Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej

Sterowanie i symulacja robotów

Sprawozdanie z bloku drugiego

Zespół:

Konrad Winnicki
Jakub Sikora

Prowadzący:

mgr. inż. Wojciech Dudek

Warszawa, 14 stycznia 2019

Spis treści

1. Laboratorium 1

1.1. Znalezione błędy w funkcjonowaniu systemu robota

W trakcie zajęć po wprowadzeniu przez prowadzącego kilku modyfikacji, system robota przestał działać poprawnie. Po zadaniu robotowi prędkości, robot zatrzymywał się i ruszał w bardzo dziwny i niezrozumiały sposób. Naszym zadaniem było zlokalizować usterki za pomocą narzędzi systemu ROS.

1.1.1. Wykorzystane narzędzia

Do zdiagnozowania usterki systemu wykorzystaliśmy następujące narzędzia.

`rqt_graph`

Narzędzie `rqt_graph` służy do wizualizacji struktury systemu. Pozwala na śledzenie topologii komunikacji pomiędzy poszczególnymi węzłami. Za pomocą tego narzędzia mogliśmy sprawdzić czy zaburzona została komunikacja pomiędzy węzłami oraz czy nie dodano nowego węzła wprowadzającego zaburzenia.

`rqt_tf_tree`

Narzędzie `rqt_tf_tree` pozwala na wizualizację drzewa transformacji w systemie robota. Program pozwolił nam na sprawdzenie czy publikowane są wszystkie transformację w systemie, pozwalające na poprawne wykonywanie obliczeń w systemie.

`rostopic`

Program `rostopic` jest narzędziem do sprawdzenia informacji na temat ROS Topiców. Narzędzie to pozwala na wypisywanie wiadomości nadawanych na zadanym kanale, znajdywania wszystkich topiców czy publikowanie nowych wiadomości.

`rqt_plot`

Narzędzie `rqt_plot` pozwala na wizualizację danych z wiadomości na kolorowych wykresach. Zwykle używane jako pierwsze narzędzie po które sięgamy w trakcie pracy z systemem, to jednak w zadaniu lokalizacji usterki okazał się bezużyteczny.

1.1.2. Zlokalizowane usterki

Za pomocą `rqt_tf_tree` udało nam się ustalić że brakuje transformacji z układu bazy robota do układu wieży. Brak tego przekształcenia nie pozwalał chociażby na poprawną wizualizację robota w programie `rviz`.

Znacznie poważniejszą usterką wprowadzoną przez prowadzącego było dołączenie do systemu nowego węzła nadającego na topic `mux_vel_nav/cmd_vel`. Anonimowy węzeł udało nam się zlokalizować za pomocą narzędzia `rqt_graph`. Po podsłuchaniu niechcianego gościa za pomocą polecenia `rostopic echo` udało nam się ustalić że nowy węzeł nadaje zerowe prędkości. Wyjaśnia to zrywane ruchy robota, który najpierw otrzymywał niezerowe wartości prędkości aby następnie otrzymać polecenie zatrzymania się.

1.2. Opis algorytmu interpolacji liniowej

Głównym zadaniem do zrealizowania w trakcie laboratorium pierwszego było napisanie węzłów implementujących względne sterowanie pozycyjne. Należało zrealizować zadanie w dwóch wariantach.

1.2.1. Interpolacja na podstawie zadawanych prędkości

W pierwszej wersji, interpolowaliśmy punkty wyłacznie za pomocą zadawanych prędkości. Węzeł był inicjalizowany w punkcie (0,0,0) a kolejne sterowania miały być wyznaczane na podstawie domniemanej idealnie zrealizowanej poprzedniej akcji ruchu.

Zasada działania algorytmu wydaje się banalnie prosta. System sterowania robota cały czas zapamiętuje pozycję między akcjami. Po przyjściu nowego polecenia, wyznaczane są przemieszczenia o które należy przesunąć robota. Robot najpierw obraca się w stronę zadanego punktu, następnie jedzie do przodu aż do wyznaczonego miejsca aby ostatecznie obrócić się do zadanego kąta obrotu.

