kartki A4, zabroniona jest komunikacja z robotem w trakcie przejazdu [1]. Dodatkowo konkurencja zwykle dzieli się na klasyczną i „Turbo". Związane jest to z zastosowaniem napędów tunelowych, które znacznie zwiększają przyczepność robotów, pozwalając zyskać przewagę nad konstrukcjami bez tego rozwiązania.
Biorąc udział w zawodach można zauważyć, że roboty zawodników niewiele różnią się od siebie od strony mechanicznej. Narzucone regulaminem ograniczenia, oraz optymalizacja parametrów robota do uzyskania jak najlepszego czasu sprawiły, że roboty zwykle posiadają taką samą konstrukcję. Żeby uzyskać przewagę na torze, trzeba usprawniać algorytmy sterujące robotem.
Regulamin konkurencji określa prawidłowy przejazd jako ten, w trakcie którego robot nie opuścił trasy. Robot nie musi śledzić linii dokładnie po środku, dopóki jego zarys nie wyjedzie całkowicie za linię, przejazd jest zaliczony. Ten zapis umożliwia jazdę po torze innym niż wyznacza sama linia, na przykład po torze najkrótszym, przejeżdżając każdy zakręt po wewnętrznej stronie. Optymalizacja toru ruchu jest zagadnieniem związanym z wszelkiego rodzaju wyścigami, na przykład Formułą 1. Algorytmy optymalizacji jak i symulacja były wykorzystywane przez zespoły już od lat 50 XX wieku [2].
Roboty typu linefollower znalazły szerokie zastosowanie w wielu dziedzinach, takich jak przemysł czy transport. Od automatycznych magazynów po inteligentne pojazdy autonomiczne, roboty linefollower przyczyniają się do zwiększenia efektywności, precyzji i bezpieczeństwa różnych procesów.



*Rysunek 2. a) Dostawcze roboty mobilne na linii produkcyjnej automotive; b) Paletyzacja z pomocą robotów AGV [7]*

Roboty linefollower mają wiele zastosowań przemysłowych, które przyczyniają się do zwiększenia efektywności, precyzji i bezpieczeństwa w różnych branżach.
Są wykorzystywane do testowania nowych rozwiązań technologicznych, algorytmów sterowania czy czujników. Mogą służyć jako platforma do eksperymentów i prototypowania, umożliwiając rozwijanie innowacyjnych rozwiązań.

## 2. Cel i zakres pracy

Celem pracy jest badanie metod optymalizacji toru ruchu robota typu linefollower.
Zakres pracy obejmuje:

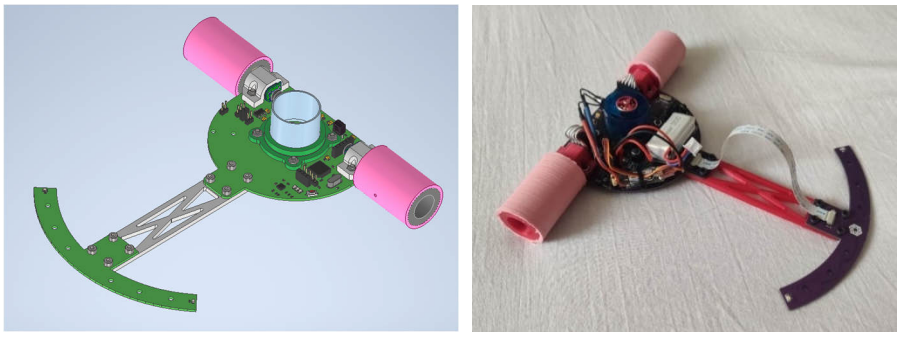
- projekt i wykonanie prototypu robota linefollower,
- wykonanie aplikacji symulacyjnej na podstawie prototypu,

- opracowanie metod optymalizacji toru ruchu, najkrótszej drogi i najmniejszej krzywizny toru,
- badanie wpływu optymalizacji toru ruchu na czas przejazdu symulowanego robota.

Prototyp robota musi spełniać wymagania zawarte w regulaminie zawodów. Długość nie może przekraczać 297 mm, a szerokość 210 mm. Robot musi być w stanie podążać za linią o szerokości 19 mm [1]. Elementy prototypu będą wykonane w technologii druku 3D FDM, odlewane z silikonu formierskiego oraz zamawiane u producenta PCB.
Symulacja oraz optymalizacja opracowane będą w języku Python, z użyciem niezbędnych bibliotek. Założono, że symulacja będzie zawierać graficzny interfejs użytkownika.

## 3. Konstrukcja robota

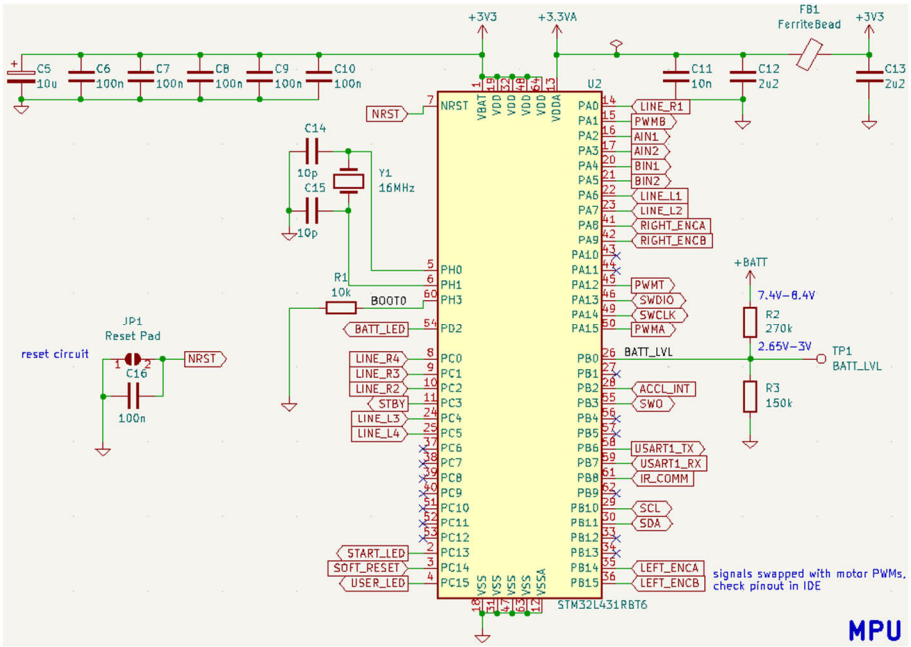
Projekt robota został wykonany z zastosowaniem oprogramowania KiCad oraz Autodesk Inventor. Prace wymagały ścisłej integracji projektu płytki drukowanej z elementami mechaniki robota. Płytka PCB robota poza zapewnieniem połączeń elektrycznych między poszczególnymi elementami jest również platformą, do której zamontowane są wszystkie podzespoły robota.
Jednostką sterującą robota jest mikrokontroler STM32L431RBT6. W układzie napędowym wykorzystano silniki Pololu HP z przekładnią 10:1 zapewniają dobry balans między momentem obrotowym i prędkością obrotową dla lekkiej konstrukcji robota. Nominalna prędkość wynosi 3000 obr/min, moment obrotowy 0,029 Nm. W robocie zastosowano wariant silnika z obustronnym wałem, pozwalającym na założenie enkodera magnetycznego, który dokonuje pomiaru z dokładnością 12 impulsów na obrót. Nominalne napięcie zasilania wynoszące 6V pozwala na zasilenie silników z dwuogniwowego pakietu Li-Pol [3]. Elementy robota wykonano m.in. w technologii druku 3D.



