POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Robotyka (ARR)

PRACA DYPLOMOWA MAGISTERSKA

Badania porównawcze algorytmów śledzenia trasy dla robotów klasy linefollower

Comparison of line tracking methods for linefollower robots

AUTOR:
Witold Lipieta

PROWADZĄCY PRACĘ:
Dr inż. Robert Muszyński

OCENA PRACY:

WROCŁAW 2017

Spis treści

1 Wstęp **3**

2 Specyfikacja trasy **5**
2.1 Reprezentacja przebiegu trasy 6
2.2 Trasy użyte do badań 6

3 Algorytmy śledzenia trasy dla robotów klasy linefollower **9**
3.1 Algorytm optymalizujący trajektorię na podstawie mapy trasy 9
3.2 Algorytmy śledzenia ścieżki 11
3.2.1 Prosty automat skończony 11
3.2.2 Algorytm PD 12
3.2.3 Algorytm PD z kontrolą rzeczywistej prędkości obrotowej kół 13
3.3 Algorytm z nadrzędnym profilerem prędkości bazującym na mapie trasy . . 13
3.4 Własności algorytmów sterowania 16

4 Robot mobilny klasy linefollower **19**
4.1 Konstrukcja mechaniczna 19
4.2 Napęd 20
4.3 Czujniki 21
4.3.1 Wykrywanie linii 21
4.3.2 Estymacja położenia robota 22
4.4 Komunikacja 23

5 Badanie własności algorytmów **25**
5.1 Badanie charakterystyki napędów 25
5.2 Badanie maksymalnych przyspieszeń 26
5.3 Badanie modułu korygującego estymację położenia 28
5.4 Badanie porównawcze algorytmów 29
5.5 Porównanie z innymi konstrukcjami 32
5.5.1 XIV Festiwal Robotyki Cyberbot 2017 w Poznaniu 34
5.5.2 Trójmiejski Turniej Robotów 2017 w Gdańsku 37

6 Podsumowanie **41**

Bibliografia **42**

Rozdział 1

Wstęp

W ostatnich latach widzimy postępujący wzrost popularności robotyki na całym świecie. Przejawia się to również poprzez rosnącą liczbę zawodów skierowanych do amatorów robotyki. Jedną z najbardziej popularnych kategorii na takich zawodach są wyścigi robotów klasy linefollower, w której autonomiczny robot mobilny musi pokonać trasę, wyznaczoną przez kontrastującą z podłożem linię, w jak najkrótszym czasie.

Zasady obowiązujące w kategorii linefollower są ściśle doprecyzowane [3, 4]. Przede wszystkim ograniczone są wymiary robota do wielkości mieszczącej się na kartce papieru formatu A4, przy nieograniczonej wysokości robota i jego masie. Określone są także parametry techniczne trasy jaką należy pokonać i jej dopuszczalne elementy składowe, takie jak skrzyżowania czy ostre zakręty. Zgodnie z regulaminem konkurencji linefollower, robot nie może ominąć żadnego elementu trasy oraz podczas wykonywania przejazdu musi cały czas pozostawać swoim obrysem nad linią, a w przypadku całkowitego wypadnięcia poza trasę powinien powrócić do tego samego miejsca.

Podstawowe elementy składowe robota klasy linefollower to napęd umożliwiający poruszanie się oraz zestaw czujników, pozwalających na określenie położenia linii. Obecnie konstrukcje tej kategorii są wysoce wyspecjalizowane, aby jak najlepiej wykonywać postawione przed nimi zadanie. Robot mobilny klasy (2,0) zapewnia odpowiednią zwrotność oraz proste sterowanie, dlatego roboty tej klasy są zdecydowanie najpopularniejszym rozwiązaniem w kategorii linefollower.

Można zaobserwować, iż obecnie wysiłki konstruktorów skupione są przede wszystkim na dopracowywaniu aspektów mechanicznych robotów [25]. Użyciu odpowiedniego materiału do wykonania opon, aby zapewnić jak najlepszą przyczepność do podłoża, co pozwala na pokonywanie ostrych zakrętów z dużą prędkością. Zmniejszeniu wagi robota, dzięki czemu maleją działające na niego w trakcie ruchu siły bezwładności. Jednakże, ponieważ tego typu działania są ostatnio jedynymi podejmowanymi, od kilku lat można zauważyć pewną stagnację w rozwoju robotów klasy linefollower. Roboty zoptymalizowane pod względem konstrukcji mechanicznej zaczęły osiągać zbliżone wyniki (różnice w czasach przejazdów pomiędzy najlepszymi robotami na zawodach liczone są często w setnych sekundy). Zjawisko to jest spowodowane brakiem rozwoju algorytmów śledzenia linii — większość zawodników korzysta z tych samych rozwiązań, bazujących na prostych regulatorach PD [14, 7], które uwzględniają jedynie informacje o kształcie małej części trasy wykrywanej aktualnie przez czujniki. Wynika stąd potrzeba usystematyzowania algorytmów umożliwiających śledzenie trasy z dużą prędkością oraz opracowania rozwiązań poszerzających możliwości robotów tej klasy.

Celem pracy jest porównanie istniejących metod śledzenia trasy dla robotów klasy linefollower i zaproponowanie rozwiązania pozwalającego na poprawę uzyskiwanych re-

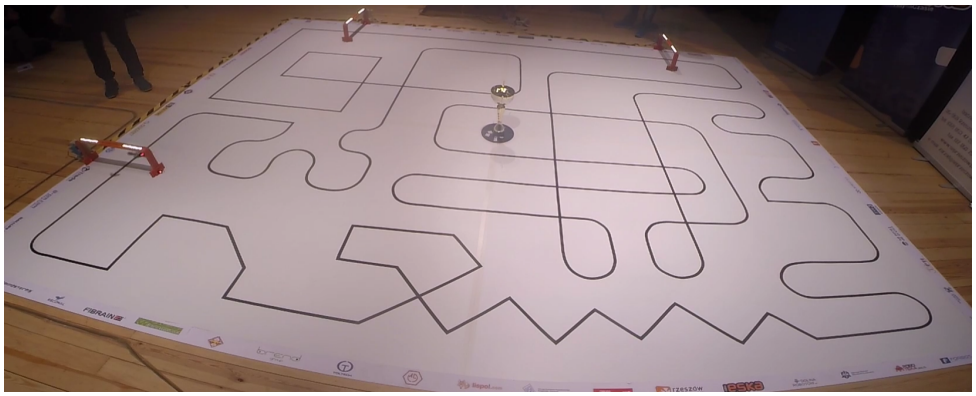
zultatów. Aby to osiągnąć należy zaimplementować na rzeczywistym robocie mobilnym wybrane algorytmy sterowania oraz zbadać ich własności w przygotowanym środowisku testowym.

Praca składa się z sześciu rozdziałów. W rozdziale 2 przybliżono specyfikację tras w konkurencji linefollower oraz metody ich reprezentacji. W rozdziale 3 opisano algorytmy śledzenia tras oraz ich właściwości. Rozdział 4 poświęcono opisowi robota używanego podczas testów. Natomiast w rozdziale 5 przedstawiono wyniki przeprowadzonych badań porównawczych. Rezultaty pracy oraz wnioski zawarto w rozdziale 6.

Rozdział 2

Specyfikacja trasy

Aby rozpocząć badania nad zadaniem postawionym przed robotami klasy linefollower, czyli śledzeniem trasy, należy najpierw doprecyzować jak może ona wyglądać. Zgodnie z zasadami konkurencji linefollower [3, 4], trasa przejazdu jest wyznaczona przez czarną linię o szerokości około 19 mm umieszczoną na jasnym tle. Zwykle jest to zrealizowane za pomocą czarnej taśmy izolacyjnej przyklejonej na kilku, połączonych ze sobą, białych płytach MDF. Przykładowa trasa została przedstawiona na rysunku 2.1.



Rysunek 2.1: Trasa finałowa w kategorii linefollower na zawodach ROBO~motion 2016 w Rzeszowie

Jak wspomniano we wstępie, zasady kategorii linefollower precyzują dozwolony kształt tras, który pomimo dowolności przebiegu, musi spełniać kilka ograniczeń. Najczęściej spotykane są trasy zamknięte, gdzie start oraz meta są w tym samym miejscu. Linia wytyczająca trasę musi być ciągła, nie są dozwolone żadne przerwy ani rozwidlenia. Na trasie mogą pojawiać się skrzyżowania pod kątem prostym, które należy pokonać przejeżdżając na wprost, jednak poza nimi, elementy trasy nie mogą znajdować się bliżej siebie niż 210 mm. Taka specyfika trasy pozwala kłaść mniejszy nacisk na dokładne śledzenie trasy, np. podczas pokonywania ostrych zakrętów przy dużych prędkościach, można pozwolić sobie na wypadnięcie robota poza trasę, bez obawy że wjedzie on na zupełnie inny odcinek trasy i zacznie go śledzić zamiast powrócić na właściwy tor. Dodatkowo ograniczenie to wpływa na charakterystykę zakrętów, ponieważ ich minimalny promień skrętu musi być większy niż 10 cm. Ponadto, większość regulaminów zawodów robotycznych dopuszcza możliwość pojawienia się na trasie zakrętów o promieniu 0 mm, przy czym maksymalny kąt skrętu wynosi wtedy zwykle 90 stopni — linie tworzą ze sobą kąt prosty. Trasa pokazana na rysunku 2.1 zawiera wszystkie te elementy: skrzyżowania, kąty proste oraz 180-stopniowe zakręty o promieniu skrętu ok. 10 cm.

2.1 Reprezentacja przebiegu trasy

Do działania zaawansowanych algorytmów, analizujących globalnie przebieg trasy, istnieje potrzeba jego właściwego reprezentowania. Reprezentacja trasy w formie uszeregowanego zbioru punktów opisanych za pomocą współrzędnych kartezjańskich $(x, y)^T$ jest intuicyjna dla ludzi i dlatego przydaje się do weryfikacji poprawności przeprowadzonych obliczeń oraz danych zbieranych przez robota. Nie jest ona natomiast zbyt przydatna do obliczania optymalnego profilu prędkości przejazdu, opisanego w podrozdziale 3.3. Dlatego do tego celu wykorzystano reprezentację przebiegu trasy jako krzywizny trasy $k(s)$ w funkcji odległości s [27]. Poniżej zaprezentowano przekształcenia wymagane do budowy obu reprezentacji na podstawie pomiarów wykonywanych przez robota.

Ruch robota mobilnego klasy (2,0), wyposażonego w przyrostowe czujniki ruchu, można jednoznacznie opisać za pomocą translacji t oraz rotacji r

$$t = \frac{\Delta l_r + \Delta l_l}{2}, \tag{2.1}$$

$$r = \frac{\Delta l_r - \Delta l_l}{b}, \tag{2.2}$$

gdzie Δl_r i Δl_l to zmiany odległości przejechanej przez kolejno prawe i lewe koło, a b to odległość między kołami. Znając model kinematyki robota klasy (2,0) [23] oraz zakładając początkowe ustawienie robota w przestrzeni $(x_0, y_0, \theta_0) = (0, 0, 0)$, możemy wyznaczyć iteracyjny wzór na obliczanie pozycji robota w czasie ruchu, a co za tym idzie, reprezentacji przebiegu trasy przez niego pokonanej jako zbioru punktów (x, y), w postaci

$$\begin{cases} x_i = x_{i-1} + t_i \cos\theta_i \\ y_i = y_{i-1} + t_i \sin\theta_i \\ \theta_i = \theta_{i-1} + r_i \end{cases}, \tag{2.3}$$

gdzie x_i, y_i to współrzędne w iteracji i, θ_i to orientacja robota w iteracji i, natomiast t_i i r_i to translacja i rotacja w i-tej iteracji.

Korzystając z translacji i rotacji, możliwe jest również wyznaczenie przebiegu trasy pokonywanej przez robota za pomocą jej krzywizny $k(s)$ w funkcji odległości s [19]

$$k_i = \frac{r_i}{t_i}, \tag{2.4}$$

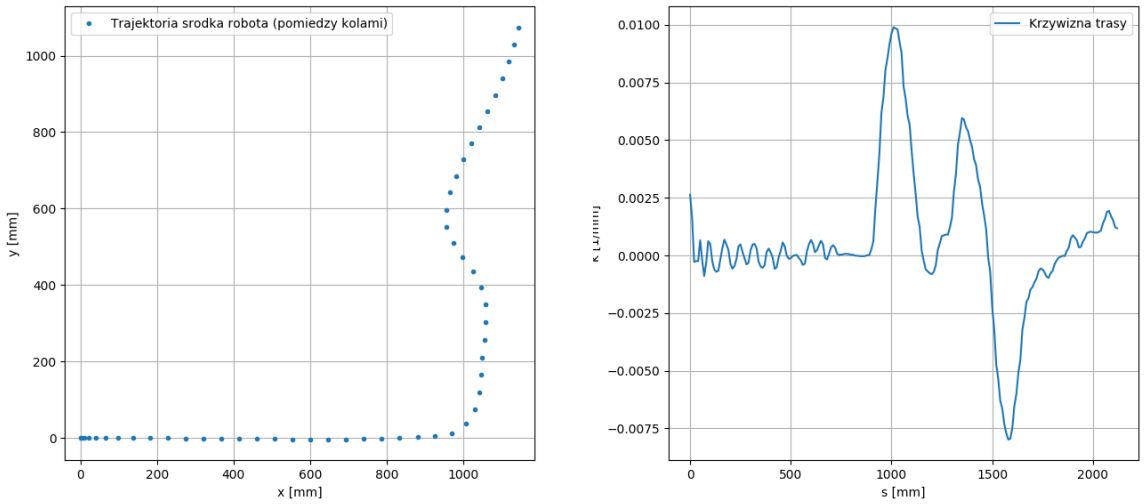
$$s_i = s_{i-1} + t_i, \tag{2.5}$$

gdzie k_i to krzywizna w iteracji i, s_i to odległość przebyta do iteracji i, przy czym $s_0 = 0$, a t_i i r_i to translacja i rotacja w i-tej iteracji.

Mając do dyspozycji dane zebrane podczas przejazdu, możliwe jest zatem stworzenie reprezentacji trasy zarówno w postaci punktów (x, y) jak i przebiegu jej krzywizny. Przykładowy fragment trasy został przedstawiony w obu reprezentacjach na rysunku 2.2. Należy zauważyć, że powyższe formuły prowadzą do uzyskania przebiegu trasy pokonywanej przez środek robota, umiejscowiony pomiędzy jego kołami, a nie przebiegu samej, śledzonej linii.