Najciekawszym z punktu widzenia projektowania algorytmu lokomocji wydaje się sposób interpolacji położenia. Robot poruszając się do przodu zna odległość swoją od położenia zadanego oraz prędkość jaką zadaje na koła. Korzystając z prostego wzoru na drogę:

$$s = vt$$

możemy w prostu sposób wyznaczyć przez ile sekund należy jechać/obracać się aby osiągnąć cel. W ten sam sposób interpolowaliśmy pozycję kątową robota. Korzystając z zależności

$$\alpha = \omega t$$

udało nam się w prosty sposób uzyskać zadany kąt. Z podanego sposobu korzystamy na początku ruchu aby obrócić robota w stronę pozycji zadanej i na końcu ruchu aby obrócić robota do docelowego kąta zadanego.

Korzystając z tej metody, udało nam się uzyskać wyniki które można opisać jako wysoko niezadowalające. Nawigacja robotem w ten sposób ma wiele wad. Jest bardzo niedokładna a błąd bardzo szybko narasta, co czyni wykonywanie złożonych zadań przemieszczania się niemożliwymi do wykonania. Jedyną zaletą tej metody, jest jej prostota. Można ją wykorzystać do zgrubnego przemieszczania się robota, w przypadku gdy podłoże nie jest śliskie oraz nie jest dostępna informacja z enkoderów.

Implementacja

Zgodnie z przyjętą konwencją nazewniczą, węzeł wykonujący interpolację na podstawie zadawanych prędkości nazwaliśmy `normalmente_movimiento.py`. Węzeł nasłuchuje nowej pozycji zadanej na topicu `new_pose`, który przyjmuje wiadomości typu `turtlesim/Pose`. Polecenie ruchu można wysłać używając przykładowego polecenia:

```
rostopic pub /new_pose turtlesim/Pose 'x: 2.0, y: 4.0, theta: 0'
```

po którym węzeł zacznie wysyłać wiadomości typu `geometry/Twist` na topic `mux_vel_nav/cmd_vel`.

1.2.2. Interpolacja na podstawie danych odometrii

W drugiej wersji węzła, implementowaliśmy interpolację liniową punktów na podstawie danych z odometrii. Informacja zwrotna na temat zrealizowanego położenia powinna pozwolić na znaczącą poprawę dokładności przemieszczania robota.

Przed wyjaśnieniem zasady działania algorytmu, należy wyjaśnić czym dokładnie jest odometria. Najogólniej rzecz biorąć, odometria to dział miernictwa, który zajmuje się pomiarem odległości, wykorzystującym czujniki które określają przemieszczenie względem pozycji początkowej.

W przypadku systemu robotycznego Elektron, do dyspozycji mamy dwa rodzaje czujników. Pierwszym są enkodery umieszczone na kołach robota, które pozwalają na pomiar o ile obróciło się koło, co przy znanym promieniu powinno z pewną precyzją zwrócić nam przemieszczenie. Drugą możliwością, jest zastosowanie czujnika laserowego do pomiaru przemieszczenia względem otoczenia.

Zasada działania algortymu nie jest specjalnie skomplikowana. Na początku obracamy robota, aż jego orientacja wynikająca z odometrii będzie zgodna (z zadaną dokładnością) z orientacją wymaganą do jazdy w kierunku zadanego punktu. Następnie robot jedzie prosto do momentu aż jego pozycja wynikająca z odometrii będzie zgodna z pozycją zadaną. Ostatecznie, ponownie obracamy robota do zadanej orientacji w ten sam sposób w jaki to robiliśmy w pierwszej fazie ruchu.

Głównym czynnikiem wpływającym na jakość działania węzła jest jakość samego sygnału z odometrii. W przypadku kierowania się sygnałem pozycji pochodzącym z enkoderów umieszczonych na kołach, który jest podatny na błędy z powodu poślizgów i skończonej rozdzielczości samego enkodera, możemy nie uzyskać dokładnego odwzorowania pozycji zadanej w rzeczywistą pozycję robota. W celu poprawienia jakości sygnału z odometrii należy skorzystać z czujnika laserowego LIDAR, który zwraca położenie na podstawie przemieszczenia względem otoczenia.

Implementacja

W celu wykonania zadania napisaliśmy skrypt `odometria_movimiento.py`, który uruchamia węzeł poruszający robotem z uwzględeniem informacji z odometrii. Podobnie jak w poprzednim zadaniu robot nasłuchuje nowej pozycji zadanej na temacie `new_pose`. Dodatkowo, umożliwiliśmy zadawanie nowej pozycji jako `rosservice`. Użytkownik wysyła żądanie przemieszczenia do usługi o nazwie `stero/go_to_stpt`. Aby poprawnie wywołać usługę, należy wysłać wiadomość zdefiniowanego przez nas typu `stero_mobile_init/STPT`. Węzeł odczytuje informacje z odometrii z tematu `/elektron/mobile_base_controller/odom`. Wiadomości nadawane na tym temacie są typu `nav/Odometry`.