*Rysunek 3. Robot linefollower: a) model CAD; b) obiekt rzeczywisty*

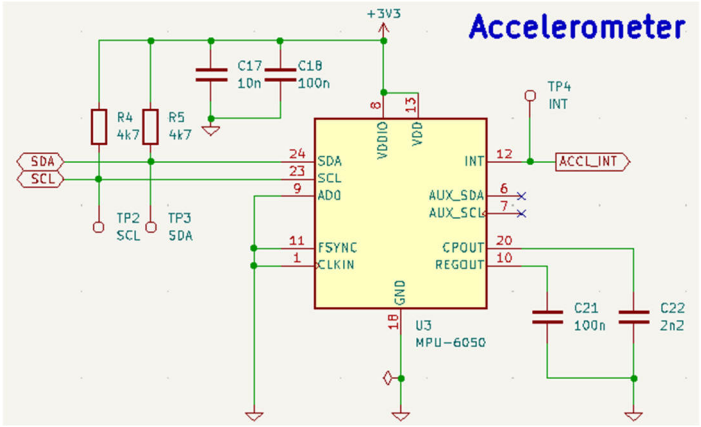
Płyta główna robota została zaprojektowana na czterech warstwach. Warstwa pierwsza i czwarta są warstwami sygnałowymi, warstwa druga i trzecia zapewniają zasilanie 3.3V do wszystkich komponentów. Wybór takiego układu warstw był

uwarunkowany dużą gęstością wyprowadzeń układu STM32L431RBT; dokonanie wszystkich połączeń na płytce dwuwarstwowej byłoby niemożliwe.



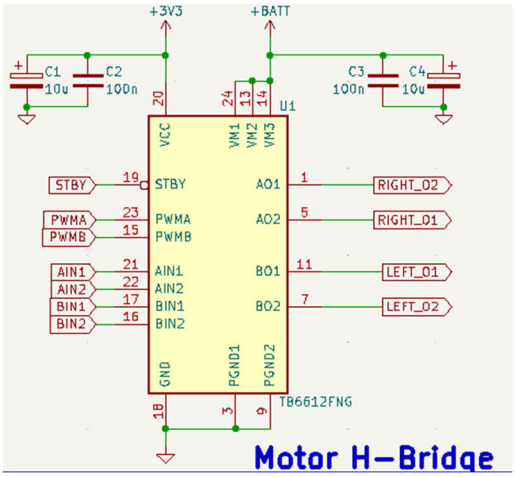

*Rysunek 4. Schemat połączeń układu STM32L431RBT*

Kondensatory C5-C10 odpowiadają za blokowanie zakłóceń zasilania układu [13]. Na płytce umieszczone są blisko układu, przy parach wyprowadzeń VSS VDD. Zasilanie części analogowej układu posiada osobny filtr złożony z dławika ferrytowego FB1 i kondensatorów C12 i C13.Za źródło pulsu zegara odpowiada rezonator kwarcowy Y1 wraz z kondensatorami. Do wyprowadzenia PB0 doprowadzono napięcie z dzielnika napięcia, którego zadaniem jest obniżenie napięcia baterii do poziomu, który może być zmierzony przez mikrokontroler.



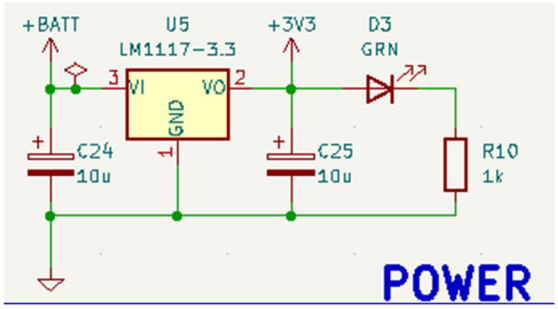

*Rysunek 5. Schemat połączeń układu MPU 6050*

Układ akcelerometru został wykonany wedle zaleceń producenta zawartych w dokumentacji. Kondensatory C17 i C18 odpowiadają za filtrację zasilania. Ważnymi elementami są rezystory R4 i R5 odpowiadające za podciąganie linii magistrali I2C do poziomu zasilania. Są one wymagane przez standard komunikacji I2C, każdy nadajnik na magistrali jest typu otwartego kolektora lub otwartego drenu [12].



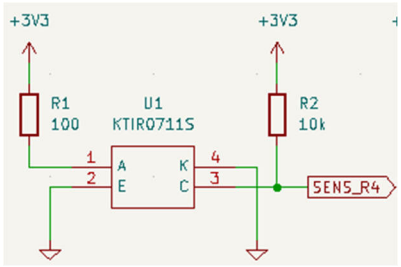

*Rysunek 6. Schemat połączeń układu TB6612FNG*

Układ posiada też drugą magistralę I2C służącą do komunikacji ze wskazanym przez producenta magnetometrem, jednak nie została ona wykorzystana. Sterownik silników nie potrzebuje żadnych elementów wspierających, poza kondensatorami filtrującymi zasilanie logiki oraz zasilanie z baterii. Sygnały sterujące podłączone są bezpośrednio z mikrokontrolera. Wyjścia układu połączone są ze złączami silnika.