2.2 Trasy użyte do badań

Do przeprowadzenia badań przygotowano trzy trasy zawierające charakterystyczne elementy typowe dla konkurencji linefollower. Zostały one zaprojektowane tak, aby pozwalały

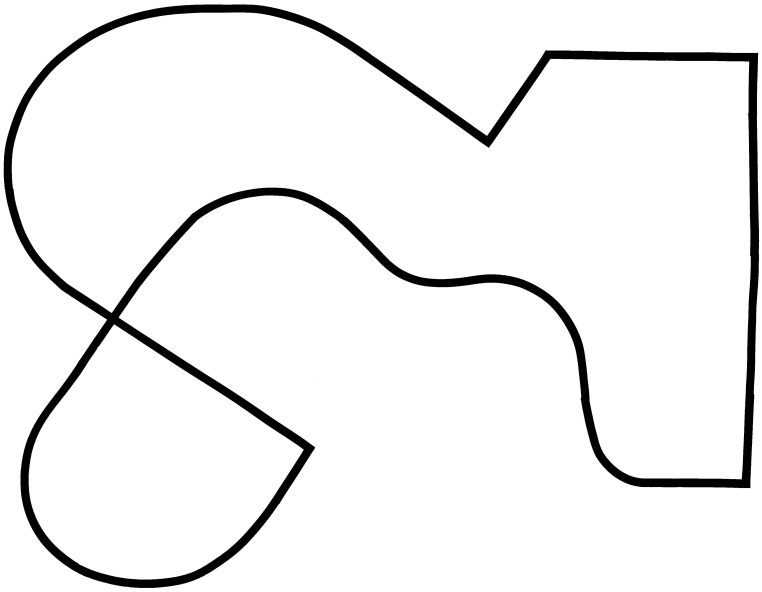


(a) Trasa w postaci zbioru punktów (x, y) (b) Trasa w postaci $k(s)$

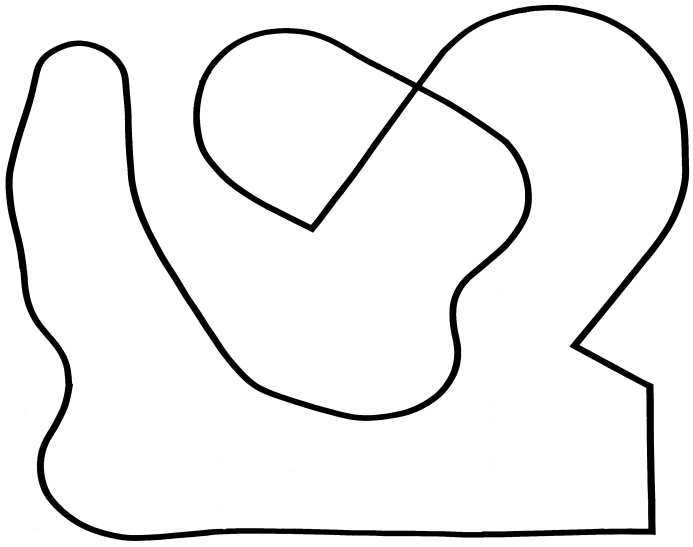
Rysunek 2.2: Porównanie dwóch metod reprezentacji przebiegu trasy

Rysunek 2.3: Pierwsza trasa testowa

na zbadanie różnych właściwości zaimplementowanych algorytmów. Trasy przedstawiono na rysunkach 2.3, 2.4 oraz 2.5. Pierwsza z tras, o długości 757 cm, składa się z długich odcinków prostych zakończonych nawrotami, z jednej strony jest to ciasny zakręt ok. 180 stopni o promieniu 10 cm a z drugiej dwa następujące po sobie kąty proste. Druga trasa ma 684 cm długości i składa się z wielu ciasnych zakrętów oraz skrzyżowania. Ostatnia trasa testowa, która ma 1017 cm długości, stanowi połączenie dwóch pierwszych. Użyte do badań trasy reprezentują dwa typy spotykane najczęściej w konkurencji linefollower, czyli trasy kręte i wąskie, na których osiągana prędkość jest niska, oraz trasy o dużej ilości długich odcinków prostych, które połączone są ze sobą pojedynczymi zakrętami.



Rysunek 2.4: Druga trasa testowa



Rysunek 2.5: Trzecia trasa testowa

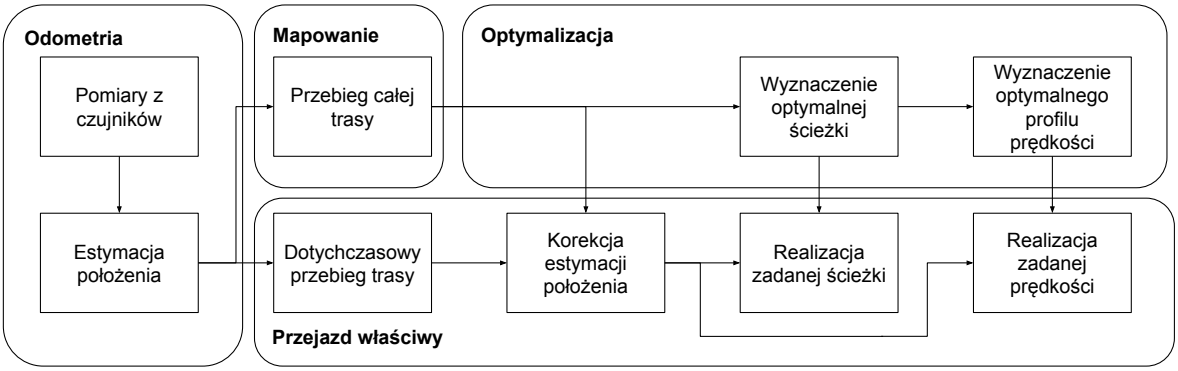
Rozdział 3

Algorytmy śledzenia trasy dla robotów klasy linefollower

W rozdziale opisano różne sposoby podejścia do realizacji zadania śledzenia trasy przez roboty klasy linefollower. Najpierw rozważono koncepcję algorytmu globalnie optymalizującego trajektorię na podstawie zebranej wcześniej mapy trasy [19]. Następnie opisano najczęściej stosowane algorytmy śledzenia ścieżki, które zostały zaimplementowane do badań. W dalszej części przedstawiono opracowany w ramach pracy algorytm z nadrzędnym sterownikiem prędkości bazującym na przebiegu trasy. W celu uzyskania kompletności, rozdział zakończono podsumowaniem zalet i wad wszystkich zaimplementowanych algorytmów, które zaobserwowano podczas badań.

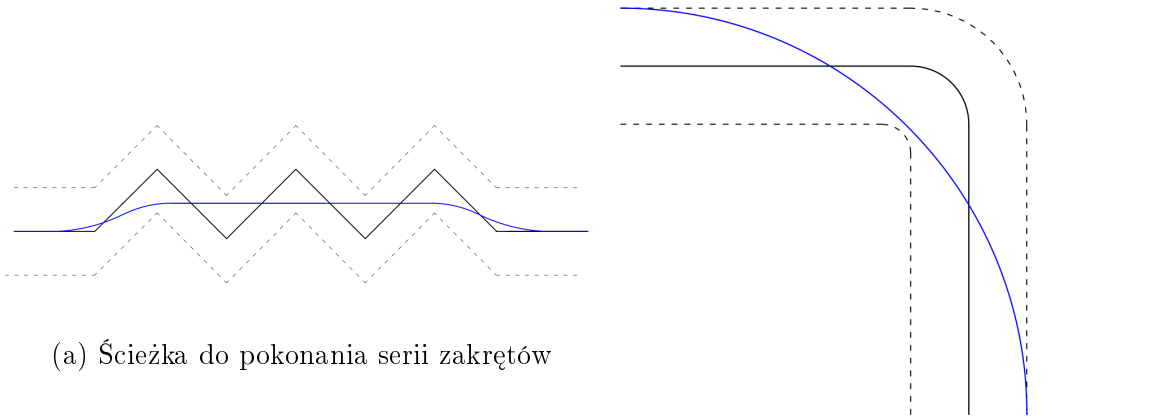
3.1 Algorytm optymalizujący trajektorię na podstawie mapy trasy

Rozważmy kompleksowy algorytm śledzenia trasy przez roboty typu linefollower, którego schemat pokazano na rysunku 3.1. Idea algorytmu opiera się na założeniu, że na zawo-



Rysunek 3.1: Schemat działania algorytmu optymalizującego trajektorię

dach jest możliwość przejechania danej trasy więcej niż jeden raz. Można więc wykorzystać informacje zdobyte podczas poprzednich przejazdów do optymalizacji kolejnych prób. Algorytm ten jest wieloetapowy. W pierwszym etapie robot powinien określić dokładny przebieg trasy, np. poprzez powolny przejazd podczas którego, na podstawie danych odczytanych z czujników, będzie budować mapę trasy. Niska prędkość podczas takiego przejazdu



(a) Ścieżka do pokonania serii zakrętów

(b) Ścieżka do pokonania ostrego zakrętu

Rysunek 3.2: Przykłady lokalnie optymalnych ścieżek

jest podyktowana potrzebą zminimalizowania poślizgów robota, co wpływa znacząco na jakość utworzonej mapy. Mapa ta pozwala na optymalizację ruchu robota w kolejnych etapach, co powinno przełożyć się na krótsze czasy przejazdów.

Drugim krokiem jest wyznaczenie ścieżki, po której robot pokona daną trasę. Na tym etapie korzystamy z faktu, że nie ma potrzeby aby linia wyznaczająca trasę znajdowała się cały czas po środku robota, ponieważ jak nadmieniono wcześniej, podczas przejazdu robot musi znajdować się nad nią jedynie swoim obrysem. Daje to pewne pole manewru, ponieważ odpowiednio optymalizując przebieg planowanej ścieżki ruchu [13], robot może np. ścinać pewne odcinki lub rozluźniać zakręty — wjeżdżać w zakręt od jego strony zewnętrznej, podczas pokonywania zakrętu zbliżyć się do jego szczytu a następnie wyjeżdżając ponownie wrócić blisko strony zewnętrznej [26]. Przykłady ścieżek umożliwiających ścinanie serii zakrętów i optymalne pokonanie zakrętu zostały przedstawione na rysunku 3.2. Zaznaczono na nich linię którą należy śledzić, obszar w jakim zgodnie z zasadami może znaleźć się środek robota (zależny od jego szerokości) oraz optymalną ścieżkę ruchu.

Takie postępowanie prowadzi jednak jedynie do lokalnej optymalizacji trasy [21] — łatwo wyobrazić sobie sytuację, w której rozluźnienie pierwszego z serii zakrętów powoduje w rezultacie wydłużenie ogólnego czasu przejazdu. Dlatego w ogólności trasę należy rozpatrywać globalnie [22], aby optymalizować czas całego przejazdu a nie jego pojedynczych elementów. Algorytmy globalnie optymalizujące trasę przejazdu są szeroko wykorzystywane np. w wyścigach Formuły 1, a wiele badań poświęcono sposobom wyznaczania tak zwanej „racing line" [9, 29], czyli najlepszej trajektorii do pokonania danej trasy. Jednym ze sposobów wyznaczania optymalnej trasy jest algorytm MCP (ang. *Minimal Curvature Path*) [10], który polega na globalnym minimalizowaniu krzywizny trasy, co przekłada się na zwiększenie maksymalnej prędkości z jaką można ją pokonać.

Po wyznaczeniu ścieżki ruchu należy obliczyć optymalny profil prędkości [27], tak aby robot przyspieszał na prostych odcinkach i zwalniał tuż przed zakrętami. Prędkość musi być dobrana w taki sposób, żeby robot cały czas wykorzystywał w pełni swoje możliwości konstrukcyjne. Wyznaczenie takiego profilu jest możliwe przy użyciu informacji o krzywiźnie trasy i współczynniku tarcia opon, ponieważ na ich podstawie można obliczyć prędkość, jaką robot może osiągnąć w danym punkcie trasy.

Dopiero po przejściu przez wszystkie wymienione etapy robot może przystąpić do samego przejazdu. Algorytm musi mieć możliwość zrealizowania zadanego wcześniej ruchu, co wymaga precyzyjnej kontroli nad silnikami i synchronizacji sygnałów ze wszystkich

dostępnych czujników. By optymalny przejazd był możliwy, robot powinien być w stanie określić swoją pozycję podczas ruchu. Ze względu na błędy wynikające z niepewności pomiarowej czujników oraz poślizgów estymowane położenie robota różni się od rzeczywistego. Istnieje zatem potrzeba przeprowadzania korekcji położenia np. na podstawie czujników linii — przebieg trasy może być punktem odniesienia. Korekcja ta musi być wykonywana na bieżąco podczas przejazdu lub doraźnie po każdym skomplikowanym fragmencie trasy, tak aby błąd estymacji położenia nie zdążył się skumulować.

Pomimo, że opisany algorytm wydaje się być intuicyjny, a wiele z wymienionych problemów zostało rozwiązanych, nie jest on stosowany. Spowodowane jest to trudną implementacją poszczególnych etapów na rzeczywistym robocie. Większość badań jest opartych jedynie na komputerowych symulacjach, zakładających pewne uproszczenia, których nie można przyjąć w praktyce. Również rozpatrywany w pracach ruch robotów realizowany jest zwykle z niską prędkością, która w konkurencji linefollower powinna być jak najwyższa. Dlatego w budowanych konstrukcjach stosowane są prostsze algorytmy [14], realizujące śledzenie trasy tylko na podstawie aktualnych pomiarów z czujników. Przykłady takich algorytmów zostaną przedstawione w kolejnym podrozdziale.

3.2 Algorytmy śledzenia ścieżki

Poniżej zebrano algorytmy śledzenia trasy stosowane typowo w robotach klasy linefollower. Wszystkie te algorytmy bazują na 3 podstawowych etapach umożliwiających podążanie za linią. Najpierw uzyskiwane są dane z dostępnych czujników. Następnie na ich podstawie wyliczany jest błąd położenia robota względem linii. Ostatecznie podawany jest na silniki taki sygnał sterujący, aby zminimalizować ten błąd. Wykonywanie tych operacji w trakcie jazdy z odpowiednią częstotliwością pozwala na śledzenie trasy wytyczonej przez linię.

3.2.1 Prosty automat skończony

Najprostszym sposobem na śledzenie trasy jest napisanie prostego automatu skończonego definiującego zachowanie robota podczas jazdy. Algorytm ten zbudowany jest z serii instrukcji warunkowych, które obejmują różne konfiguracje, w jakich linia może zostać wykryta przez czujniki i na tej podstawie wykonujących daną akcję. Pseudokod prezentujący przykładową implementację algorytmu dla trzech czujników linii został przedstawiony na wydruku 3.1.

Wydruk 3.1: Automat skończony dla trzech czujników

```
if czujnik_srodkowy then
        jedz_prosto
if czujnik_lewy then
        skrec_w_lewo
if czujnik_prawy then
        skrec_w_prawo
if not ( czujnik_srodkowy or czujnik_lewy or czujnik_prawy ) then
        skrec_w_lewo
```

Tego typu algorytm działa dobrze dla małej liczby sensorów, np. w konkurencji linefollower dla robotów zbudowanych przy użyciu klocków LEGO, gdzie stosowane są zwykle

nie więcej niż trzy czujniki linii. Mała liczba czujników oznacza jednak niską dokładność oszacowania położenia robota względem linii, przez co musi być ograniczona jego prędkość.

W tym rozwiązaniu liczba wymaganych instrukcji znacząco wzrasta wraz z liczbą zastosowanych czujników — musi obejmować większość konfiguracji w jakich śledzona linia może być wykryta przez czujniki, a na zachowanie robota wpływa również kolejność w jakiej sprawdzane są poszczególne konfiguracje. Ze względu na obszerność kodu dla większej liczby czujników, algorytm jest nieintuicyjny oraz trudny w późniejszym dostrajaniu. Aby wyeliminować te wady, algorytm ten można rozwinąć i zaimplementować w formie regulatora proporcjonalnego, którego rozbudowana forma została dokładniej opisana w następnym podrozdziale. Regulator proporcjonalny charakteryzuje się jednak oscylacjami, które mogą wpłynąć na jakość przejazdu.

3.2.2 Algorytm PD

Do dokładnego śledzenia trasy, a co za tym idzie możliwości pokonania jej z większą prędkością, potrzeba większej liczby czujników. Efektywne przetwarzanie dużej ilości pochodzących z nich danych jest możliwe przy użyciu regulatora proporcjonalno-różniczkującego. W tym algorytmie błąd położenia obliczany jest jako średnia ważona z odczytów wszystkich czujników linii dostępnych na robocie

$$e = \frac{\sum_{i=1}^{n} \omega_i x_i}{\sum_{i=1}^{n} x_i}, \tag{3.1}$$

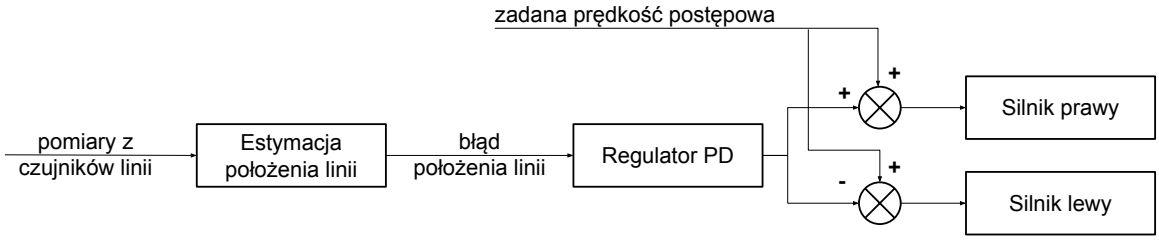
gdzie n oznacza liczbę czujników linii, x_i to pomiar z i-tego czujnika linii

$$x_i = \begin{cases} 1 & \text{jeśli linia wykryta przez czujnik } i\text{-ty} \\ 0 & \text{w przeciwnym wypadku} \end{cases}, \tag{3.2}$$

natomiast $[\omega_1, \omega_2, ..., \omega_n]$ to wagi poszczególnych czujników. Wagi te skorelowane są z odległością czujnika od środka — każda z nich odzwierciedla błąd położenia jaki jest przyjmowany przez robota, gdy jedynie dany czujnik wykrywa linię. Do podstawowej wersji modułu liczącego błąd położenia dodaje się również dodatkowe warunki, biorące pod uwagę sytuację, gdy linia nie jest wykrywana przez żaden z czujników. Zwykle bierze się wtedy pod uwagę błąd obliczony w poprzedniej iteracji algorytmu, co zazwyczaj pozwala robotowi powrócić na trasę. Dobór odpowiednich wag jest istotnym elementem testowania takich algorytmów, wpływają one bowiem na reakcje robota na zmieniający się kształt śledzonej linii.