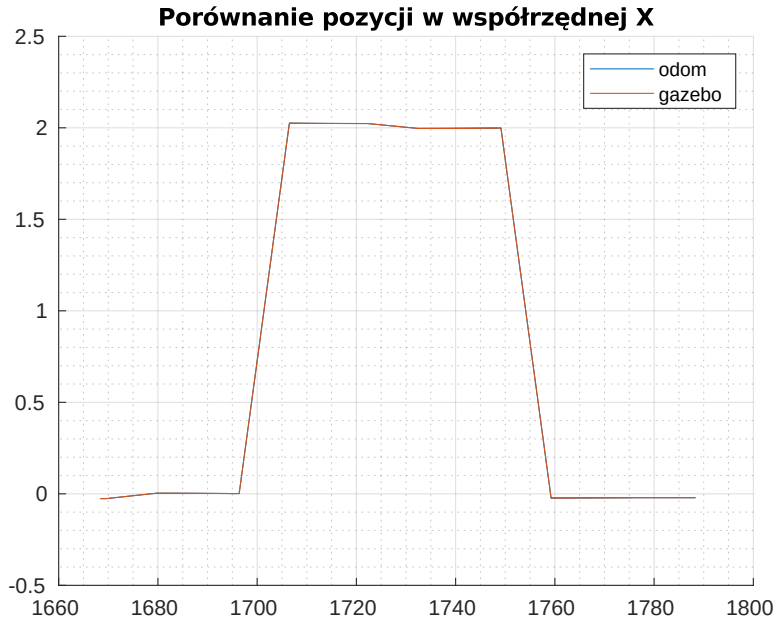
Po poprawnym zgłoszeniu żądania przemieszczenia, robot podobnie jak w przypadku poprzedniego węzła, wykonuje trzystopniowe przemieszczenie się do pozycji zadanej:

— Obrót w stronę pozycji zadanej
— Ruch po prostej do punktu zadanego
— Obrót do zadanej orientacji

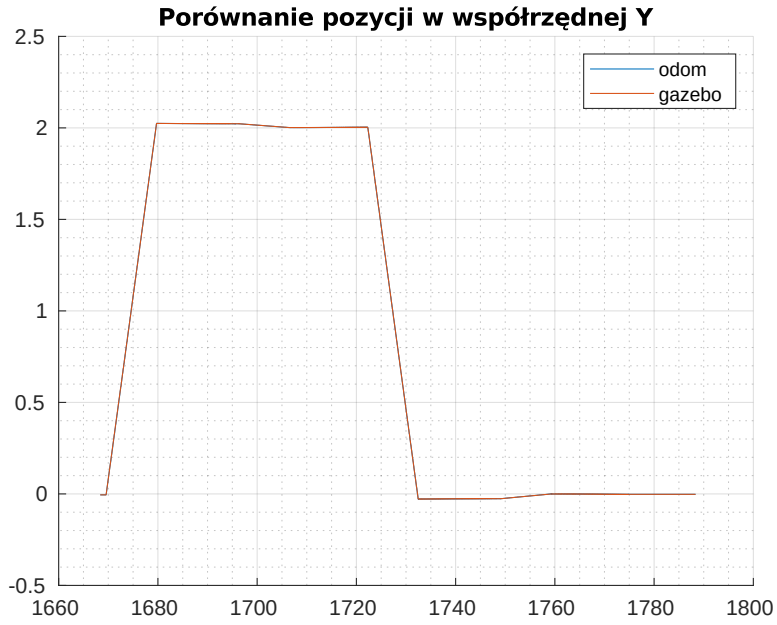
Każda z faz jest wykonywana do momentu gdy pozycja z odometrii będzie zgodna z pozycją zadaną. Ponieważ pozycja jest zwracana w pewnych odstępach, przemieszczenia i obroty wykonujemy do momentu aż różnica pozycji odometrii z pozycją zadaną będzie mniejsza od zadanego poziomu dokładności.