*Rysunek 7. Sekcja zasilania*

Napięcie baterii jest za wysokie, żeby zasilić układy logiczne zawarte w projekcie. Do obniżenia napięcia do 3.3V zastosowano regulator liniowy LM1117 wraz z zalecanymi przez producenta kondensatorami tantalowymi. W celach diagnostycznych w projekcie zamieszczono diodę świecącą D3, wskazującą stan zasilania. Do rozłączenia zasilania zastosowano zworkę, ze względu na mniejszy rozmiar i możliwość rozłączania większego prądu niż dostępne przełączniki.




*Rysunek 8. Schemat połączeń czujnika odbiciowego*

Każdy z czujników odbiciowych ma połączone ze sobą dwa rezystory. Pierwszy z nich odpowiada za ograniczenie prądu przepływającego przez diodę podczerwoną. Drugi jest częścią układu pozwalającą na zmierzenie odpowiedzi fototranzystora. Po pierwszych testach całego robota zmieniono wartość rezystancji rezystorów diod podczerwonych z 100Ω na 470Ω. Przed zmianą łączny prąd pobierany przez diody wynosił blisko 200 mA, co znacząco obciążało regulator liniowy, powodując jego nadmierne nagrzewanie. Żeby móc wykorzystać w pełni możliwości świecenia diod zawartych w czujnikach należałoby rozważyć zastosowanie osobnego zasilania dla nich, lub wykorzystać bardziej wydajne rozwiązanie na obniżenie napięcia z baterii, np. przetwornicy.

## 4. Optymalizacja toru ruchu

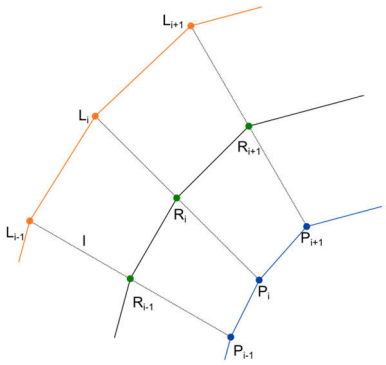
Optymalizacja jest problemem polegającym na znalezieniu ekstremum zadanej funkcji celu. Formułowanie zadań optymalizacji można podzielić na trzy etapy:
- budowa modelu matematycznego zawierającego zmienne decyzyjne dla zadania,
- przyjęcie ograniczeń zadania,
- przyjęcie funkcji celu [14].

W przypadku zagadnienia optymalizacji toru ruchu robota, zmienne decyzyjne zawarte będą w parametryzacji położenia poszczególnych punktów trasy. Przyjęte ograniczenia wynikają z konstrukcji robota oraz z regulaminu zawodów. Badane funkcje celu, długość oraz krzywizna toru, zostały wybrane jako czynniki które mogą mieć największy wpływ na czas pokonania trasy.
Podstawowym podziałem zagadnień optymalizacji jest podział ze względu na charakter zmiennych decyzyjnych. Rozróżniamy optymalizacje statyczną (zmienne są parametrami modelu matematycznego) i dynamiczną (zmienne zadania są funkcjami jednej zmiennej, na przykład czasu).
Aby przeprowadzić optymalizację toru ruchu robota należy sformułować ograniczenia dla nowo wyznaczonych punktów trasy. Według regulaminu zawodów, robot może opuścić trasę, lecz sędzia może unieważnić przejazd, jeśli opuszczenie trasy dało robotowi przewagę [1]. Zoptymalizowana trasa musi być ułożona w taki sposób, aby zarys robota nie wyjeżdżał poza linię trasy referencyjnej. Każdy z punktów trasy może być przesunięty maksymalnie o szerokość robota w lewo lub prawo od linii referencyjnej.
Wprowadzono dodatkową parametryzację, tak aby umożliwić manipulację punktem trasy za pomocą jednej zmiennej. Punkty prawego i lewego ograniczenia połączono

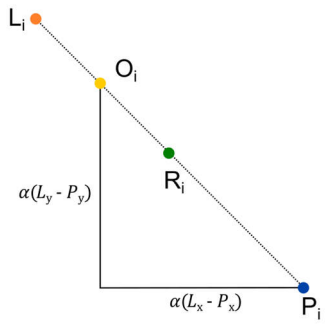
linią. Parametr α ustala pozycje punktu trasy optymalnej. Gdy α = 0 nowy punkt trasy pokrywa się z prawym ograniczeniem, gdy α = 1 punkt pokrywa się z lewym ograniczeniem. Zmieniając α w przedziale ⟨0; 1⟩ można otrzymać wszystkie możliwe położenia punktu trasy, spełniające założone ograniczenia.

$$\begin{cases} O_x = P_x + \alpha\left(L_x - P_x\right) \\ O_y = P_y + \alpha\left(L_y - P_y\right) \end{cases} \tag{1}$$

Takie przygotowanie trasy pozwala na sformułowanie n-wymiarowych zadań optymalizacji, gdzie n jest liczbą punktów trasy, a zmienną optymalizowaną jest wektor $\bar{\alpha}$. Dzięki temu możliwe było zastosowanie gotowych bibliotek do optymalizacji funkcji metodami numerycznymi. Optymalizację przeprowadzono za pomocą narzędzi bibliotek scipy i numpy języka Python.