Błąd wyliczony według wzoru (3.1) jest przekazywany do regulatora PD, który ma za zadanie wyznaczenie odpowiedniego sygnału PWM, podawanego na silniki. Wyjście regulatora, sumowane jest z sygnałem odpowiadającym zadanej wartości prędkości postępowej robota. W tym algorytmie zadana prędkość jest stała i ustawiana *a priori*, nic nie stoi jednak na przeszkodzie aby zmieniać ją w trakcie jazdy, co zostało wykorzystane w algorytmie opisanym w podrozdziale 3.3. Suma tych sygnałów jest następnie przekazywana do modułu sterującego prędkością obrotową silników. Proces ten został przedstawiony na rysunku 3.3. Zastosowanie członu różniczkującego w regulatorze ogranicza oscylacje robota podczas przejazdu. Do działania regulator PD używa jedynie aktualnie wyliczonego błędu, z czego wynika jego podstawowe ograniczenie — średnia prędkość robota musi być ograniczona do wartości pozwalającej na przejechanie najtrudniejszego etapu trasy.

Ze względu na niezawodność i łatwą implementację jest to zdecydowanie najczęściej stosowany algorytm śledzenia linii wśród robotów klasy linefollower [7].



Rysunek 3.3: Schemat działania algorytmu PD do śledzenia linii

3.2.3 Algorytm PD z kontrolą rzeczywistej prędkości obrotowej kół

Algorytm opisywany w poprzednim podrozdziale nie steruje rzeczywistą prędkością silników, co wpływa na jakość sterowania, szczególnie na czas reakcji na dynamiczne zmiany prędkości. Mając do dyspozycji czujniki umożliwiające pomiar prędkości obrotowej silników, można dodać kolejny regulator pracujący w zamkniętej pętli sprzężenia zwrotnego. Jako sprzężenie zwrotne stosowane są najczęściej enkodery magnetyczne, mierzące przesunięcie magnesów zamontowanych na kołach lub bezpośrednio na wale silnika.

W tej wersji algorytmu, regulator położenia robota względem linii nie podaje sygnału sterującego bezpośrednio na silniki, a na dodatkowy regulator, który zapewnia ustawienie zadanej prędkości na każdym z kół. Dzięki temu możemy dokładniej kontrolować ruch robota, np. określając przyspieszenia z jakimi ma się poruszać, czy pozwalać na szybszą reakcję robota na gwałtowne zakręty.

3.3 Algorytm z nadrzędnym profilerem prędkości bazującym na mapie trasy

Jak zauważono we wstępie, istnieje potrzeba rozwoju algorytmów dla robotów klasy linefollower. W niniejszej pracy zaproponowano zatem algorytm, który jest pewnym przybliżeniem kompleksowego rozwiązania, przedstawionego w podrozdziale 3.1. Jego procedura polega na zmierzeniu przebiegu trasy podczas pierwszego przejazdu, a następnie wykorzystaniu zebranych informacji do wyznaczenia optymalnego profilu prędkości, który można wykorzystać jako wejście do dodatkowego nadrzędnego sterownika, uruchamianego w czasie kolejnych prób. Prowadzi to do lepszego wykorzystania potencjału robota a w rezultacie skrócenia czasu przejazdu. Większość z przeprowadzanych w algorytmie operacji wymaga dużej ilości skomplikowanych obliczeń, jednak może być wykonywana off-line, dlatego zdecydowano się na zaprogramowanie części funkcjonalności na komputerze, który jest zsynchronizowany z robotem mobilnym. Sposób komunikacji komputer–robot opisano w podrozdziale 4.4. Kolejne kroki algorytmu wyglądają następująco.

Utworzenie mapy trasy

Pierwszy etap algorytmu, czyli budowa mapy trasy, odbywa się przy małej prędkości. Robot przejeżdża nieznaną trasę przy użyciu konwencjonalnego algorytmu PD z kontrolą rzeczywistej prędkości obrotowej silników, zbierając przy tym dane ze wszystkich sensorów. Zebrane dane są następnie filtrowane i przetwarzane na reprezentację przebiegu trasy w postaci jej krzywizny, która została opisana w podrozdziale 2.1.

Wyznaczanie optymalnej prędkości

Wybór formy $k(s)$ do reprezentacji przebiegu trasy jest podyktowany łatwością obliczania na jej podstawie pożądanego profilu prędkości [8, 19, 28]. Mając przebieg $k(s)$, oraz znając przyspieszenia z jakimi może poruszać się robot, możemy wyznaczyć maksymalną dopuszczalną prędkość w każdym punkcie trasy. Metoda wyznaczania tej prędkości wykorzystuje założenie, że cały czas staramy się wykorzystać dostępne przyspieszenie, które wynika ze współczynnika tarcia kół, a więc:

$$\|\mu g\| = \|a_{lat} + a_{long}\|, \tag{3.3}$$

gdzie μ jest współczynnikiem tarcia, g wektorem grawitacji, $a_{lat} = v^2|k|$ oznacza wektor przyspieszenia dośrodkowego, natomiast $a_{long} = \frac{dv}{dt}$ to wektor przyspieszenia wzdłużnego. Na podstawie tej zależności chcemy uzyskać maksymalną, stałą możliwą prędkość poruszania się, możemy zatem założyć, że $a_{long} = 0$, z czego uzyskujemy

$$\mu g = v^2|k|, \tag{3.4}$$

Prędkość maksymalną można zatem wyrazić wzorem

$$v = \sqrt{\frac{\mu g}{|k|}}. \tag{3.5}$$

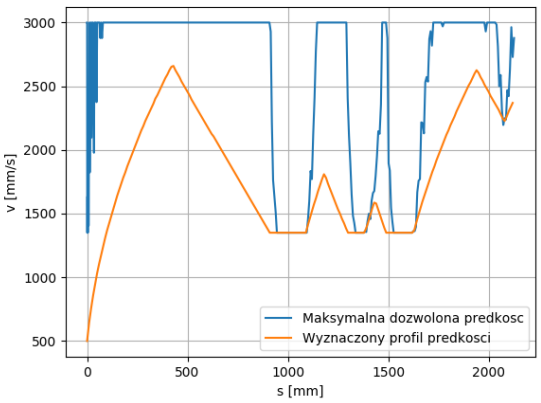
Aby uzyskać jak najmniejszą złożoność obliczeniową, w opracowanym algorytmie przyjęto pewne dodatkowe uproszczenia. Zamiast obliczać prędkość maksymalną dla każdego z maksimum i minimum lokalnego na przebiegu $k(s)$ trasy, a następnie wyznaczać zadany profil prędkości między tymi punktami za pomocą zależności możliwej do wyprowadzenia ze wzoru (3.3), przyjęto wartości obliczone ze wzoru (3.5) jako odgórne ograniczenie na całej trasie. Ponieważ dla prostych odcinków wyliczona ze wzoru prędkość dąży do nieskończoności, na wyznaczony przez nią profil nakładane jest ograniczenie w postaci prędkości maksymalnej, ustawianej jako jeden z parametrów algorytmu. Tak wyznaczony profil, nie jest jednak ciągły (prędkość zmienia się skokowo), a więc jest niemożliwy do uzyskania w praktyce. Przyjmując maksymalne wartości przyspieszeń, których sposób wyznaczenia opisano w podrozdziale 5.2, możemy wyeliminować ten problem i wyznaczyć pożądany profil przy użyciu wzoru na ruch jednostajnie przyspieszony

$$v(t) = v_0 + at, \tag{3.6}$$

gdzie v_0 oznacza prędkość w poprzednim analizowanym punkcie, a to aktualne przyspieszenie, t to czas ruchu od poprzedniego punktu. Ponieważ naszym celem jest minimalizacja tego czasu, należy przyjąć jak największe wartości przyspieszeń podczas planowania ruchu. Poza mniejszą złożonością obliczeniową, takie rozwiązanie pozwala nam na łatwiejszą konfigurację parametrów algorytmu, ponieważ możemy ustalać różne wartości przyspieszania i hamowania oraz modyfikować je wedle potrzeb. Przykładowy profil prędkości wraz z odgórnym ograniczeniem maksymalnej prędkości widoczne są na rysunku 3.4.

Przejazd z nadrzędnym sterownikiem

Uzyskany profil prędkości jest następnie przesyłany do robota, gdzie służy jako wejście do nadrzędnego sterownika prędkości zadanej. Taki regulator dodajemy do



Rysunek 3.4: Prędkość maksymalna i optymalny profil prędkości obliczone dla trasy z rysunku 2.2

układu sterowania w miejsce zadanej prędkości zadanej w zaprezentowanym na rysunku 3.3 schemacie. Sterownik ten na bieżąco estymuje pozycję robota na podstawie jego odometrii i pobiera z profilu wartość prędkości odpowiednią dla danej pozycji. Dzięki takiemu postępowaniu cały przejazd jest wykonywany z optymalną prędkością.

Korekcja estymacji położenia

Skuteczność algorytmu zależy od dokładnego oszacowania położenia robota w czasie przejazdu. Niestety metoda obliczania przejechanego dystansu, przedstawiona w podrozdziale 2.1, obarczona jest kumulującym się w czasie błędem, co jest szczególnie istotne dla bardzo długich tras. Ponadto, poza błędami wynikającymi z poślizgów i niepewności pomiarowej sensorów, estymacja położenia, bazująca na przejechanej dotychczas odległości, narażona jest również na błędy wynikające z innego sposobu pokonywania zakrętów przy różnych prędkościach. Brana pod uwagę krzywizna nie zależy bowiem od kształtu linii, a trasy pokonywanej przez środek robota umiejscowiony pomiędzy kołami. Kształt krzywizny może się zatem zmieniać w zależności od prędkości z jaką pokonywana jest trasa, co jest problemem w przypadku wykonywania szybkich przejazdów z użyciem mapy.

Wynika stąd potrzeba opracowania metody korekcji estymacji położenia. Zaimplementowany w algorytmie moduł korekcji działa w oparciu o metodę zwaną dopasowaniem do wzorca [11]. Znając mapę trasy, uzyskaną podczas pierwszego przejazdu, możemy w czasie szybkich przejazdów próbować znaleźć na niej fragmenty pasujące najlepiej do właśnie przejechanego odcinka trasy i na tej podstawie korygować swoją pozycję. Do znajdowania najlepszego dopasowania miedzy dwoma przebiegami zdecydowano się na wykorzystanie algorytmu SSD (ang. *Sum of squared differences*) [17], który można opisać za pomocą wzoru

$$SSD(\tau) = \sum_i (a(i) - b(i-\tau))^2, \quad x_s = \arg\min_\tau SSD(\tau) \tag{3.7}$$

gdzie a i b to porównywane sygnały, i oznacza numer próbki, natomiast τ to przesunięcie. Algorytm ten pozwala na znalezienie takiego przesunięcia x_s pomiędzy sygnałami, aby zminimalizować kwadrat błędu, wynikający z różnic między nimi. Umożliwia to wyszukanie na przebiegu $k(s)$ przesunięcia między pozycją rzeczywistą a estymowaną.

3.4 Własności algorytmów sterowania

Poniżej zebrano własności algorytmów opisanych we wcześniejszych częściach rozdziału, które zidentyfikowano na podstawie ich implementacji i przeprowadzonych badań. Podsumowanie to ma formę opisu zalet i wad charakteryzujących dane algorytmy.

1. Maszyna stanów

   Zalety:

   - mała liczba potrzebnych czujników,
   - bardzo łatwa w implementacji,
   - może być zaimplementowana w formie regulatora proporcjonalnego.

   Wady:

   - w najprostszej wersji bierze pod uwagę tylko aktualnie wykryte położenie linii,
   - wraz ze wzrostem liczby czujników, kod potrzebny do ich opisania jest dużo obszerniejszy,
   - niska prędkość przejazdu,
   - mała dokładność śledzenia, duże oscylacje,
   - nie radzi sobie z trudniejszymi trasami.

2. Regulator PD

   Zalety

   - idealnie nadaje się do obsługi większej liczby czujników,
   - stosunkowo prosty w implementacji, wymaga jedynie znajomości podstaw sterowania,
   - człon różniczkujący ogranicza oscylacje,
   - umożliwia osiąganie dużych prędkości,
   - poza czujnikami linii, nie potrzebuje żadnych innych czujników,
   - algorytm jest odporny na poślizgi kół.

   Wady

   - maksymalna prędkość robota, jest ograniczona do prędkości umożliwiającej pokonanie najtrudniejszego elementu trasy,
   - algorytm nie steruje rzeczywistą prędkością silników,
   - wymaga dużej ilości testów do wybrania optymalnych nastaw, które mogą się różnić w zależności od trasy,
   - aby osiągać duże prędkości, należy dodać dodatkowe warunki umożliwiające powrót na linię po wyjechaniu z trasy,
   - duże poślizgi kół podczas pokonywania zakrętów wydłużają czas przejazdu.

3. Regulator PD + regulator prędkości silników

   Zalety:

- wszystkie zalety zwykłego regulatora PD,
- kontrola rzeczywistej prędkości silników,
- pozwala efektywnie skręcać,
- szybciej osiąga ustaloną prędkość maksymalną po starcie i ostrym zakręcie.

Wady:

- robot musi być wyposażony w odpowiednie czujniki, np. enkodery,
- kod sterujący pracą robota musi być rozszerzony o funkcje odczytujące dane z czujników i przetwarzające je na użyteczną wartość,
- użyte w robocie silniki muszą być przetestowane, aby dobrać odpowiednie nastawy do algorytmu kontrolującego ich prędkość.

4. Regulator PD ze sprzężeniem zwrotnym z silników + nadrzędny sterownik prędkości robota bazujący na mapie trasy

   Zalety:

   - znając mapę trasy i aktualne położenie robota algorytm jest w stanie wyznaczyć optymalną prędkość z jaką można pokonać dany odcinek,
   - prędkość maksymalna jest ograniczona jedynie przez specyfikę odcinka trasy w którym aktualnie robot się znajduje, a nie przez jej najtrudniejszy fragment,
   - możliwa jest odpowiednia reakcja na nadchodzące zakręty, np. wcześniejsze hamowanie.

   Wady:

   - implementacja takiego algorytmu jest bardzo skomplikowana, potrzebne są dwa tryby pracy: mapowanie oraz przejazd z wykorzystaniem mapy,
   - przed szybkim przejazdem, robot musi przynajmniej raz przejechać daną trasę aby zbudować jej mapę, co jest sporym ograniczeniem w przypadku finałów zawodów robotycznych, podczas których można wykonać jedynie dwa lub trzy przejazdy,
   - należy na bieżąco obliczać aktualne położenie oraz określać prędkość z jaką należy się poruszać,
   - do określenia prędkości należy wyznaczyć współczynnik tarcia opon robota oraz dopuszczalnych wartości przyspieszania i hamowania,
   - skuteczność algorytmu znacząco zależy od prawidłowego określenia pozycji robota w czasie jazdy,
   - estymacja położenia jest narażona na błędy spowodowane poślizgami i niedokładnością czujników,
   - błąd położenia kumuluje się z czasem, co nakłada wymóg opracowania metody korekcji położenia,
   - aby zminimalizować poślizgi, prędkość na zakrętach musi być ograniczona co nie jest efektywne przy bardzo krętych trasach.

Rozdział 4

Robot mobilny klasy linefollower

Do przeprowadzenia eksperymentów w rzeczywistym środowisku testowym potrzebna jest gotowa konstrukcja robota, zdolna przejechać wyznaczoną trasę. Aby uzyskane wyniki były miarodajne wszystkie testy należy przeprowadzić na tym samym robocie. Zamieszczony tutaj opis robota, na którym zaimplementowano omawiane w rozdziale 3 algorytmy śledzenia trasy, może służyć za przewodnik po najważniejszych cechach szybkich robotów klasy linefollower.