1.3. Porównanie odometrii z pozycją referencyjną

1.3.1. Test kwadratu zgodnie z ruchem wskazówek zegara

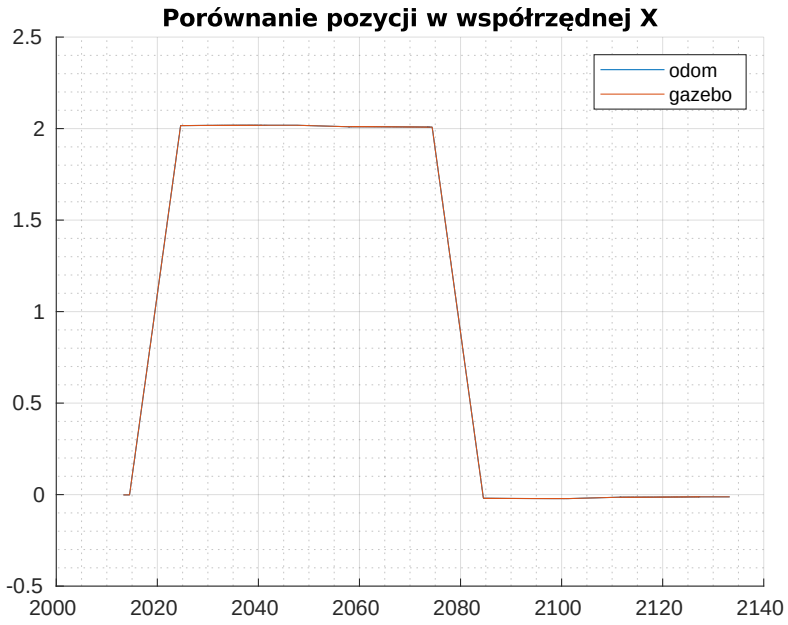


Rysunek 1.1. Pozycja we współrzędnej x osiągana przez robota w trakcie testu kwadratu (zgodnie z ruchem wskazówek zegara)

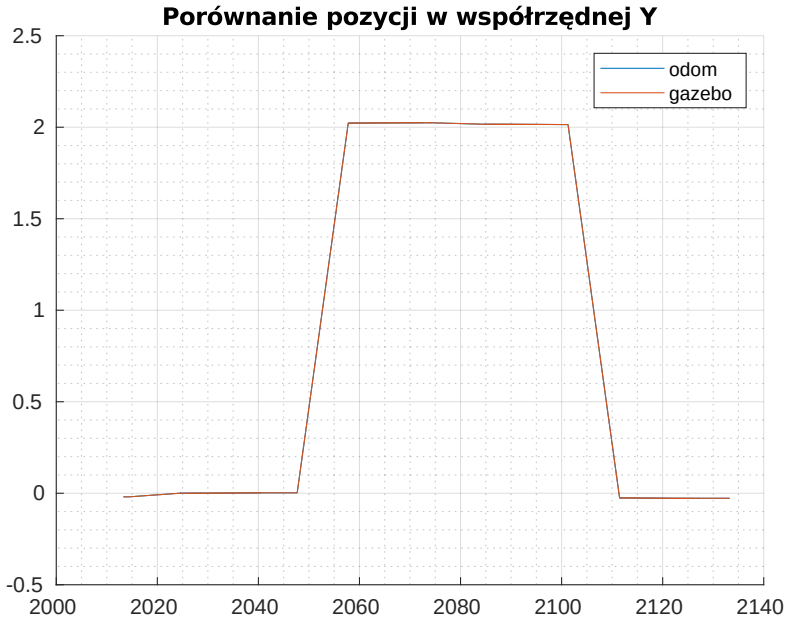


Rysunek 1.2. Pozycja we współrzędnej y osiągana przez robota w trakcie testu kwadratu (zgodnie z ruchem wskazówek zegara)

1.3.2. Test kwadratu przeciwnie do ruchu wskazówek zegara



Rysunek 1.3. Pozycja we współrzędnej x osiągana przez robota w trakcie testu kwadratu (przeciwnie do ruchu wskazówek zegara)