*Rysunek 9. Konstrukcja ograniczeń optymalizowanej trasy. Lewe i prawe ograniczenie znajdują się w odległości szerokości robota od linii referencyjnej*




*Rysunek 10. Parametryzacja punktu trasy*

### 4.1. Najkrótsza droga

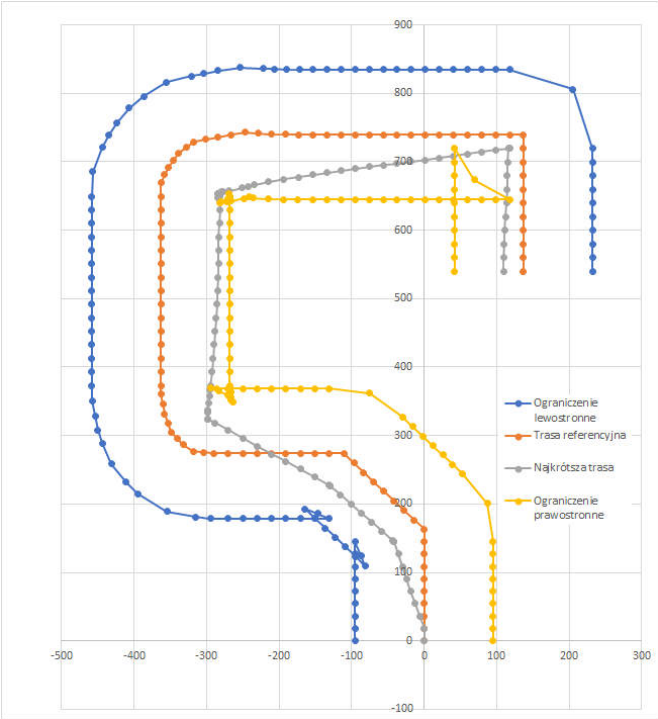
Pierwszą metodą uzyskania lepszego czasu na przejeździe robota, którą postanowiono sprawdzić, jest zminimalizowanie długości trasy. Dzięki temu podejściu wszystkie zakręty znajdujące się na trasie zostaną pokonane po wewnętrznej stronie. Przygotowane ograniczenia zapewnią, że robot pozostanie na trasie na całej długości przejazdu. Aby otrzymać funkcję celu do minimalizacji należy wyznaczyć długość trasy. Długość odcinka można wyznaczyć w następujący sposób:

$$s = \sqrt{\left(O_{x_{i+1}} - O_{x_i}\right)^2 + \left(O_{y_{i+1}} - O_{y_i}\right)^2} \tag{2}$$

Sumując długości poszczególnych odcinków otrzymano funkcję celu.

$$S(\bar{\alpha}) = \sum_{i=1}^{n-1} \sqrt{\left(O_{x_{i+1}} - O_{x_i}\right)^2 + \left(O_{y_{i+1}} - O_{y_i}\right)^2} \tag{3}$$

Parametry $\alpha_i$ są jedynymi zmiennymi funkcji, na ich podstawie wyznaczane są współrzędne punktów $O$ według równania (1).
Długość przykładowej trasy przedstawionej na rysunku 11 została zredukowana o 15% względem długości trasy referencyjnej. Jednak jak widać efektem ubocznym minimalizacji jest powstanie ostrych kątów na każdym z zakrętów, które potencjalnie mogą bardziej zwolnić robota, mimo skróconej trasy.




*Rysunek 11. Minimalizacja długości trasy*

### 4.2. Minimalna krzywizna toru

Drugim podejściem do problemu optymalizacji toru ruchu jest minimalizacja krzywizny toru. Ma to na celu rozwiązanie problemu, który wyniknął w trakcie minimalizacji długości trasy, czyli ostrych zakrętów. Im łagodniejsze są zakręty na całej trasie, tym mniej prędkości wytraci robot. Żeby wyznaczyć krzywiznę toru, jej przebieg musi być wyznaczony gładką krzywą. Dokonano interpolacji punktów toru za pomocą parametrycznych funkcji sklejanych trzeciego rzędu. Współczynniki wielomianu wyznaczane są na podstawie punktów odcinka oraz kierunków odcinka bieżącego i następnego.

$$\begin{cases} x(t) = a_0^x + a_1^x t + a_2^x t^2 + a_3^x t^3 \\ y(t) = a_0^y + a_1^y t + a_2^y t^2 + a_3^y t^3 \\ t \in \langle 0;1 \rangle \end{cases} \tag{4}$$

Wyznaczono pierwszą i drugą pochodną równań parametrycznych. Równania te potrzebne są do określenia współczynników interpolacji, jak i też wartości krzywizny funkcji.

$$\begin{cases} x'(t) = a_1^x + 2a_2^x t + 3a_3^x t^2 \\ y'(t) = a_1^y + 2a_2^y t + 3a_3^y t^2 \\ t \in \langle 0;1 \rangle \end{cases} \tag{5}$$

$$\begin{cases} x''(t) = 2a_2^x + 6a_3^x t \\ y''(t) = 2a_2^y + 6a_3^y t \\ t \in \langle 0;1 \rangle \end{cases} \tag{6}$$

Ułożono układy równań w celu określenia współczynników wielomianu na podstawie informacji zawartych w punktach trasy.

$$\begin{cases} x(0) = a_0^x = O_{x_i} \\ x(1) = a_0^x + a_1^x + a_2^x + a_3^x = O_{x_{i+1}} \\ x'(0) = a_1^x = \left(O_{x_{i+2}} - O_{x_i}\right) / \left\|O_i O_{i+1}\right\| \\ x'(1) = a_1^x + 2a_2^x + 3a_3^x = \left(O_{x_{i+2}} - O_{x_{i+1}}\right) / \left\|O_{i+1} O_{i+2}\right\| \end{cases} \tag{7}$$

$$\begin{cases} y(0) = a_0^y = O_{y_i} \\ y(1) = a_0^y + a_1^y + a_2^y + a_3^y = O_{y_{i+1}} \\ y'(0) = a_1^y = \left(O_{y_{i+2}} - O_{y_i}\right) / \left\|O_i O_{i+1}\right\| \\ y'(1) = a_1^y + 2a_2^y + 3a_3^y = \left(O_{y_{i+2}} - O_{y_{i+1}}\right) / \left\|O_{i+1} O_{i+2}\right\| \end{cases} \tag{7}$$

Układy równań przekształcono w równania macierzowe.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_0^x \\ a_1^x \\ a_2^x \\ a_3^x \end{bmatrix} = \begin{bmatrix} x(0) \\ x(1) \\ x'(0) \\ x'(1) \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_0^y \\ a_1^y \\ a_2^y \\ a_3^y \end{bmatrix} = \begin{bmatrix} y(0) \\ y(1) \\ y'(0) \\ y'(1) \end{bmatrix} \tag{9}$$

Po odwróceniu macierzy i przekształceniu równań wyznaczono współczynniki wielomianu interpolującego.