Prezentowany robot, który jest typowym robotem mobilnym klasy (2,0), jest zbudowany zgodnie z zasadami konkurencji linefollower [3, 4], które ograniczają jego rozmiary do wielkości kartki A4 (około 30 cm na 20 cm, bez limitu wysokości). Wybór klasy (2,0), jako sposobu zamocowania napędów, jest podyktowany dużą zwrotnością takiej konstrukcji, możliwością osiągania dużych prędkości oraz prostego sterowania. Dzięki temu istnieje możliwość porównania czasów osiąganych przez zaimplementowane algorytmy z czasami najlepszych konstrukcji występujących na zawodach robotycznych w dokładnie tych samych warunkach, co zostało szczegółowo omówione w podrozdziale 5.5.

4.1 Konstrukcja mechaniczna

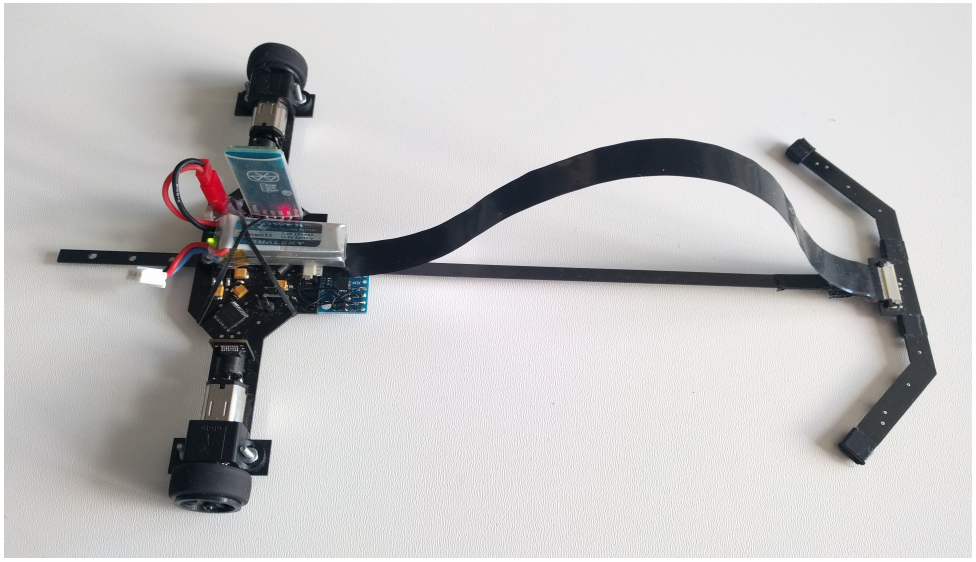
Konstrukcja mechaniczna wymaga sprecyzowania wielu aspektów, które trzeba wziąć pod uwagę przy projektowaniu robota klasy linefollower. Są to m.in.:

- waga robota i położenie środka ciężkości — robot musi być jak najlżejszy a jego środek ciężkości położony jak najniżej aby zminimalizować bezwładność i pozwolić na szybkie przyspieszanie/zwalnianie,

- rozstaw kół — wpływa na skrętność robota oraz jego przyczepność,

- liczba i ułożenie czujników linii — należy zapewnić ich wystarczającą liczbę do pokonania trasy każdego typu oraz odpowiednie ich wysunięcie, celem dostarczenia informacji o kształcie linii z pewnym wyprzedzeniem. Wysunięcie to jest jednak przydatne tylko przy zastosowaniu prostszych algorytmów. Dla algorytmów uwzględniających mapę trasy i optymalizujących jej trajektorię, jak opisany w podrozdziale 3.1, pożądane może być umiejscowienie czujników pomiędzy kołami co ułatwi określanie położenia względem trasy i ewentualne korekty,

- inne czujniki — do określania konfiguracji robota w czasie ruchu, potrzebnej w bardziej zaawansowanych algorytmach, należy użyć odpowiednich sensorów.



Rysunek 4.1: Robot użyty do testów

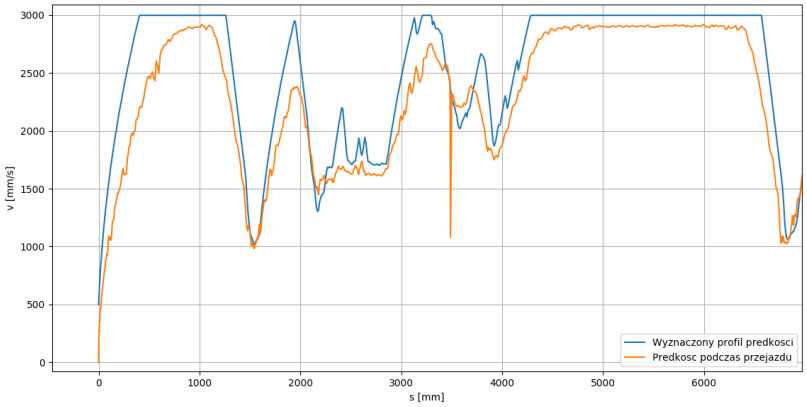
Tak jak w przypadku większości konstrukcji tego typu, podwozie omawianego robota jest jednocześnie płytką PCB na której zamontowano wszystkie potrzebne elementy. Aby dodatkowo zmniejszyć wagę robota podzielono jego konstrukcję na dwie części: płytkę z czujnikami linii oraz moduł główny, połączone ze sobą za pomocą listwy węglowej oraz taśmy sygnałowej. Dzięki temu, poza redukcją wagi, możliwa jest regulacja wysunięcia listwy z czujnikami oraz jej ewentualna wymiana. Robot został zaprezentowany na rysunku 4.1.

4.2 Napęd

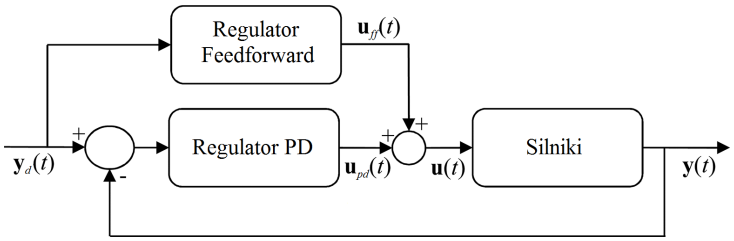
Niezbędnym elementem każdego robota mobilnego jest umożliwiający ruch napęd. W opisywanej konstrukcji użyto popularne w amatorskiej robotyce mikrosilniki DC firmy Pololu. Charakteryzują się one prędkością obrotową 3000 obr/min i momentem obrotowym wynoszącym 0,3 kg*cm. Posiadają one przekładnie 10:1 oraz przedłużony wał do montażu enkoderów. Parametry te w zupełności wystarczą do poruszania się z dużymi prędkościami i osiągania wymaganych przyspieszeń. Do sterowania silnikami użyto mostków H *TB6612FNG* [24].

Sterowanie silnikami w układzie otwartym jest niewystarczające dla niektórych z zaimplementowanych algorytmów. Do kontroli prędkości obrotowej silników przy użyciu sprzężenia zwrotnego najczęściej stosowany jest regulator PID [15], który w większości przypadków dobrze spełnia swoje zadania. Specyfika konkurencji linefollower, gdzie zmiany prędkości są bardzo dynamiczne a czas przejazdu jest krótki, powoduje, że sygnał na członie całkującym nie nadąża narastać i wprowadza niepożądane opóźnienie, przez co lepiej sprawuje się prostszy regulator PD [12].

Jednak, aby zaimplementować wysokiej jakości sterownik prędkości obrotowej silnika, nie wystarczy dodać sprzężenie zwrotne oraz regulator PD. Dla dużych prędkości sterowanie samym uchybem może okazać się niewystarczające. Taką sytuację przedstawiono na rysunku 4.2, gdzie próbuje się na prostym odcinku trasy osiągnąć prędkość 3 m/s, stosując jedynie regulator PD na silnikach. Można zauważyć pewien błąd ustalony, ponieważ robot przestaje przyspieszać i nie osiąga zadanej prędkości. Dlatego w robotyce często stosuje się dodatkowo tak zwane sterowanie feedforward, czyli ze sprzężeniem w przód,



Rysunek 4.2: Porównanie prędkości zadanej z osiągniętą przy użyciu regulatora PD



Rysunek 4.3: Układ sterowania silnikami z dodanym członem feedforward

nazywane też w literaturze regulatorem PD z korekcją [23], którego sposób działania został przedstawiony na rysunku 4.3. Do skutecznego sterowania tą metodą potrzebna jest jednak znajomość charakterystyki silników. Badania przeprowadzone w celu jej zebrania zostały omówione w podrozdziale 5.1.

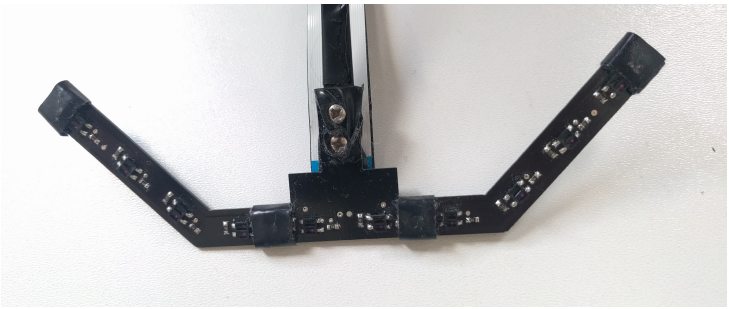
4.3 Czujniki

4.3.1 Wykrywanie linii

Podstawowym zadaniem każdego robota klasy linefollower jest wykrycie linii. Zdecydowanie najczęściej stosowanym rozwiązaniem są czujniki odbiciowe, które są zbudowane z diody IR oraz fototranzystora, dzięki czemu umożliwiają wykrycie linii kontrastującej z podłożem. Ich ograniczeniem jest jednak mały zasięg, ponieważ linia musi znajdować się bezpośrednio pod czujnikiem, aby mogła być wykryta.

Drugim sposobem wykrywania linii jest użycie kamery, która z początku wydaje się być najlepszym rozwiązaniem. Zamocowanie jej pod odpowiednim kątem pozwala na wykrywanie linii znajdującej się przed robotem i przygotowanie go odpowiednio do nadchodzących zakrętów. Jednakże użycie kamery nastręcza wiele trudności: wymaga zastosowania złożonych obliczeniowo algorytmów przetwarzania obrazów a co za tym idzie wydajniejszej jednostki obliczeniowej. Z powodu prędkości i częstych zakrętów obraz może okazać się rozmyty i musi być przetwarzany z dużą częstotliwością dochodzącą do 100 FPS na co pozwalają jedynie nieliczne kamery. Co więcej, zamontowanie kamery wraz z odpowiednim układem kontrolnym znacząco zwiększa wagę robota.

Ze względu na przedstawione argumenty w omawianym robocie zdecydowano się na użycie czujników odbiciowych. Aby umożliwić poruszanie się z dużą prędkością moduł



Rysunek 4.4: Moduł z czujnikami linii

z czujnikami musi być wysunięty do przodu — im dalej, tym wcześniej dostajemy informację o zakrętach. Jednak nie należy przesadzić, moduł stanowi bowiem dodatkową masę położoną daleko od środka ciężkości robota, przez co znacząco wpływa na jego bezwładność. Dlatego też, w opisywanym rozwiązaniu moduł jest jak najmniejszy i ma kształt zakrzywionej listwy, przedstawionej na rysunku 4.4. Zakrzywienie listwy ma znaczenie przy pokonywaniu ostrych zakrętów (np. kąta prostego), ponieważ nawet jeśli robot nie jest podczas manewru ustawiony równolegle do linii lub "wypadnie" czujnikami poza linię, skrajny czujnik wysunięty do tyłu jako ostatni ją wykrywa i pozwala na powrót na trasę.

Zastosowany w robocie moduł wyposażony jest w 10 czujników w formie transoptorów odbiciowych *KTIR0711S* [20], co w zupełności wystarcza na zmierzenie położenia linii podczas przejazdów. Pomimo, że czujniki te pozwalają na odczyt analogowy, zwykle stosuje się progowanie, aby sygnał z nich wychodzący był binarny — linia wykryta lub nie. Pomiary analogowe można wykorzystać do dokładniejszej estymacji położenia linii [19], jednak zysk płynący z zastosowania takiego rozwiązania jest mały a implementacja utrudniona, ponieważ do poprawnego działania wymaga dużej ilości obliczeń i kompensacji różnic w charakterystykach poszczególnych czujników. Przy konfiguracji modułu jak na rysunku 4.4, linia znajdująca się pod listwą jest zawsze wykrywana przez jeden lub dwa czujniki, co po uśrednianiu odczytów minimalizuje błąd pomiaru jej położenia do ok. 4 mm.

4.3.2 Estymacja położenia robota

Do najprostszych algorytmów śledzenia trasy w zupełności wystarczą tylko czujniki linii, jednak aby móc zaimplementować algorytmy budujące mapę przejechanej trasy i wykorzystujące ją w kolejnych próbach potrzebne są dodatkowe czujniki.

Podstawowymi czujnikami mierzącymi przebytą drogę stosowanymi w robotach mobilnych są enkodery. Pozwalają one na mierzenie przesunięcia kątowego, a co za tym idzie prędkości obrotowej silników, z czego, po odpowiednich przekształceniach, obliczyć można dystans przebyty przez każde z kół robota.

Zdecydowano się na użycie inkrementalnych enkoderów magnetycznych *AS5040* firmy AMS [6] które, wraz zamocowanym na wale silnika magnesem, zapewniają rozdzielczość 1024 impulsów na obrót wału, co po uwzględnieniu przekładni silnika, daje 10240 impulsów na obrót koła. Przy używanych kołach o średnicy 22 mm daje to rozdzielczość 6.7 μm. Używając enkoderów możemy zatem z dużą dokładnością estymować położenie robota na trasie. Przy poruszaniu się z dużymi prędkościami, musimy jednak wziąć pod uwagę poślizgi kół, które mocno zależą od materiału z jakiego wykonane są opony, powierzchni po której robot się porusza oraz poziomu jej zabrudzenia.

Aby zniwelować wpływ poślizgów na estymację położenia zastosowano dodatkowo mo-

duł 3-osiowego akcelerometru oraz żyroskopu *MPU-6050* [18]. Żyroskop pozwala na mierzenie prędkości kątowej robota w trzech osiach, w efekcie, po scałkowaniu, jego orientacji. Odczyty z żyroskopu są niewrażliwe na poślizgi, ale charakteryzują się dryftem, który jest błędem pomiarowym kumulującym się w czasie. Oznacza to na przykład, że gdy robot stoi nieruchomo wartości odczytów są stale różne od zera. Błąd ten można odfiltrować stosując metody takie jak filtr Kalmana lub filtr komplementarny [16]. Filtr Kalmana jest jednak skomplikowany obliczeniowo i do działania wymaga wyprowadzenia modelu. Dużo prostszy w implementacji filtr komplementarny nie nadaje się natomiast do wyznaczania orientacji robota bez dodatkowych odczytów z magnetometru.

Z racji, że przejazdy po trasie są krótkie i trwają jedynie kilkanaście sekund, a błąd żyroskopu nie zdąży znacząco wpłynąć na pomiary, w robocie zastosowano prostą kompensację tego błędu. Robot, stojąc nieruchomo przed startem przez 5 sekund, oblicza skumulowany błąd i na jego podstawie szacuje dryft żyroskopu, który następnie odejmuje od odczytów podczas właściwego przejazdu. Rozwiązanie to w zupełności wystarcza do bieżących zastosowań robota.

4.4 Komunikacja

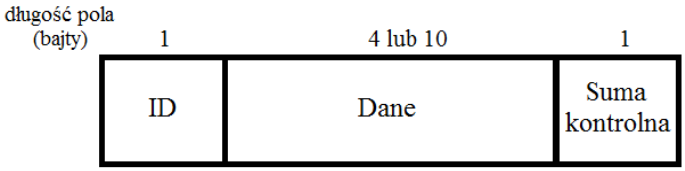
Budowanie mapy oraz badania algorytmów wymagają dokładnej analizy trasy, a co za tym idzie dużej ilości danych, dlatego podczas przejazdu robot odczytuje pomiary z opisanych czujników z częstotliwością 200 Hz. Przy przejazdach trwających nawet 30 sekund, oznacza to co najmniej 24000 próbek do zapamiętania. Zdalna komunikacja z robotem jest więc niezbędna do zbierania wszystkich tych danych, ponieważ pamięć dostępna w zastosowanym mikrokontrolerze jest niewystarczająca do przechowywania takich ilości informacji.