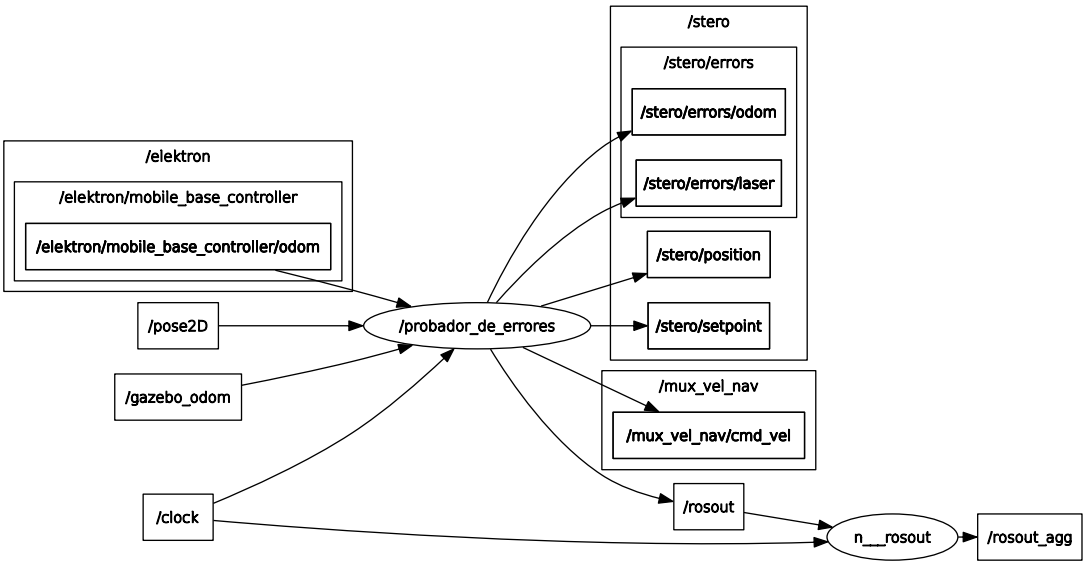
Rysunek 1.4. Pozycja we współrzędnej y osiągana przez robota w trakcie testu kwadratu (przeciwnie do ruchu wskazówek zegara)

Pozycja z enkoderów zaskakująco dobrze odwzorowuje aktualną pozycję. Oba pomiary (odom - enkodery, gazebo - pozycja w symulacji) praktycznie pokrywają się, co pozwala na precyzyjne sterowanie robotem w symulacji. Tak dobra jakość sygnału z odometrii spowodowana jest zastosowaniem lepszego sterownika `diff_drive_controller`s.

2. Projekt 1

W ramach pierwszego projektu w bloku, naszym zadaniem było zebranie danych z odometrii z różnych kontrolerów oraz dokonanie ich porównania. Do akwizycji danych należało zaimplementować dodatkowy węzeł ROSa.

2.1. Struktura oprogramowania stworzonego do zbierania danych



Rysunek 2.1. Struktura systemu akwizycji danych z odometrii

Zgodnie z przyjętą konwencją nazewniczą, węzeł zajmujący się równocześnie akwizycją i sterowanie robota został nazwany `probador_de_errores`. Uruchomienie skryptu powodowało pojawienie się następującego komunikatu:

```
Bienvenidos!
Sterowanie i Symulacja Robotow. Mini-projekt 3.
Wykonanie: Konrad Winnicki & Jakub Sikora
Dostepne argumenty wywolania:

-l, -line: test jazdy po linii
-c, -circle: test obrotu
-s, -square: test jazdy po kwadracie
```

Ponowne uruchomienie programu z odpowiednim argumentem powodowało rozpoczęcie testu. Przed testem należało upewnić się że robot znajduje się w pozycji początkowej.

2.2. Opis działania węzła zbierającego dane

Głównym zadaniem węzła była akwizycja danych z odometrii oraz publikacja danych.

2.2.1. Subskrypcje tematów

Węzeł jawnie subskrybował trzy tematy:

— `/elekton/mobile_base_controller/odom` - temat z pozycją uzyskaną z enkoderów na kołach
— `/pose2D` - temat z pozycją uzyskaną z czujnika laserowego
— `/gazebo_odom` - temat z pozycją referencyjną uzyskaną z symulatora

W węźle zaimplementowalismy funkcje reagujące na przyjście nowej wiadomości, które wyciągały interesujące nas dane i zapisywały je do globalnego stanu węzła.