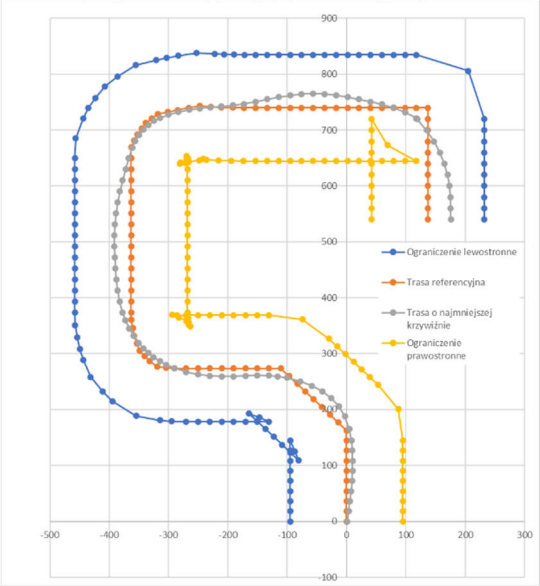
$$\begin{bmatrix} a_0^x \\ a_1^x \\ a_2^x \\ a_3^x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x'(0) \\ x'(1) \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} a_0^y \\ a_1^y \\ a_2^y \\ a_3^y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ y'(0) \\ y'(1) \end{bmatrix} \tag{11}$$

Wzór na krzywiznę funkcji parametrycznej wyraża się wzorem [4]:

$$\kappa(t) = \sqrt{x''(t)^2 + y''(t)^2} \tag{12}$$

Na potrzeby optymalizacji, krzywiznę wyznaczono w połowie krzywej interpolującej dla każdego odcinka, czyli dla $t = 0{,}5$. Funkcja celu wyrażona jest jako suma krzywizn wszystkich krzywych interpolujących trasę (Rys. 12).




*Rysunek 12. Minimalizacja krzywizny trasy*

### 4.3. Skrypt optymalizacji

Wszystkie kroki niezbędne do przygotowania toru do optymalizacji zostały zawarte w jednym skrypcie. Skrypt ten zawiera również dwie funkcje celu, według których trasa jest optymalizowana.

```
optimize.py
import sys
```

```
import csv
import pyglet
import math
import numpy as np
import scipy.optimize as optimize

L = 190

def track_length(alpha, *args):
    rightBound, deltaX, deltaY = args
    n = len(deltaX)
    S = 0
    alpha[0] = 0.5
    alpha[1] = 0.5

    for i in range(1, n):
        deltaPx = rightBound[i].x - rightBound[i - 1].x +
alpha[i]                                                 *
                  deltaX[i] - alpha[i - 1] * deltaX[i - 1]
        deltaPy = rightBound[i].y - rightBound[i - 1].y +
alpha[i]                                                 *
                  deltaY[i] - alpha[i - 1] * deltaY[i - 1]
        S = S + math.sqrt(deltaPx ** 2 + deltaPy ** 2)

    return S

def track_curvature(alpha, *args):
    rightBound, deltaX, deltaY = args
    n = len(deltaX)
    kappa = 0
    A = np.array([[1, 0, 0, 0],
                  [1, 1, 1, 1],
                  [0, 1, 0, 0],
                  [0, 1, 2, 3]])

    invA = np.linalg.inv(A)

    for i in range(n-2):
        x0 = rightBound[i].x + alpha[i] * deltaX[i]
        y0 = rightBound[i].y + alpha[i] * deltaY[i]
        x1 = rightBound[i+1].x + alpha[i+1] * deltaX[i+1]
        y1 = rightBound[i+1].y + alpha[i+1] * deltaY[i+1]
        x2 = rightBound[i+2].x + alpha[i+2] * deltaX[i+2]
        y2 = rightBound[i+2].y + alpha[i+2] * deltaY[i+2]

        l0 = math.sqrt((x1 - x0) ** 2 + (y1 - y0) ** 2)
        l1 = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

        dx0 = (x1-x0) / l0
        dy0 = (y1-y0) / l0
        dx1 = (x2-x1) / l1
        dy1 = (y2-y1) / l1

        x_coef = invA.dot(np.vstack([x0, x1, dx0, dx1]))
        x_coef = np.transpose(x_coef)[0]
```

```
        y_coef = invA.dot(np.vstack([y0, y1, dy0, dy1]))
        y_coef = np.transpose(y_coef)[0]

        t = 0.5

        x_pp = 2 * x_coef[2] + 6 * x_coef[3] * t

        y_pp = 2 * y_coef[2] + 6 * y_coef[3] * t

        k = x_pp**2 + y_pp**2

        kappa = kappa + k

    return kappa

if __name__ == '__main__':
    track_points = []
    with open(sys.argv[1], 'r') as track_file:
        fieldnames = ['x', 'y']
        reader = csv.DictReader(track_file, delimiter='\t',
quotechar='"',
                                fieldnames=fieldnames,

quoting=csv.QUOTE_NONNUMERIC)
        line: dict
        for line in reader:
            track_points.append(pyglet.math.Vec2(line['x'],
-line['y']))

    track_points.insert(0, track_points[0])
    track_points.append(track_points[-1])

    leftBound = []
    rightBound = []

    deltaX = []
    deltaY = []

    for i in range(1, len(track_points) - 1):
        direction = track_points[i + 1] - track_points[i - 1]
        direction = direction.rotate(math.pi / 2)
        direction = direction.normalize()
        left = track_points[i] + direction * L / 2
        right = track_points[i] - direction * L / 2
        leftBound.append(left)
        rightBound.append(right)

        deltaX.append(left.x - right.x)
        deltaY.append(left.y - right.y)

    n = len(deltaX)
    print(track_length([0.5]*n, rightBound, deltaX, deltaY))
```