Do komunikacji wykorzystano moduł Bluetooth *HC-06* [2] działający na interfejsie UART, co umożliwia bezpośrednie połączenie z komputerem. Parametry połączenia:

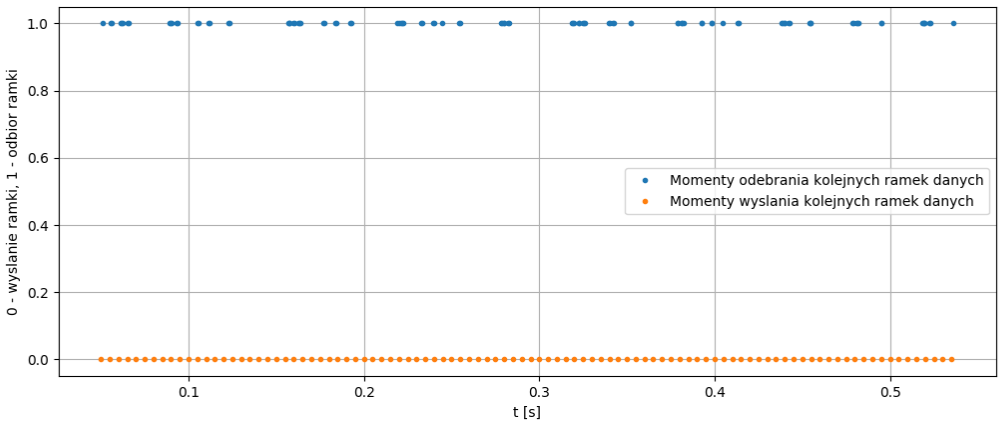
- prędkość transmisji 115200 bps,
- 8 bitów danych,
- bez bitu parzystości,
- 1 bit stopu.

W celu zmniejszenia rozmiaru danych do wysłania oraz monitorowania poprawności transmisji przesyłane wiadomości są pakowane w odpowiednie ramki danych. Dzięki temu ustandaryzowano również długość przesyłanych i odbieranych wiadomości, więc komponenty odpowiadające za komunikację mogą oczekiwać na konkretną liczbę bajtów do odebrania. Długość ramki wysyłanej przez robota to 12 bajtów a odbieranej 6 bajtów. Pierwszy bajt informuje o typie przesyłanych informacji, następnie przesyłane jest 10 (lub 4 w przypadku wysyłania do robota) bajtów danych a na samym końcu jeden bajt zawierający sumę kontrolną. Ramka danych została zaprezentowana na rysunku 4.5.

Przeprowadzone testy wykazały, że komunikacja przebiega bezproblemowo a błędy w transmisji praktycznie nie występują. Zastosowany moduł ma jednak problem z zachowaniem zaplanowanego reżimu czasowego, nie jest więc możliwe jednoczesne wysyłanie i odbieranie danych z założoną częstotliwością. Występujące opóźnienia w komunikacji, pokazane na rysunku 4.6, powodują nieregularność w odbiorze danych i ich kumulację. W przypadku implementacji algorytmu z podrozdziału 3.3, wyklucza to przechowywanie całej mapy w komputerze i przesyłanie jej na bieżąco podczas przejazdu, dlatego większa część mapy musi być przesłana wcześniej i zapisana na robocie.



Rysunek 4.5: Struktura ramki danych



Rysunek 4.6: Opoźnienia w przesyłaniu danych

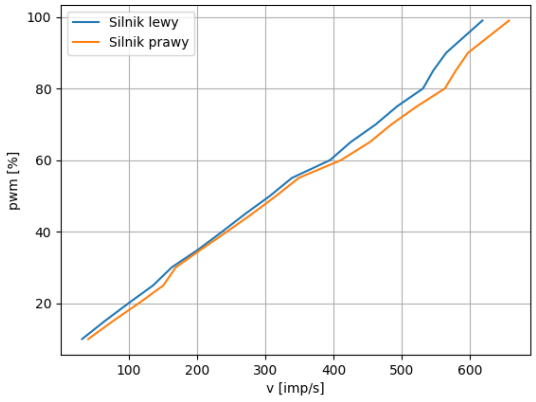
Rozdział 5

Badanie własności algorytmów

W rozdziale opisane zostały przeprowadzone w ramach pracy badania. Rozpoczęto od analizy napędów zastosowanych w robocie i możliwych do osiągnięcia przy ich pomocy przyspieszeń. Następnie opisano badania modułu korygującego oszacowanie położenia robota w czasie przejazdu z użyciem mapy trasy. W dalszej części przybliżono przeprowadzone badania porównawcze zaimplementowanych algorytmów. Rozdział zakończono sprawozdaniem ze startów w zawodach badanego robota pod kontrolą algorytmu z nadrzędnym profilerem prędkości.

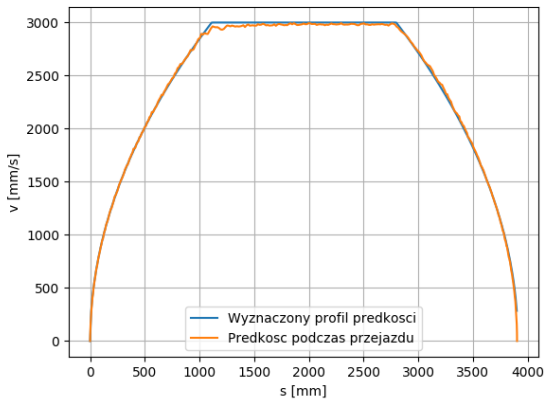
5.1 Badanie charakterystyki napędów

W celu dokonania identyfikacji silników zaproponowano eksperyment polegający na zadawaniu na silniki robota stałego sygnału sterującego (w formie sygnału PWM o określonym wypełnieniu) i przejechaniu dystansu pozwalającego na osiągnięcie ustabilizowanej prędkości. Prędkość ta została zmierzona za pomocą zamontowanych enkoderów. Charakterystyka obu silników, zbudowana na podstawie serii takich eksperymentów dla różnych poziomów wypełnienia PWM, co 5% od 10% do 100%, została przedstawiona na rysunku 5.1.



Rysunek 5.1: Prędkość osiągana przez silniki przy danym wypełnieniu sygnału PWM

Ze względu na to, że charakterystyka silnika jest zbliżona do liniowej, zdecydowano się na aproksymację linią prostą, która jest łatwa w implementacji i nie wymaga skomplikowanych obliczeń. Jak widać, charakterystyki obu silników różnią się od siebie, dlatego dla



Rysunek 5.2: Prędkość zadana oraz osiągnięta przy użyciu badanego regulatora

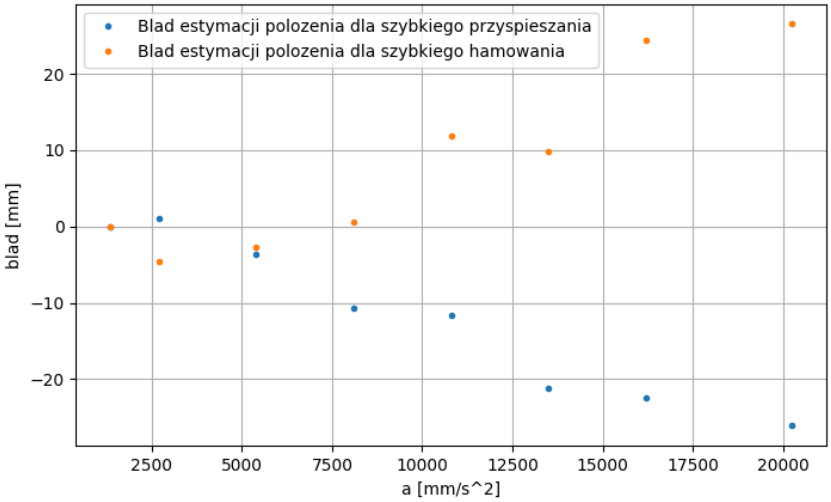
każdego z nich dobrano odpowiednią aproksymację. Obliczone aproksymacje, minimalizujące kwadrat błędu w badanych punktach, to kolejno $y_l = 1.5x + 4$ dla silnika lewego oraz $y_r = 1.4x + 4$ dla silnika prawego. Mając modele silników można już nimi skutecznie sterować używając pętli feedforward oraz dodatkowego regulatora PD do korekcji błędu. Na rysunku 5.2 przedstawiono przebieg porównujący zadaną prędkość na regulator z rzeczywistą, osiągniętą przez robota na dystansie ok. 4 metrów.

5.2 Badanie maksymalnych przyspieszeń

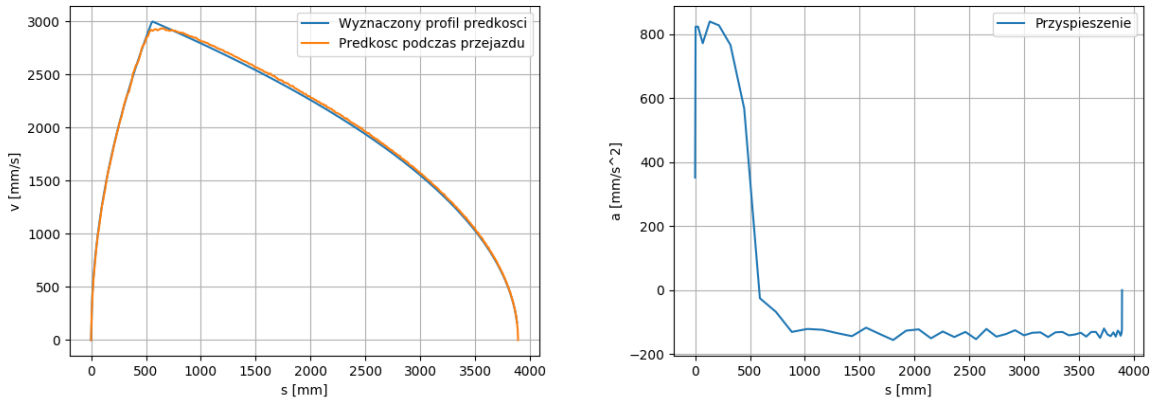
Do osiągnięcia jak najlepszych rezultatów przy zastosowaniu nadrzędnego profilera prędkości do ustalania optymalnego przebiegu po danej trasie, należy zbadać możliwości robota odnośnie maksymalnych wartości przyspieszania i hamowania. Muszą być one jak największe, aby zmaksymalizować zysk płynący z zastosowania nadrzędnego sterownika, a jednocześnie podanie za dużych wartości przyspieszenia może skutkować poślizgami, a co za tym idzie błędami w określaniu położenia i mniejszą dynamiką jazdy. Dlatego ważne jest eksperymentalne dobranie optymalnych wartości.

Przeprowadzony eksperyment zakładał zadanie regulatorowi prędkości silników przejechania określonego dystansu (w tym wypadku 3.87 metra) z określonym profilem przyspieszenia. Następnie porównano ze sobą estymowane przez robota położenie końcowe oraz zmierzone jego faktyczne położenie. W pierwszej próbie przeprowadzono referencyjny przejazd dla niskich wartości przyspieszeń, aby zbadać ustalony błąd estymacji położenia. W kolejnych próbach zwiększano przyspieszenie zachowując wartość opóźnienia, aby określić maksymalną wartość przy której niepewność pozycji końcowej mieści się w akceptowanym zakresie. Następnie wykonano taką samą serię prób, z tą różnicą, że przyspieszenie pozostało niskie a zmieniało się tempo z jakim robot się zatrzymywał.

Wyniki uzyskane podczas eksperymentu zaprezentowano na rysunku 5.3, a przykład zebranych danych dla jednego z przebiegów na rysunku 5.4. Wyniki są zgodne z intuicją, robot za szybko przyspieszając buksuje kołami, przez co estymuje, że przejechał większy dystans niż w rzeczywistości. Natomiast przy zbyt gwałtownym hamowaniu, z powodu bezwładności, robot ślizga się i przekłamuje oszacowanie własnej pozycji w drugą stronę. Na podstawie zebranych danych można założyć, że maksymalne przyspieszenie dla którego błąd położenia oscyluje w granicach 10 mm wynosi 10.8 m/s^2 a opóźnienie 13.5 m/s^2.



Rysunek 5.3: Błąd estymacji położenia dla różnych przyspieszeń

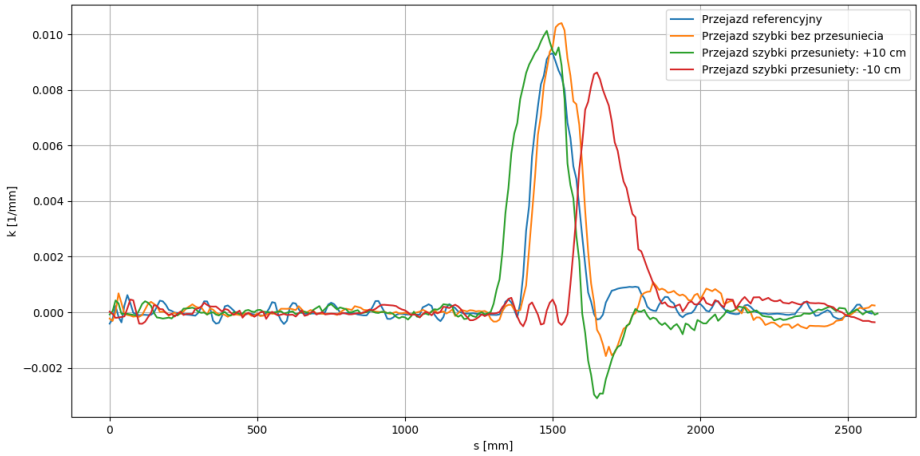


(a) Prędkość rzeczywista oraz zadana (b) Rzeczywiste przyspieszenie

Rysunek 5.4: Dane zebrane dla zadanych wartości przyspieszania i hamowania kolejno 8.1 m/s^2 oraz 1.35 m/s^2

Tabela. 5.1: Działanie modułu korygującego położenie

przesunięcie rzeczywiste	przesunięcie obliczone
0	-2
10	4
15	7
-10	-11
-15	-16



Rysunek 5.5: Porównanie $k(s)$ w zależności od przesunięcia punktu początkowego

5.3 Badanie modułu korygującego estymację położenia

Jak wskazano w podrozdziale 3.3, korekcja położenia w czasie przejazdu jest często niezbędna do poprawnego działania algorytmu bazującego na mapie. Eksperyment mający na celu weryfikację poprawności działania modułu korygującego estymację położenia robota polegał na zebraniu mapy trasy zawierającej jeden ostry zakręt, a następnie wykonaniu kilku szybkich przejazdów na jej podstawie. Podczas tych przejazdów zmienione zostały jednak położenia początkowe robota, co miało za zadanie zasymulować błąd oszacowania położenia, który powinien zostać wykryty przez moduł. Zebrane wyniki zostały przedstawione w tabeli 5.1, jak widać, działanie korekcji jest zadowalające jedynie gdy robot estymuje, że znajduje się dalej niż w rzeczywistości, natomiast gdy jest bliżej zmniejsza się dokładność korekcji. Jest to spowodowane tym, że gdy faktyczne położenie jest bliżej pokonywanego zakrętu niż oszacowano, robot wjeżdża w zakręt z większą prędkością, co zmienia przebieg jego krzywizny. W przypadku gdy oszacowanie jest błędne w drugą stronę, robot zwalnia szybciej niż powinien, ale dzięki temu wjeżdża w zakręt z małą prędkością, która jest bliższa tej, z jaką budowano mapę. Przebiegi $k(s)$ dla różnych punktów początkowych zostały zaprezentowane na rysunku 5.5. W takiej postaci moduł korekcji może wspomagać estymator położenia, jednak cały czas istnieje potrzeba opracowania mniej zawodnej techniki korygowania położenia.