2.2.2. Rozgłaszanie na temacie

Węzeł wysyła wiadomości na następujących tematach:
— `/stero/errors/odom` - na tym temacie jest nadawany aktualny błąd odometrii z enkoderów wyliczony na podstawie referencyjnego pomiaru pozycji z tematu `gazebo_odom`
— `/stero/errors/laser` - na tym temacie jest nadawany aktualny błąd odometrii z czujnika laserowego wyliczony na podstawie referencyjnego pomiaru pozycji z tematu `gazebo_odom`
— `/stero/position` - temat na której robot nadaje pozycję w której myśli że aktualnie jest, w zależności od wybranego sprzężenia (pozycja od enkoderów/pozycja z lasera)
— `/stero/setpoint` - temat z zadaną pozycją do której robot aktualnie dąży, ostatecznie nie został wykorzystany w projekcie, zaimplementowany z powodu pierwotnego złego zrozumienia zadania

Co każdy obieg pętli sterującej robotem, system rozgłasza nowe wiadomości na wyżej wymienionych tematach. Dzięki temu mogliśmy w prosty sposób za pomocą narzędzia `rosbag` nagrać wiadomości do późniejszej analizy.

2.3. Opis działania węzła sterującego robota

Węzeł `probador_de_errores` zajmuje się również sterowaniem robota w ramach przeprowadzanego testu. Do wyboru są trzy testy:

— test jazdy na wprost
— test obrotu
— test jazdy po kwadracie

Każdy z testów przeprowadzany jest na zasadzie podobnej jak do tej przedstawionej w 1.2.1.

2.4. Sposób analizy danych

Po przeprowadzeniu testów, uzyskaliśmy bardzo dużo danych do analizy. Zdecydowaliśmy się na skorzystanie z z programu MATLAB. Program oferuje możliwość instalacji pakietu `Robotics System Toolbox`, który posiada narzędzia do analizy danych zagregowanych w plikach *.bag* z wyszczególnieniem poszczelnych tematów. W pierwszej kolejności należy stworzyć obiekt reprezentujący plik *.bag* za pomocą polecenia

```
bag = rosbag(path_to_rosbag_file)
```

Korzystając z otrzymanego obiektu, możemy wyciągać podobiekty tematów na podstawie ich nazwy za pomocą następującego polecenia

```
odom = select(bag,'Topic','/elektron/mobile_base_controller/odom')
```

Wykorzystując obiekt tematu, w prosty sposób można wyodrębnić obiektu typu `timeseries`, które w bardzo prosty sposób można przedstawiać na wykresach, dzięki temu że wiążą dyskretną wartość wyjścia z dyskretnym czasem próbkowania. Obiekty te tworzymy za pomocą polecenia

```
ts_odom_X = timeseries(odom, 'Pose.Pose.Position.X');
```

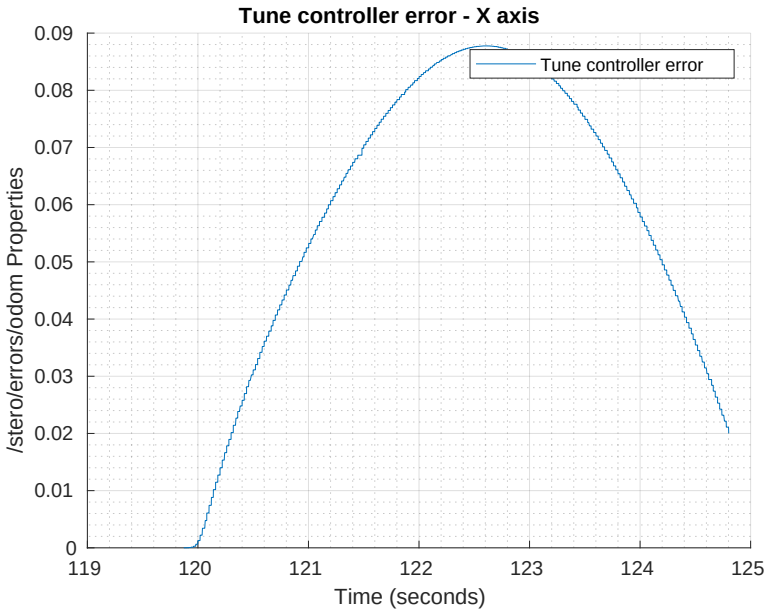
Mając już obiekt typu `typeseries`, pozostało już tylko stworzyć wykres i zapisać go do dalszej analizy. Wykres tworzymy za pomocą standardowego polecenia

```
plot(ts_odom_X)
```