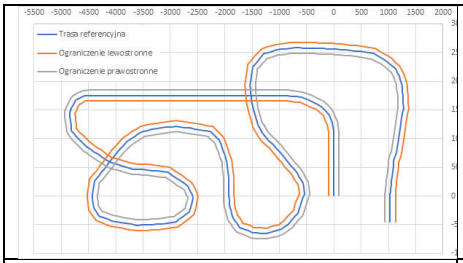
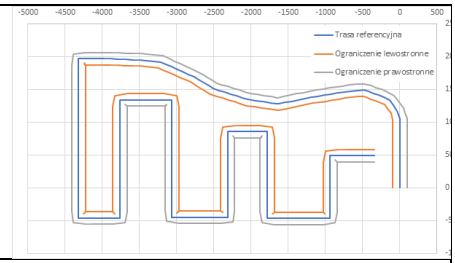
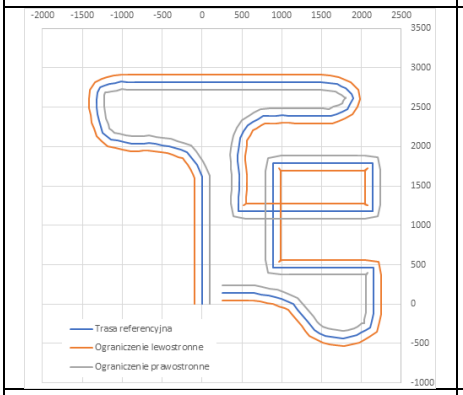
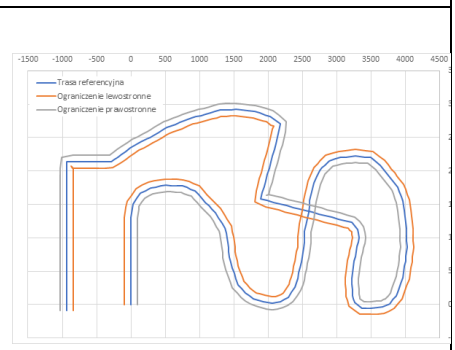
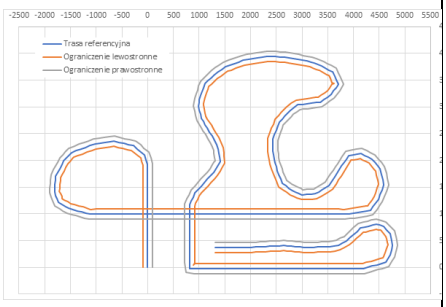
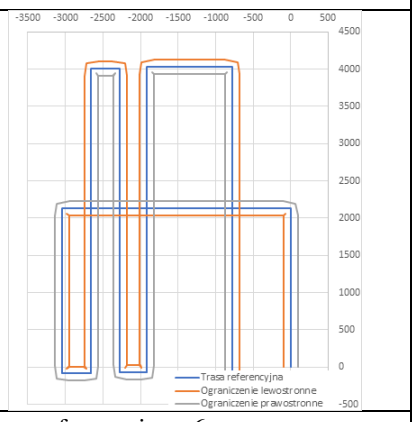
```
    result        =        optimize.minimize(track_curvature,
np.random.random(n),
                                bounds=optimize.Bounds(0, 1),
                                args=(rightBound,     deltaX,
deltaY),
                                options={'maxfun':100000000,
'ftol':0.00001,
                                'disp':True})
    print(result.message)
    print(result.x)
    alpha = result.x

    with open(sys.argv[2], 'w', newline='') as curv_file:
        writer   =   csv.writer(curv_file,   delimiter='\t',
quotechar='"',
                            quoting=csv.QUOTE_NONNUMERIC)
        for i in range(len(leftBound)):

writer.writerow([rightBound[i].x+alpha[i]*deltaX[i],

rightBound[i].y+alpha[i]*deltaY[i]])

    result         =         optimize.minimize(track_length,
np.random.random(n),
                                bounds=optimize.Bounds(0, 1),
                                args=(rightBound,     deltaX,
deltaY),

options={'maxfun':100000000,'disp': True})
    print(result.message)
    print(result.x)
    alpha = result.x

    with open(sys.argv[3], 'w', newline='') as len_file:
        writer   =   csv.writer(len_file,   delimiter='\t',
quotechar='"',
                            quoting=csv.QUOTE_NONNUMERIC)
        for i in range(len(leftBound)):

writer.writerow([rightBound[i].x+alpha[i]*deltaX[i],

rightBound[i].y+alpha[i]*deltaY[i]])
```

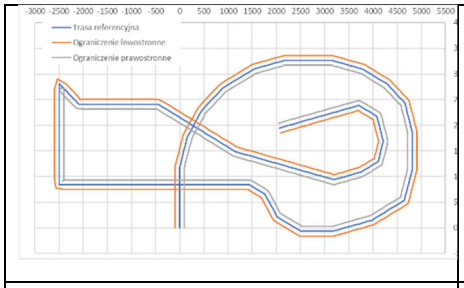
Funkcje celu zostały zaimplementowane w funkcjach track_length i track_curvature. Główny blok funkcji importuje punkty ze wskazanego poprzez argument linii poleceń pliku csv. Następnie wyznaczane są ograniczenia lewo i prawostronne. Za algorytmy optymalizacji odpowiada biblioteka scipy.optimize. Funkcja minimize pozwala na minimalizacje dowolnej funkcji, zwracającej wartość skalarną. Domyślną metodą dla zadań z ograniczoną przestrzenią argumentów jest algorytm L-BFGS-B. Algorytm wyznacza numerycznie gradient funkcji, a następnie na jego podstawie przeprowadza minimalizację [5]. Do algorytmu przekazywane są parametry maxfun, Jako punkt startowy optymalizacji przyjęto wektor o losowych składowych z przedziału $\langle 0; 1 \rangle$.

## 5. Badania doświadczalne

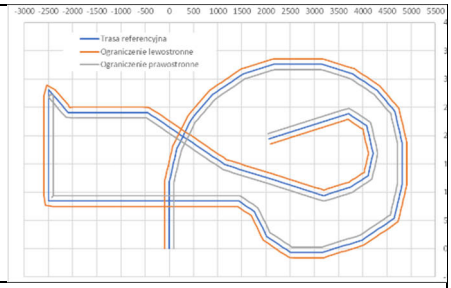
Optymalizacje dwiema metodami wykonano dla dziesięciu tras, następnie wykonano symulacje przejazdów robota (Tabela 1).