Tabela. 5.2: Czasy uzyskane przy użyciu algorytmów — 1 trasa

algorytm	czas [s]	średnia prędkość [m/s]
P	5.7	1.33
PD	4.3	1.76
PD + sterownik prędkości	4.2	1.80
mapowanie	5.6	1.35
PD + profiler (max 2 m/s)	4.1	1.85
PD + profiler (max 3 m/s)	3.5	2.16
PD + profiler (max 4 m/s)	3.2	2.37

5.4 Badanie porównawcze algorytmów

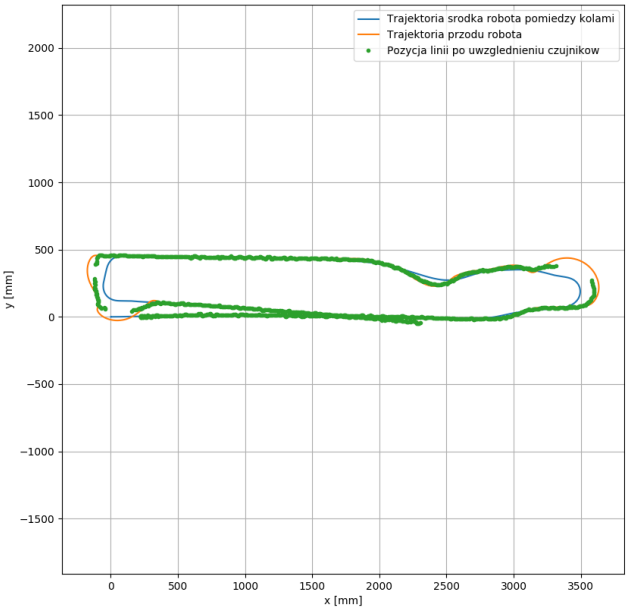
Badania porównawcze zaimplementowanych algorytmów przeprowadzono na trasach zaprezentowanych w podrozdziale 2.2. Wykonane eksperymenty polegały na mierzeniu czasu przejazdu jednego okrążenia robota kontrolowanego algorytmem:

- regulator P (automat skończony),
- regulator PD,
- regulator PD z kontrolą rzeczywistej prędkości,
- regulator PD z nadrzędnym sterownikiem prędkości zadanej.

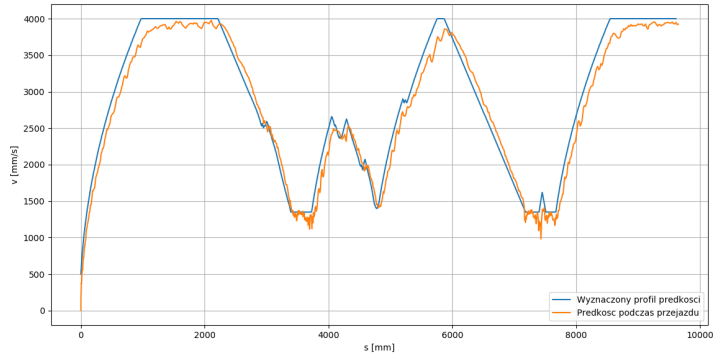
Prędkość maksymalna, ustawiana podczas przejazdów z użyciem algorytmu analizującego przebieg trasy, była ograniczona w kolejnych próbach do 2 i 3 m/s (oraz 4 m/s dla pierwszej trasy), przy czym profil dla szybszej próby bazował na mapie zebranej podczas przejazdu poprzedniego. Dzięki temu ograniczono wpływ poślizgów na błąd estymacji położenia robota w czasie ruchu i wyeliminowano potrzebę stosowania modułu korygującego.

Pierwsza trasa testowa została zaprojektowana tak, aby w pełni wykorzystać zalety algorytmu używającego mapy. Długie proste, na których można osiągnąć duże prędkości, są zakończone ostrymi zakrętami, które ograniczają prędkość maksymalną możliwą do uzyskania przez prostsze algorytmy. Wyniki przedstawione na rysunku 5.6 oraz czasy uzyskane przez poszczególne algorytmy, zestawione w tabeli 5.2, pokazują dużą poprawę wyników uzyskiwanych dzięki zastosowaniu algorytmu optymalizującego prędkość. Uzyskiwany przez niego czas jest o 24% krótszy niż drugiego z kolei algorytmu. Można zauważyć również minimalną poprawę wyników osiągniętych przez algorytm PD po dodaniu regulatora rzeczywistej prędkości. Pomimo takich samych zadawanych parametrów, jest on zwrotniejszy i szybciej osiąga zadaną prędkość, szczególnie przy pokonywaniu zakrętów. Najprostszy algorytm, działający jedynie na regulatorze P osiąga dużo gorsze rezultaty niż reszta algorytmów, ponieważ jego prędkość ograniczają silne oscylacje pojawiające się po zakrętach.

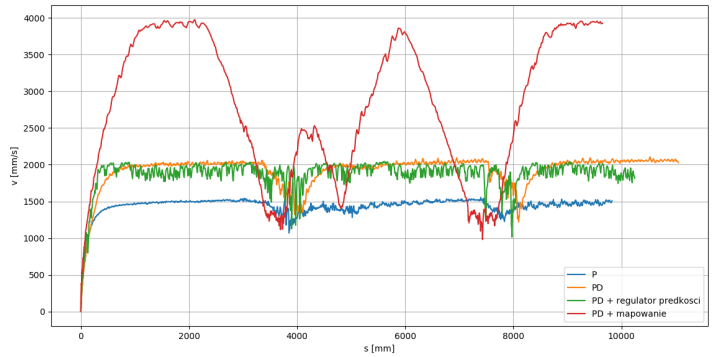
Druga trasa nie zawiera długich odcinków prostych, jest bardzo ciasna i składa się z następujących po sobie zakrętów o różnym stopniu skomplikowania oraz jednego skrzyżowania. Takie ukształtowanie minimalizuje korzyści płynące z zastosowania optymalizacji prędkości, ponieważ krzywizna trasy osiąga nieustannie duże wartości, co ogranicza prędkość maksymalną ustawianą przez nadrzędny sterownik. Wyniki, zaprezentowane w tabeli 5.3 oraz na rysunku 5.7, potwierdzają te założenia. Pomimo, że algorytm wykorzystujący mapowanie znowu osiągnął najlepszy rezultat, różnice są w tym wypadku niewielkie.



(a) Mapa trasy zebrana podczas przejazdu

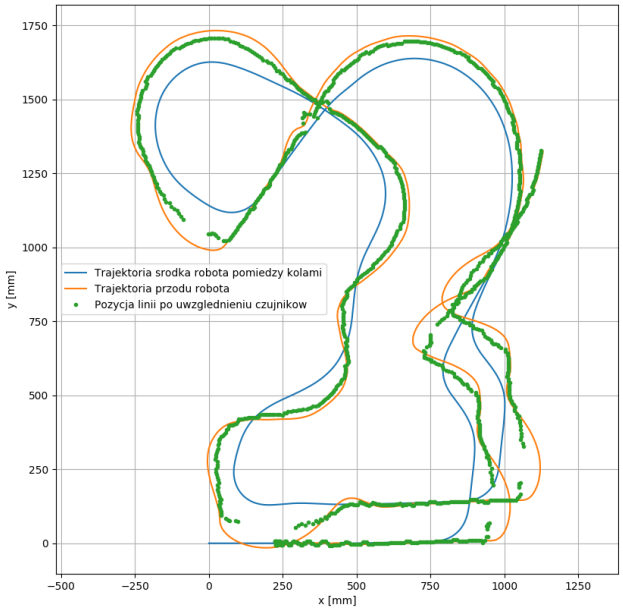


(b) Wyznaczony profil oraz rzeczywista prędkość osiągnięta podczas przejazdu z użyciem algorytmu analizującego mapę

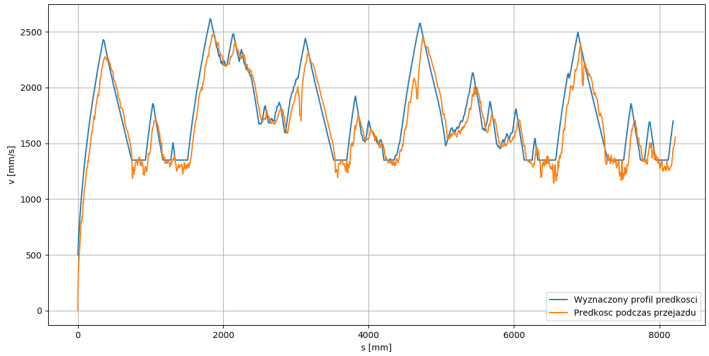


(c) Prędkości przejazdów przy użyciu porównywanych algorytmów

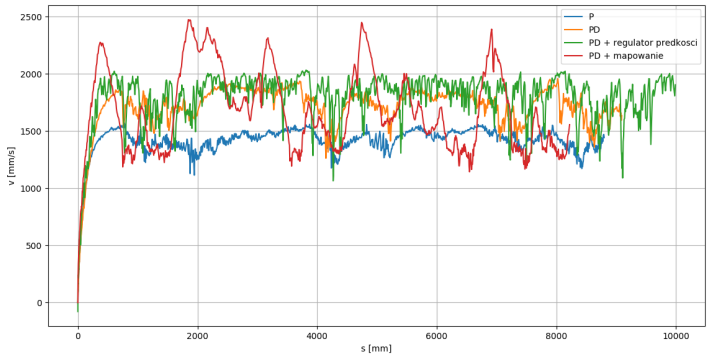
Rysunek 5.6: Trasa 1 — prezentacja danch



(a) Map trasy zebrana podczas przejazdu



(b) Wyznaczony profil oraz rzeczywista prędkość osiągnięta podczas przejazdu z użyciem algorytmu analizującego mapę



(c) Prędkości przejazdów przy użyciu porównywanych algorytmów

Rysunek 5.7: Trasa 2 — prezentacja danch

Tabela. 5.3: Czasy uzyskane przy użyciu algorytmów — 2 trasa

algorytm	czas [s]	średnia prędkość [m/s]
P	5.1	1.34
PD	4.1	1.67
PD + sterownik prędkości	4.0	1.71
mapowanie	4.9	1.39
PD + profiler (max 2 m/s)	4.0	1.71
PD + profiler (max 3 m/s)	3.9	1.75

Tabela. 5.4: Czasy uzyskane przy użyciu algorytmów — 3 trasa

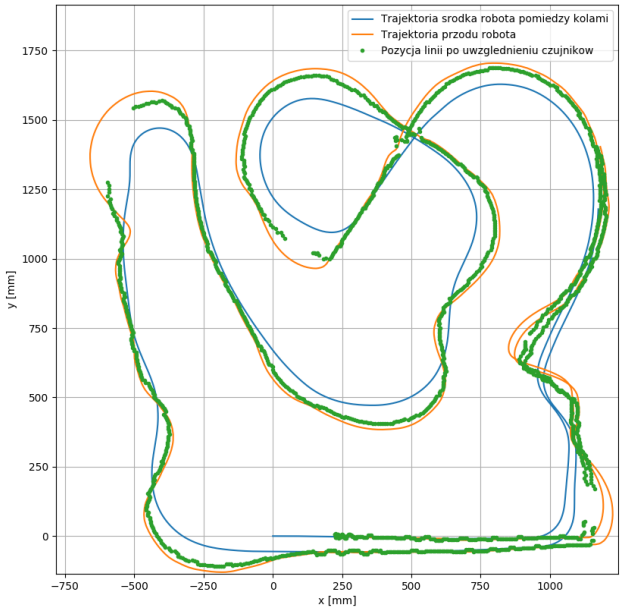
algorytm	czas [s]	średnia prędkość [m/s]
P	7.2	1.41
PD	6.1	1.67
PD + sterownik prędkości	5.9	1.72
mapowanie	7.2	
PD + profiler (max 2 m/s)	5.7	1.78
PD + profiler (max 3 m/s)	5.5	1.85

Po raz kolejny oba warianty typowego regulatora PD uzyskały podobne czasy przejazdu, z lekką przewagą wersji rozbudowanej.

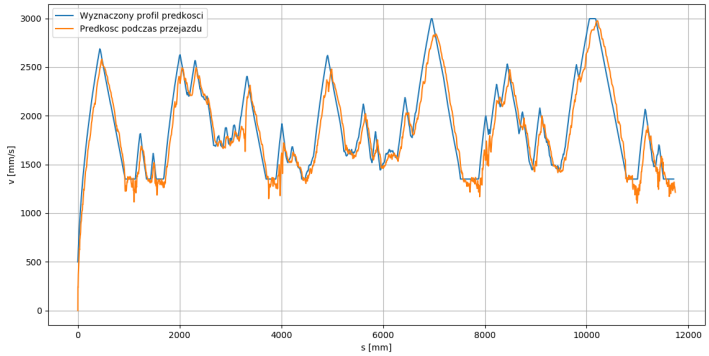
Ostatnia badana trasa, najdłuższa ze wszystkich, reprezentuje typową trasę spotykaną podczas zawodów, zawiera zarówno proste odcinki jak i skomplikowane elementy, takie jak skrzyżowania czy serie ostrych zakrętów. Wnioski wyciągnięte z uzyskanych na tej trasie wyników, które zostały przedstawione w tabeli 5.4 oraz na rysunku 5.8, pokrywają się z zaobserwowanymi wcześniej. Stosunek odcinków prostych do długości całej trasy był większy niż na trasie drugiej, można więc zauważyć poprawę rezultatów osiągniętych przez algorytm z nadrzędnym profilerem prędkości, natomiast średnia prędkość osiągana przez konwencjonalne regulatory PD praktycznie się nie zmieniła. Małe oscylacje prędkości, które można zauważyć podczas przejazdów z algorytmem PD z regulatorem rzeczywistej prędkości, wynikają z silnych korekt pozycji, wymuszanych podczas jazdy przez ten regulator, które w prostszych algorytmach są wygładzone przez niską responsywność. Nie zaobserwowano wpływu tych oscylacji na uzyskiwany czas, mogą jednak one być zminimalizowane przez odpowiedni dobór wag odzwierciedlających błąd położenia dla czujników linii położonych bliżej środka modułu. Warto odnotować, że czas przejazdu robota kontrolowanego regulatorem P znacząco zależy od tego, czy robot wpadnie w oscylacje podczas danej próby, natomiast dla pozostałych algorytmów uzyskiwane czasy są powtarzalne.

5.5 Porównanie z innymi konstrukcjami

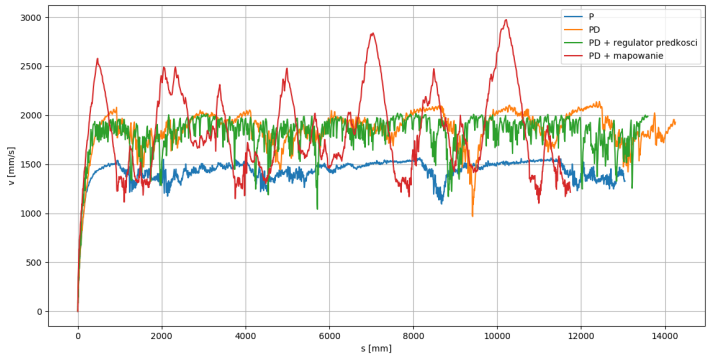
Aby należycie zbadać korzyści płynące z zastosowania zaproponowanego algorytmu, analizującego przebieg trasy i wykorzystującego go do wyznaczania optymalnych parametrów podczas kolejnych przejazdów, wystawiono testowaną konstrukcję na dwóch zawodach robotycznych. Umożliwiło to porównanie czasów osiąganych na tych samych trasach z najlepszymi konstrukcjami tego typu w Polsce. Warto zauważyć, że podczas zawodów badany algorytm nie był jeszcze gotowy w swojej finalnej wersji, start w zawodach dał więc możliwość obiektywnego spojrzenia na postępy i wyznaczenia elementów wymagających



(a) Mapa trasy zebrana podczas przejazdu

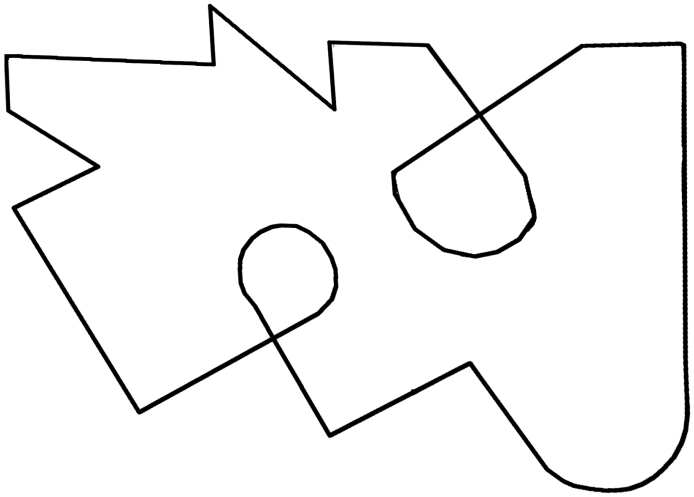


(b) Wyznaczony profil oraz rzeczywista prędkość osiągnięta podczas przejazdu z użyciem algorytmu analizującego mapę



(c) Prędkości przejazdów przy użyciu porównywanych algorytmów

Rysunek 5.8: Trasa 3 — prezentacja danch



Rysunek 5.9: Trasa eliminacyjna podczas zawodów Cyberbot 2017

poprawy oraz rozbudowy.