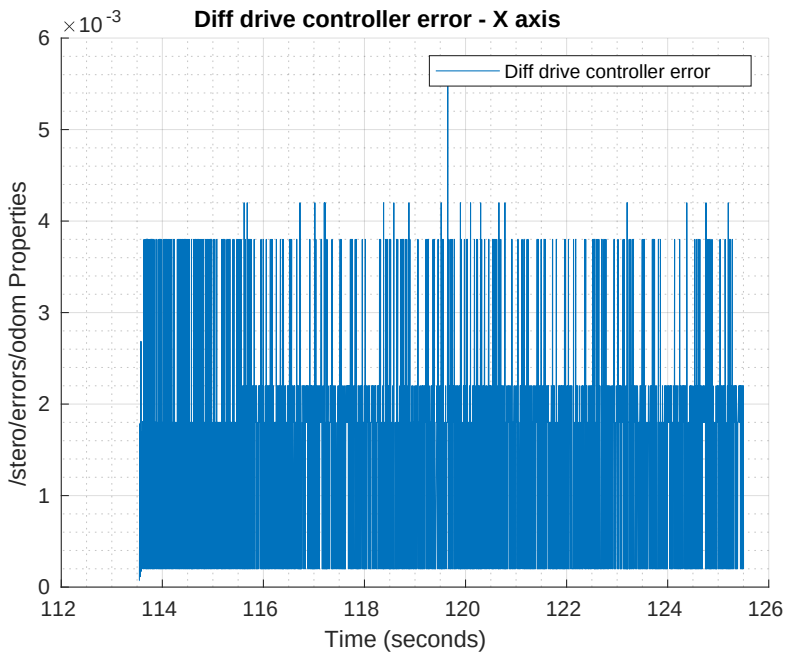
Korzystając z wyżej wymienionych poleceń, udało nam się napisać skrypt `bag_analyzer.m`, który automatycznie generuje wykresy błędów i pozycji we wszystkich współrzędnych z podanego rosbaga.

2.5. Wykresy i wnioski

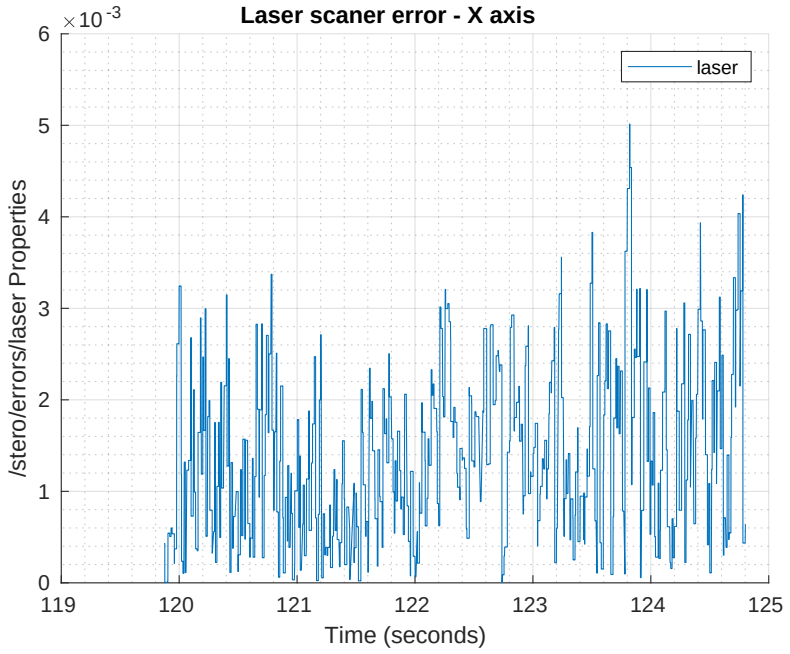
2.5.1. Test jazdy na wprost



Rysunek 2.2. Błąd odometrii we współrzędnej x przy sterowaniu za pomocą `tune_controller` w teście jazdy na wprost