*Tabela 1. Zestawienie tras referencyjnych robota*

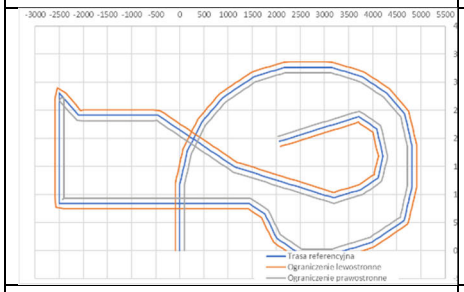
| | |
|---|---|
|  |  |
| Trasa referencyjna - 1 | Trasa referencyjna - 2 |
|  |  |
| Trasa referencyjna - 3 | Trasa referencyjna - 4 |
|  |  |
| Trasa referencyjna - 5 | Trasa referencyjna - 6 |



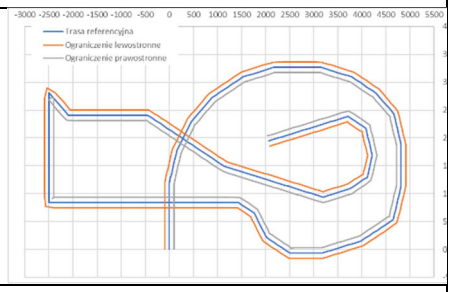

Tr□□□r□f□r□□□yj□□- 7




Tr□□□r□f□r□□□yj□□- 8

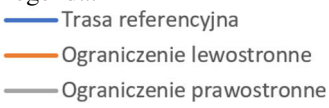



Tr□□□r□f□r□□□yj□□- 9




Tr□□□r□f□r□□□yj□□- 10

Legenda:




Zestawienie wszystkich czasów przejazdów podano w Tabeli nr 2, natomiast zmiany czasu przejazdów względem trasy referencyjnej w Tabeli nr 3.

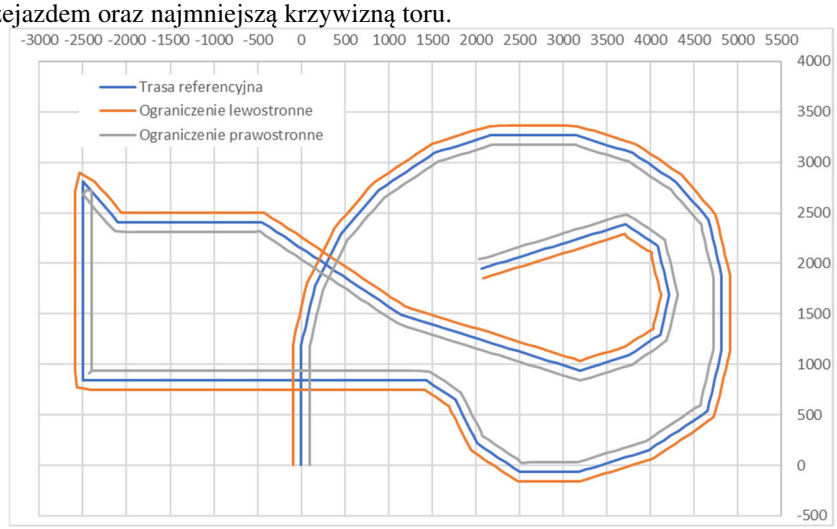
*Tabela 2. Zestawienie czasów przejazdów*

| **Tra□a** | **R□f□r□ncyjna, □** | **Najkrót□za, □** | **Najmni□j□za krzywizna, □** |
|---|---|---|---|
| 1 | 17.770 | 16.950 | 14.966 |
| 2 | 19.007 | 14.204 | 10.565 |
| 3 | 18.790 | 13.740 | 10.425 |
| 4 | 19.416 | 13.761 | 11.917 |
| 5 | 32.883 | 21.604 | 19.927 |
| 6 | 24.823 | 21.950 | 16.705 |
| 7 | 42.120 | 38.853 | 22.833 |
| 8 | 51.766 | 42.017 | 42.551 |
| 9 | 62.211 | 52.620 | 50.125 |
| 10 | 38.033 | 32.800 | 21.133 |

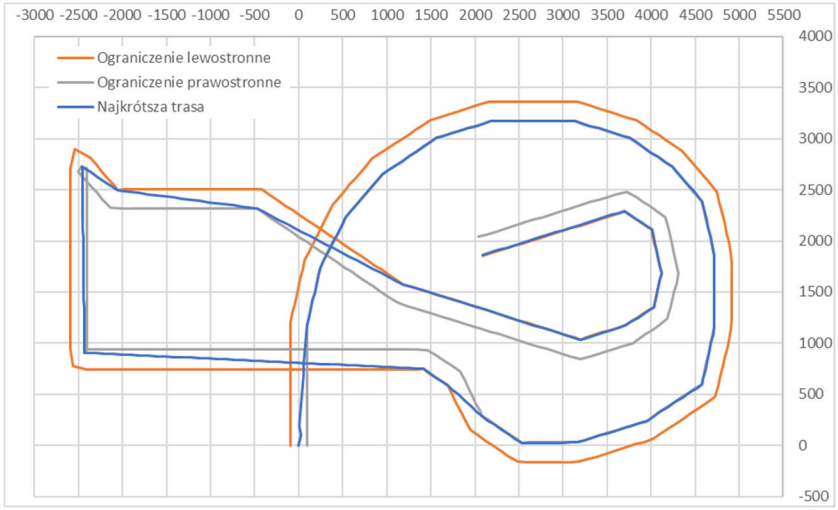
*Tabela 3. Poprawa czasu przejazdu względem trasy referencyjnej*

| Trasa | Referencyjna, s | Najkrótsza | Najmniejsza krzywizna |
|---|---|---|---|
| 1 | 17.770 | 5% | 16% |
| 2 | 19.007 | 25% | 44% |
| 3 | 18.790 | 27% | 45% |
| 4 | 19.416 | 29% | 39% |
| 5 | 32.883 | 34% | 39% |
| 6 | 24.823 | 12% | 33% |
| 7 | 42.120 | 8% | 46% |
| 8 | 51.766 | 19% | 18% |
| 9 | 62.211 | 15% | 19% |
| 10 | 38.033 | 14% | 44% |