5.5.1 XIV Festiwal Robotyki Cyberbot 2017 w Poznaniu

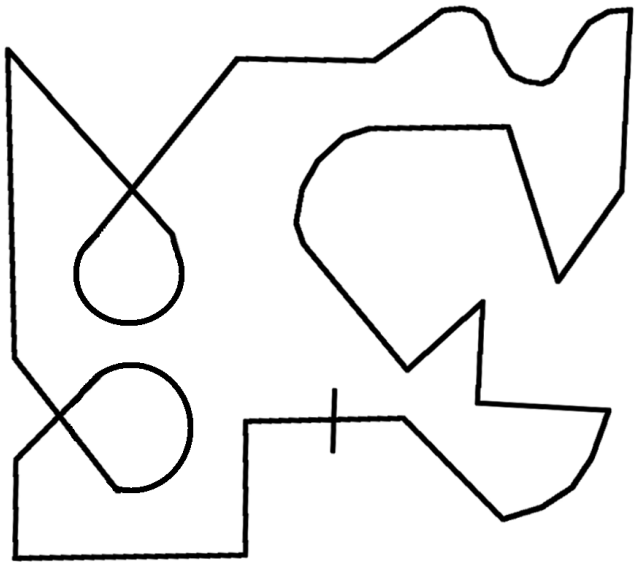
Pierwszy test algorytmu wykorzystującego mapę przeprowadzono podczas zawodów na Festiwalu Robotyki Cyberbot [1] w Poznaniu. Na zawodach tych organizatorzy przygotowali trasy zawierające długie odcinki proste oraz niewiele krętych odcinków, np. nie występujące nigdzie indziej zakręty o kącie 135 stopni. Trasy zostały przedstawione na rysunkach 5.9 oraz 5.10. Jak wspomniano już wcześniej, jest to idealny typ trasy dla algorytmów bazujących na mapie trasy. Podczas gdy roboty ze standardowymi algorytmami mają ograniczoną prędkość do takiej, która pozwala na przejechanie najtrudniejszego odcinka, robot znający specyfikę trasy przyspiesza na prostszych odcinkach i zwalnia tuż przed zakrętami.

Potwierdzają to wyniki osiągnięte przez robota. Trasa do pokonania podczas eliminacji, miała ponad 23 metry długości, z czego znaczącą część stanowiły odcinki proste. Uzyskany czas najszybszego przejazdu z algorytmem mapującym wyniósł 11.47 sekundy, a drugiego w kolejności robota 12.56 sekundy.

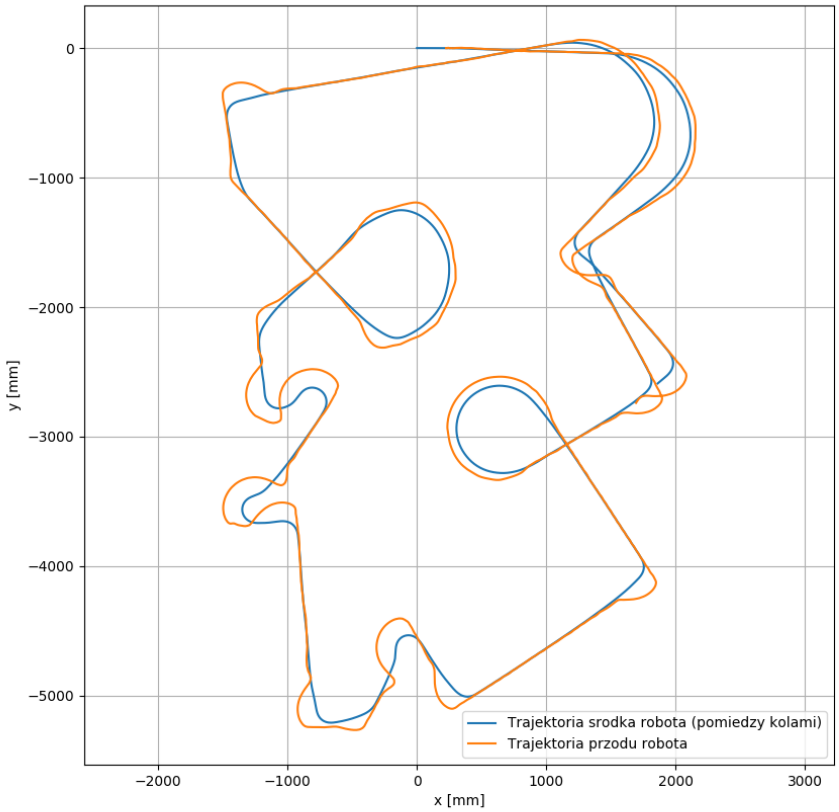
Mapa trasy, zbudowana na podstawie danych zgromadzonych podczas przejazdu została przedstawiona na rysunku 5.11, obliczony dzięki niej profil prędkości pokazano na rysunku 5.12. Jak widać, pomimo że robot zwalnia przed ostrymi zakrętami do około 1 m/s, trasa zawiera wiele odcinków na których robot może rozwinąć maksymalną prędkość, dzięki czemu średnia prędkość podczas całego przejazdu przekracza 2 m/s. Algorytm wyznaczający prędkość, może również obliczyć szacowany czas przejazdu robota, który w przypadku najszybszego przejazdu wyniósł 11.36 sekundy co jest wynikiem zbliżonym do osiągniętego. Dowodzi to, że robot wiernie realizuje zadany profil prędkości.

W trakcie dwóch dostępnych w finale prób, udało się tylko zebrać mapę trasy, próba szybkiego pokonania trasy się nie powiodła. Udany szybki przejazd z użyciem mapy przeprowadzono dopiero poza konkursem, znowu osiągając wynik dużo lepszy od robotów z zaimplementowanymi algorytmami wykorzystującymi jedynie aktualne odczyty z czujników. Wyniósł on 14.3 sekundy, podczas gdy zwycięski czas to aż 15.16 sekundy.

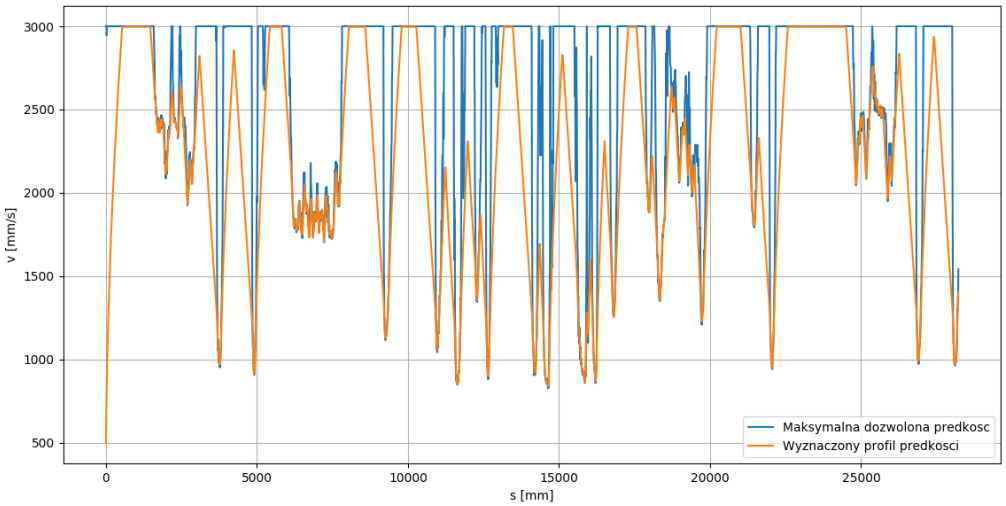
Mapa uzyskana w czasie przejazdów oraz profil prędkości z nałożoną rzeczywistą prędkością najszybszego przejazdu zostały przedstawione na rysunkach 5.13 i 5.14. Podczas



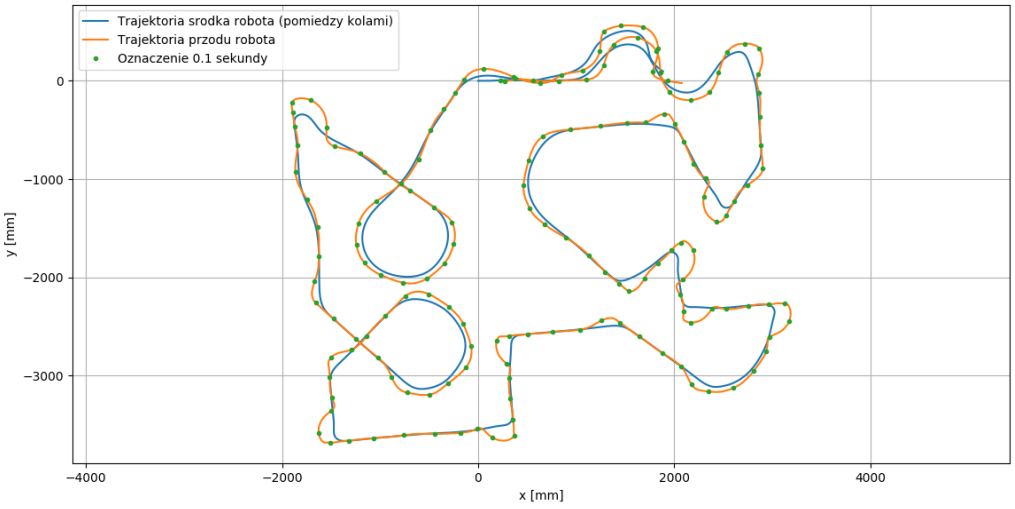
Rysunek 5.10: Trasa finałowa podczas zawodów Cyberbot 2017



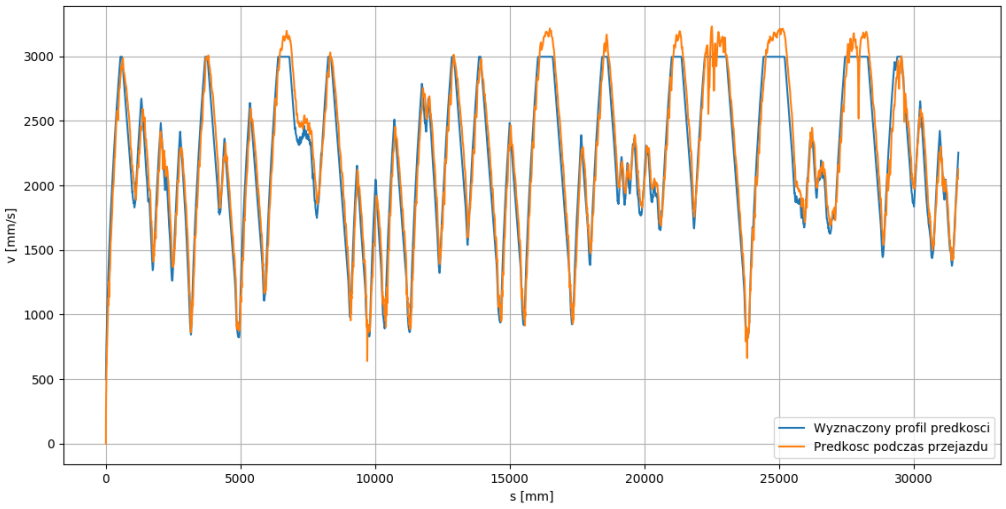
Rysunek 5.11: Mapa trasy eliminacyjnej zebrana podczas przejazdu — Cyberbot 2017



Rysunek 5.12: Profil prędkości wyznaczony dla trasy eliminacyjnej — Cyberbot 2017



Rysunek 5.13: Mapa trasy finałowej zebrana podczas przejazdu — Cyberbot 2017



Rysunek 5.14: Wyznaczony profil oraz rzeczywista prędkość osiągnięta podczas przejazdu na trasie finałowej — Cyberbot 2017

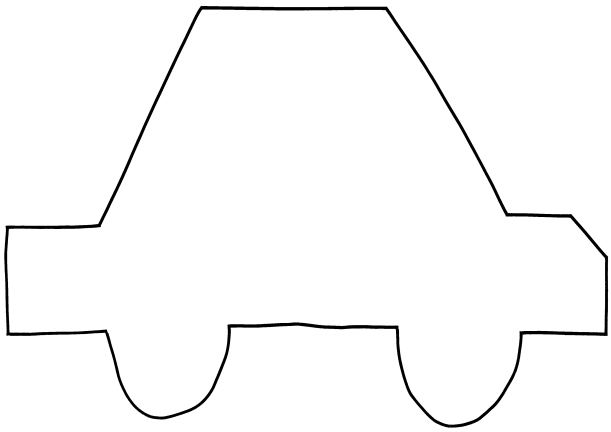
zawodów nie był jeszcze dostrojony człon feedforward, co widać na porównaniu prędkości rzeczywistej z zadaną, która jest większa na prostych odcinkach. Aby pokazać jak zmieniała się prędkość w zależności od położenia na trasie, do mapy przejazdu dodano kropki, oznaczające znaczniki czasu co 0.1 sekundy — odległości między znacznikami są większe na prostych odcinkach a zmniejszają się, gdy robot zbliża się do zakrętów, co obrazuje zmiany prędkości.

5.5.2 Trójmiejski Turniej Robotów 2017 w Gdańsku

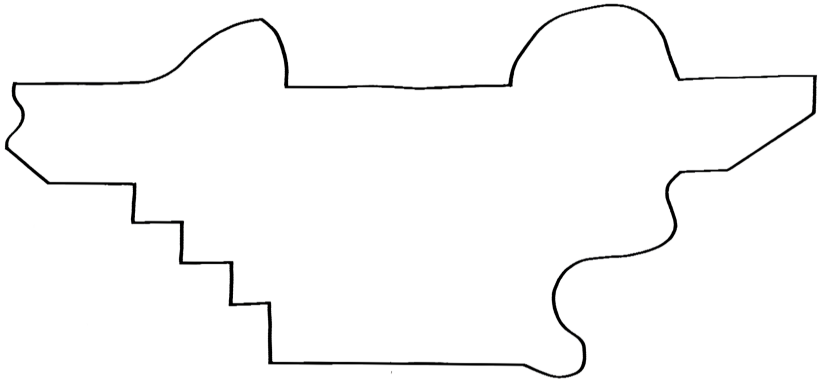
Trasy przygotowane przez organizatorów TTR [5], zaprezentowane na rysunkach 5.15 i 5.16, różniły się od tych w Poznaniu. Były krótkie (około 10 metrów) i kręte, co znacząco obniżyło efektywność zastosowania mapy do wyznaczania prędkości. Duża liczba zakrętów utrudnia estymację położenia przy dużych prędkościach, przez co musi być ona ograniczana. Dlatego na takich trasach przewagę zyskują roboty z klasycznym algorytmem PD, który jest odporniejszy na poślizgi i może wchodzić w zakręty z dużo większą prędkością.

Pomimo tego podczas eliminacji, w których liczba prób nie jest ograniczona, robot osiągał regularnie najlepsze czasy przejazdów. Najszybszy osiągnięty czas to 5.67 sekundy, podczas gdy drugi najlepszy czas w eliminacjach wyniósł 5.74 sekundy. Dla porównania z rzeczywistym przebiegiem, mapy zebrane przez robota podczas mapowania i jednego z szybkich przejazdów zostały pokazane na rysunkach 5.17 i 5.18. Pierwsza z nich bardzo dobrze oddaje faktyczny wygląd mapy, natomiast na drugiej widać wpływ wynikających z dużej prędkości poślizgów, które zniekształcają mapę.

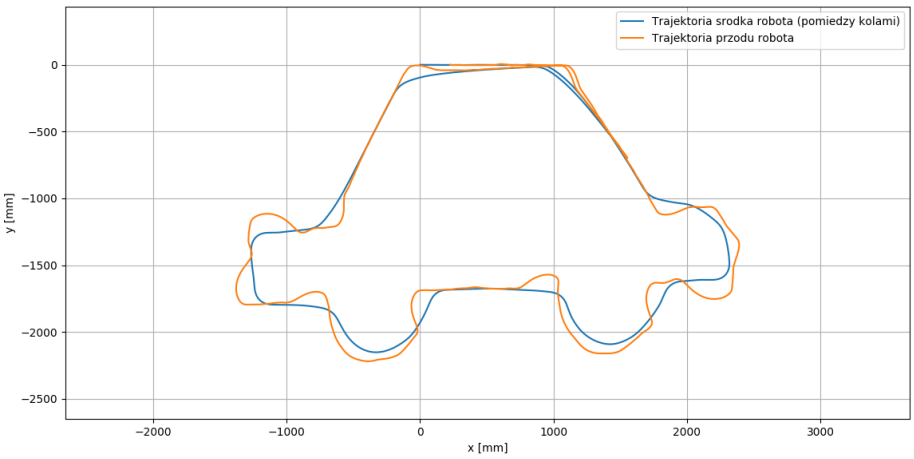
Etap finałowy tych zawodów uwidacznia jednak wszystkie wady algorytmu w jego obecnej wersji. Regulamin zawodów ogranicza liczbę prób podczas finałów do dwóch, co jest problematyczne w przypadku algorytmu wykorzystującego mapę, ponieważ pierwszy z przejazdów musi być przeznaczony na powolne zmapowanie, co wyklucza osiągnięcie krótkiego czasu przejazdu. Niestety, podczas regulaminowych prób, udało się jedynie zbudować mapę trasy, natomiast drugi przejazd, bazujący na tej mapie, był nieudany. Prze-



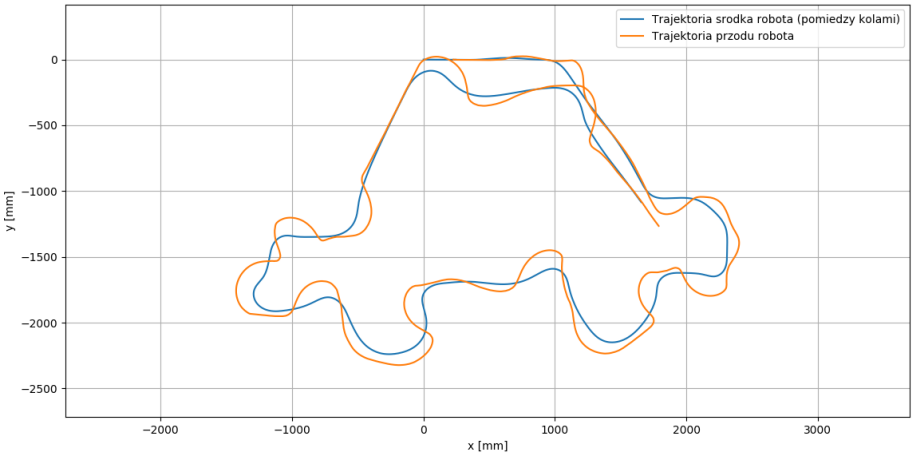
Rysunek 5.15: Trasa eliminacyjna podczas TTR 2017 w Gdańsku



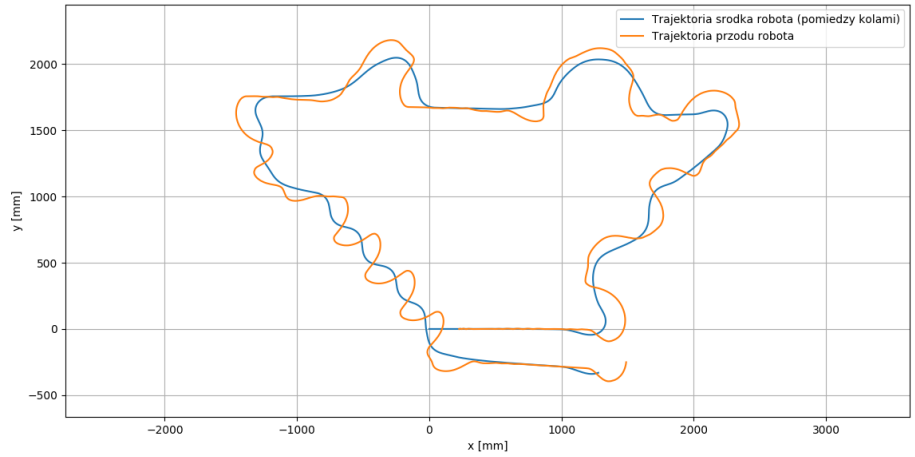
Rysunek 5.16: Trasa finałowa podczas TTR 2017 w Gdańsku



Rysunek 5.17: Mapa trasy eliminacyjnej zebrana podczas przejazdu wolnego — TTR 2017



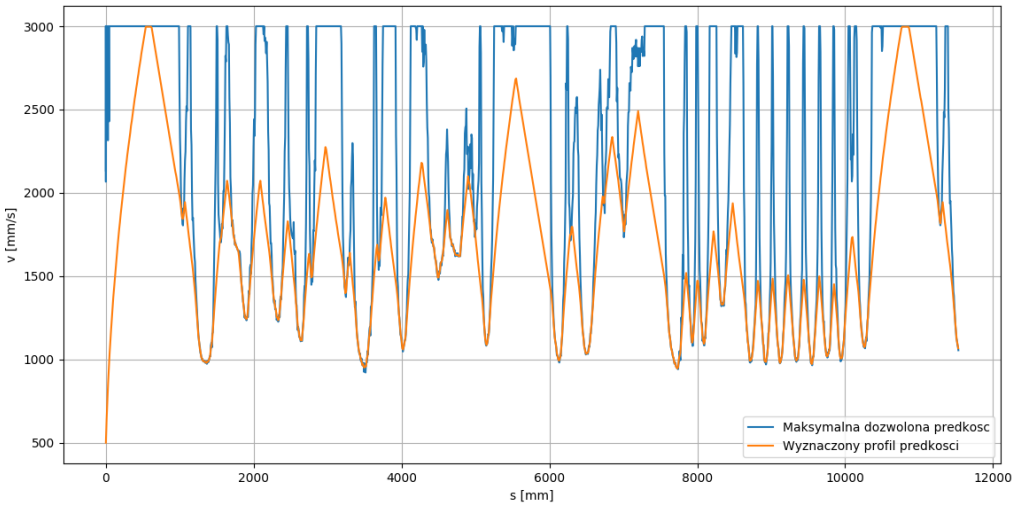
Rysunek 5.18: Mapa trasy eliminacyjnej zebrana podczas przejazdu szybkiego — TTR 2017



Rysunek 5.19: Mapa trasy finałowej zebrana podczas przejazdu wolnego — TTR 2017

jazd szybki z nadrzędnym profilerem prędkości, został przeprowadzony już po zawodach i pozwoliłby na osiągnięcie drugiego miejsca.

Mapa zebrana podczas przejazdu, zilustrowana na rysunku 5.19, pokazuje jak robot pokonywał trasę. Jak widać, jakość zebranej mapy jest niska, ze względu na dużą ilość zakrętów, kształt trasy odbiega od rzeczywistej dużo bardziej niż w eliminacjach. Na rysunku 5.20 został przedstawiony wyznaczony po mapowaniu profil prędkości z którą robot powinien poruszać się po trasie. Można zauważyć, że z powodu serii wąskich zakrętów na końcu trasy, prędkość ustawiona przez nadrzędny sterownik była na tym etapie na tyle niska, że niwelowała zysk osiągnięty na wcześniejszych odcinkach. Gorszy wynik tłumaczy również mała ilość odcinków prostych na których robot mógł osiągnąć prędkość maksymalną. Dla tego typu tras największą poprawę uzyskiwanych wyników można byłoby osiągnąć przy użyciu algorytmów optymalizujących trajektorię przejazdu.



Rysunek 5.20: Profil prędkości wyznaczony dla trasy finałowej — TTR 2017

Rozdział 6

Podsumowanie

W niniejszej pracy sprecyzowano zadanie stojące przed robotami klasy linefollower, które polega na przejechaniu określonej trasy w jak najkrótszym czasie, oraz przedstawiono różne sposoby jego realizacji. Praca miała na celu zebranie informacji na temat algorytmów śledzenia trasy oraz porównanie ze sobą ich własności. Zaproponowano systematykę algorytmów sterowania robotów mobilnych klasy linefollower oraz przedstawiono sposób działania każdego z prezentowanych algorytmów. Zbudowano robota tej klasy, a następnie zaimplementowano na nim wybrane algorytmy, które następnie przebadano w przygotowanym do tego celu środowisku.

Poza algorytmami stosowanymi typowo w konkurencji linefollower, przedstawiono możliwe rozwiązania pozwalające na poprawę uzyskiwanych wyników. Udowodniono, że w zakresie algorytmów śledzenia tras dla robotów linefollower są duże możliwości rozwoju, a poprawa rezultatów jest jak najbardziej możliwa. Oprócz dokonania opisu teoretycznego takich rozwiązań, opracowano oraz zaimplementowano na rzeczywistym robocie algorytm optymalizujący prędkość przejazdu na podstawie zebranej wcześniej mapy trasy.

Wykorzystanie informacji o przebiegu trasy do wyznaczenia optymalnego profilu prędkości znacząco skraca uzyskiwany czas przejazdu. Dzięki profilerowi prędkość robota nie jest ograniczona do najtrudniejszego elementu trasy, co ma miejsce w przypadku prostszych algorytmów, a jest dopasowana do kształtu aktualnie pokonywanego odcinka.

Algorytm śledzenia trasy z użyciem mapy jest najskuteczniejszy na trasach składających z długich odcinków prostych, na których robot może rozwijać duże prędkości. Korzyści wynikające z jego stosowania minimalizowane są jednak na wąskich i krętych trasach, gdzie nadrzędny sterownik stale ogranicza zadawaną prędkość. Stosunek długości prostych odcinków do długości całej trasy może więc służyć za kryterium wyboru algorytmu którego należy użyć, może się bowiem okazać, że na bardzo krętych trasach algorytm optymalizujący prędkość będzie osiągał gorsze wyniki niż prostsze algorytmy.

Podczas przejazdu, wraz z przejechanym dystansem, rzeczywiste położenie robota odbiega od estymowanego, przez co działający w robocie nadrzędny profiler dobiera prędkość niedopasowaną do kształtu trasy. W rezultacie robot może np. za szybko zwolnić przed zakrętem i zacząć ponownie przyspieszać jeszcze w trakcie jego pokonywania, co powoduje poślizgi i tym bardziej zwiększa błąd estymacji położenia lub nawet skutkuje wypadnięciem z trasy. Dlatego aby w pełni wykorzystać zyski płynące z użycia mapy, potrzebna jest korekcja estymacji położenia, wykonywana w czasie przejazdu. Powinna ona być na tyle dokładna, aby wyeliminować potrzebę nakładania znaczących ograniczeń na prędkość robota w czasie pokonywania zakrętów.

Potrzebę stosowania modułu korygującego oszacowanie pozycji można ograniczyć, stopniowo zwiększając prędkość zakładaną przez użyty w algorytmie profiler. Umożliwia

to stosowanie mapy uzyskanej podczas coraz szybszych prób, dzięki czemu błąd estymacji nie wpływa na jakość przejazdu. Zmniejsza to jednak konkurencyjność podczas zawodów robotycznych, ponieważ do osiągnięcia najlepszego rezultatu potrzeba kilku prób, a ich liczba w finale jest ograniczona. Innym rozwiązaniem może być próba budowy mapy na podstawie najszybszego przejazdu ze zwykłym regulatorem PD, a następnie wyznaczenie z niej profilu prędkości, podwyższając minimalną prędkość jaką może nałożyć profiler do prędkości ustawionej podczas przejazdu mapującego, co powinno prowadzić do osiągnięcia lepszego rezultatu.

Kolejnym krokiem, mającym na celu jeszcze lepszą realizację zadania śledzenia trasy, jest implementacja opisanego w pracy algorytmu optymalizującego trajektorię ruchu robota, który umożliwiłby poprawę wyników uzyskiwanych na krętych trasach z dużą ilością zakrętów, których sposób pokonywania można dodatkowo udoskonalić.

Bibliografia

[1] Festiwal Robotyki Cyberbot 2017 — Strona główna zawodów. http://cyberbot.put.poznan.pl. [dostęp online 13.06.2017].

[2] HC-06 Bluetooth module datasheet. https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf. [dostęp online 13.06.2017].

[3] Polska Unia Robotyki Turniejowej — regulamin linefollower. http://www.purt.treker.eu/sites/default/files/Linefollower.pdf. [dostęp online 13.06.2017].

[4] Robotic Arena 2016 — Regulamin konkurencji LineFollower Light. http://www.roboticarena.pl/static/Main/regulaminy/LF_Light.pdf. [dostęp online 13.06.2017].

[5] Trójmiejski Turniej Robotów 2017 — Strona główna zawodów. http://ttr.skalppg.pl/. [dostęp online 13.06.2017].

[6] AMS. AS5040 datasheet. https://ams.com/kor/content/download/1285/7214/file/AS5040_Datasheet_EN_v2.pdf. [dostęp online 13.06.2017].

[7] baton. Algorytm linefollowera w C – dla początkujących i nie tylko. http://forbot.pl/blog/artykuly/podstawy/algorytm-linefollowera-c-poczatkujacych-id2722. [dostęp online 13.06.2017].

[8] C. G. L. Bianco, M. Romano. Optimal Velocity Planning for Autonomous Vehicles Considering Curvature Constraints. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, strony 2706–2711, 2007.

[9] M. Botta, V. Gautieri, D. Loiacono, P. L. Lanzi. Evolving the Optimal Racing Line in a High-End Racing Game. *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, strony 108–115, 2012.

[10] F. Braghin, F. Cheli, S. Melzi, E. Sabbioni. Race Driver Model. *Computers & Structures*, 86:1503–1516, lipiec 2008.

[11] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, 2009.

[12] H. Cao, Y. Yin, D. Du, Y. He, H. Yu, W. Gu. Testing and Modeling of Motor Driver System Based on PD Control. *2006 6th World Congress on Intelligent Control and Automation*, wolumen 1, strony 1543–1547, 2006.

[13] D. Casanova, R. S. Sharp, P. Symonds. Minimum Time Manoeuvring: The Significance of Yaw Inertia. *Vehicle System Dynamics*, 34:77–115, 2000.

[14] M. Engin, D. Engin. Path Planning of Line Follower Robot. *2012 5th European DSP Education and Research Conference (EDERC)*, strony 1–5, 2012.

[15] G. Găşpăresc. PID Control of a DC Motor using Labview Interface for Embedded Platforms. *2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC)*, strony 145–148, 2016.

[16] P. Gui, L. Tang, S. Mukhopadhyay. MEMS Based IMU for Tilting Measurement: Comparison of Complementary and Kalman Filter Based Data Fusion. *2015 IEEE 10th Conference on Industrial Electronics and Applications*, strony 2004–2009, 2015.

[17] M. B. Hisham, i in. Template Matching using Sum of Squared Difference and Normalized Cross Correlation. *2015 IEEE Student Conference on Research and Development (SCOReD)*, strony 100–104, 2015.

[18] InvenSense. MPU-6050 datasheet. `https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf`. [dostęp online 13.06.2017].

[19] T. Januszewski. Metody reprezentacji i śledzenia trasy dla robota klasy linefollower. Projekt inżynierski, Politechnika Wrocławska, 2017.

[20] Kingbright. KTIR0711S datasheet. `http://msx.cal24.pl/boards/ktir0711s/KTIR0711S.pdf`. [dostęp online 13.06.2017].

[21] T. Rizano, D. Fontanelli, L. Palopoli, L. Pallottino, P. Salaris. Local Motion Planning for Robotic Race Cars. *52nd IEEE Conference on Decision and Control*, 2013.

[22] T. Rizano, i in. Global Path Planning for Competitive Robotic Cars. *52nd IEEE Conference on Decision and Control*, strony 4510–4516, 2013.

[23] K. Tchoń, A. Mazur, I. Dulęba, R. Hossa, R. Muszyński. *Manipulatory i roboty mobilne: modele, planowanie ruchu, sterowanie*. Problemy Współczesnej Nauki, Teoria i Zastosowania. Robotyka. Akademicka Oficyna Wydawnicza PLJ, 2000.

[24] TOSHIBA. TB6612FNG datasheet. `https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf`. [dostęp online 13.06.2017].

[25] Treker. Poznaj 8 sekretów szybkiego LineFollowera. `http://forbot.pl/blog/artykuly/teoria/poznaj-8-sekretow-szybkiego-linefollowera-id5482`. [dostęp online 13.06.2017].

[26] E. Velenis, P. Tsiotras. Minimum Time vs Maximum Exit Velocity Path Optimization During Cornering. *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, wolumen 1, strony 355–360, 2005.

[27] E. Velenis, P. Tsiotras. Optimal Velocity Profile Generation for Given Acceleration Limits: Theoretical Analysis. *Proceedings of the 2005, American Control Conference, 2005.*, wolumen 2, strony 1478–1483, 2005.

[28] P. Wang, K. Sekiyama. Curvature Based Velocity Control System for Mobile Robot. *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, strony 005211–005216, 2015.

[29] Y. Xiong. Racing Line Optimization. Praca magisterska, Shanghai Jiao Tong University, 2009.