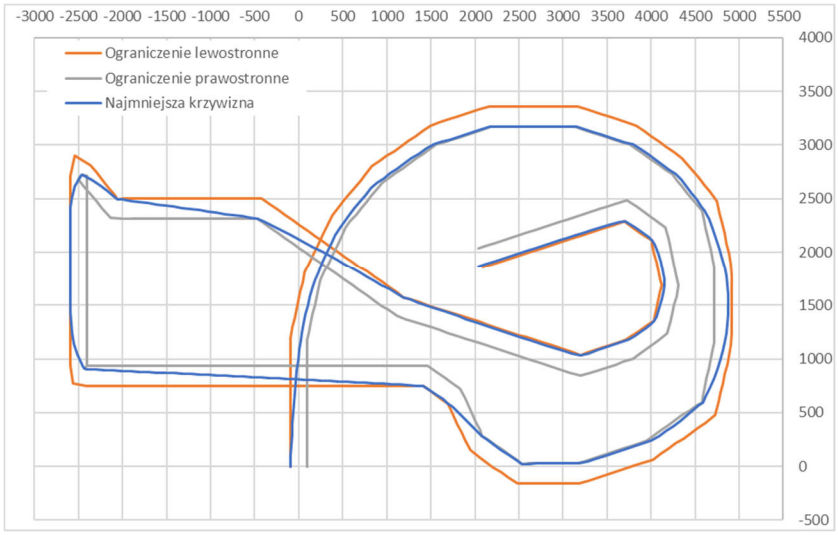
Na rys. 13 – 15 przedstawiono przykładową trasę referencyjną, z najszybszym przejazdem oraz najmniejszą krzywizną toru.




*Rysunek 13. Przykładowa trasa referencyjna nr 10*




*Rysunek 14. Najkrótsza trasa nr*




*Rysunek 15. Najmniejsza krzywizna nr 10*

## 5. Podsumowanie

Celem pracy było opracowanie metod optymalizacji toru ruchu robota typu linefollower. Przedstawiono wybór zastosowanych algorytmów, opisano implementację symulacji oraz zaprezentowano prototyp konstrukcji.
Najlepszą metodą optymalizacji toru ruchu dla większości tras okazała się minimalizacja krzywizny toru ruchu. Zmniejszenie krzywizny umożliwia robotowi osiągnięcie większych prędkości. Podejście to ułatwia też pokonanie zakrętów o kącie 90°, których było wiele w testowanych trasach.

Dla jednej z dziesięciu tras nieznacznie szybszym podejściem była minimalizacja długości trasy. Kształt trasy, jak i sam algorytm wykonywany przez robota może wpłynąć na to, która z metod będzie najlepsza.
Optymalizacja krzywizny toru ruchu okazała się być czasochłonną metodą. Otrzymanie wyniku z domyślną dokładnością stosowanych bibliotek zajmowało setki iteracji, a każda z iteracji potrafiła trwać około sekundy. W celu szybkiego otrzymania wyników zmniejszono tolerowany błąd względny wyniku do 0.00001. Możliwe, że przez to algorytm znajduje minimum lokalne funkcji celu, rozwiązanie nie jest torem o najmniejszej możliwej krzywiźnie.
Do implementacji algorytmu podążającego za optymalnym torem na fizycznym prototypie robota potrzebne byłyby zmiany w jego konstrukcji. Brak wystarczającej ilości miejsca do przechowania punktów trasy oraz niska rozdzielczość zastosowanych enkoderów uniemożliwiają przeniesienie testów z symulacji.

## LITERATURA


1. Serwis internetowy - Regulamin konkurencji LineFollower ROBOmotion. *https://xchallenge.pl/regulations/ROBOmotion_2023_LineFollower.pdf* - data odczytu 09.06.2023).
2. CASSANOVA D.: On minimum time vehicle manoeuvring: the theoretical optimal time. Uniwersytet Cranfeld, 2000.
3. Serwis internetowy - Silnik Pololu HP 10:1. Dokumentacja techniczna. *https://www.pololu.com/product/2211* - data odczytu 09.05.2023.
4. KENNEDY J.: The Arc Length Parametrization of a Curve. 2011. *https://web.archive.org/web/20150928030020/https://sites.google.com/site/johnkennedyshome/home/class-downloads* - data odczytu 07.06.2023.
5. Serwis internetowy - Dokumentacja biblioteki scipy. *https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#rdd2e1855725e-6* - data odczytu 10.06.2023.
6. Serwis internetowy - Autonomiczny robot mobilny. *https://elearning.przemyslprzyszlosci.gov.pl/slownik-pojec/autonomiczny-robot-mobilny/* - data odczytu 16.06.2023.
7. Serwis internetowy - Automatyczny czy autonomiczny? Roboty mobilne bez tajemnic. https://elearning.przemyslprzyszlosci.gov.pl/9306-2/ (data odczytu 16.06.2023).
8. Serwis internetowy - Stanford Cart. *https://cyberneticzoo.com/cyberneticanimals/1960-stanford-cart-american/* data odczytu 16.06.2023.
9. MORAVEC H.P.: Stanford cart and the A.I. Robot. Artificial Intelligence, 8(1977)2, 157-182.
10. NILSSON N. J.: Mobile robot systems: the Stanford cart and the CMU rover. AI Magazine, 5(1984)2, 45-52.

11. Serwis internetowy - Turbina EDF 27. *https://kamami.pl/napedy-tunelowe-edf/204387-turbina-edf-27-z-silnikiem-bldc-10000kv.html* - data odczytu 16.06.2023.
12. MIELCZAREK W.: Szeregowe interfejsy cyfrowe, Gliwice: Helion, 1993.
13. LANCASTER D.: TTL Cookbook', Howard W. Sams, 1975.
14. STADNICKI J.: Teoria i praktyka rozwiązywania zadań optymalizacji z przykładami zastosowań technicznych, Warszawa: WNT, 2006.
15. NOCEDAL J., WRIGHT S. J.: Numerical Optimization. Springer Science & Business Media 2006.
16. PYTKA D.: Optymalizacja toru ruchu robota typu linefollower, praca magisterska, Bielsko-Biała 2023.