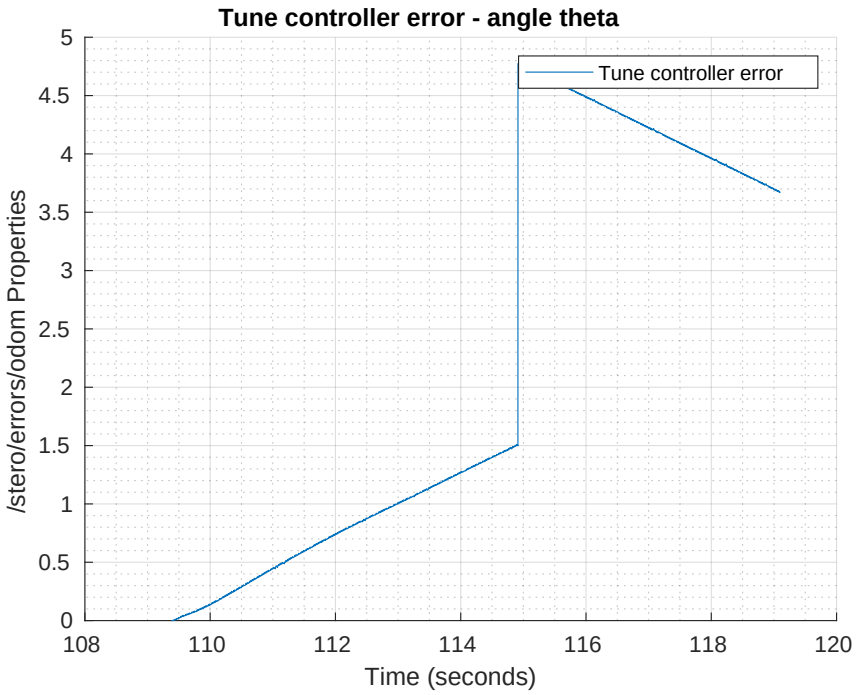
Rysunek 2.3. Błąd odometrii we współrzędnej x przy sterowaniu za pomocą `diff_drive_controller` w teście jazdy na wprost



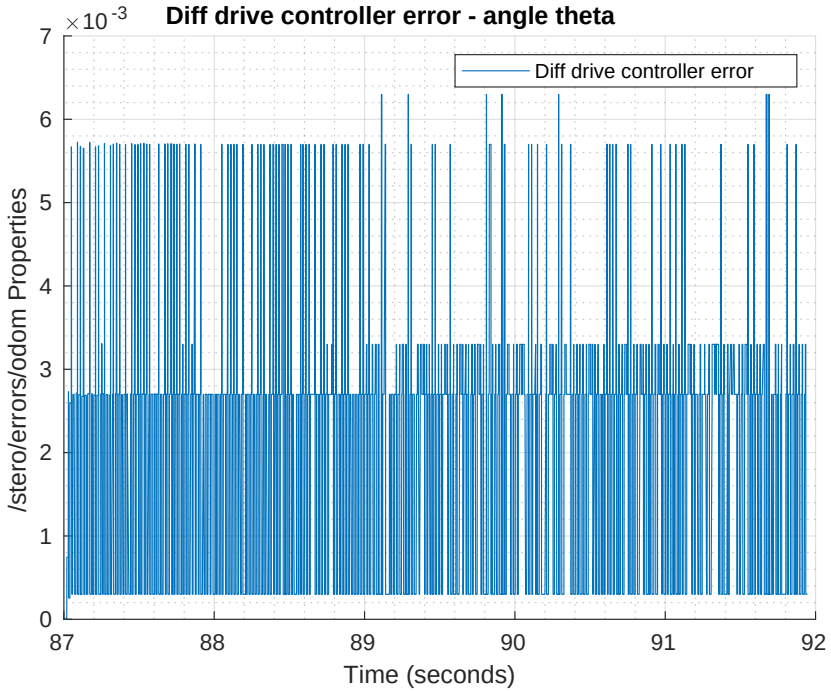
Rysunek 2.4. Błąd odometrii we współrzędnej x uzyskany z czujnika laserowego w teście jazdy na wprost

Na podstawie wyżej przedstawionych wykresów, można jednoznacznie stwierdzić że sterownik `tune_controller` w trakcie szybkiej jazdy na wprost gubi pozycję, szczególnie w trakcie startowania. W trakcie zatrzymywania się, błąd zmalał. Mimo wszystko, błąd pozycji jest znaczny, szczególnie w porównaniu z błędami sterownika `diff_drive_controller` i skanera laserowego. W trakcie jazdy na wprost, błędy drugiego sterownika i skanera laserowego pozostają na porównywalnym poziomie i są rzędu milimetrów.

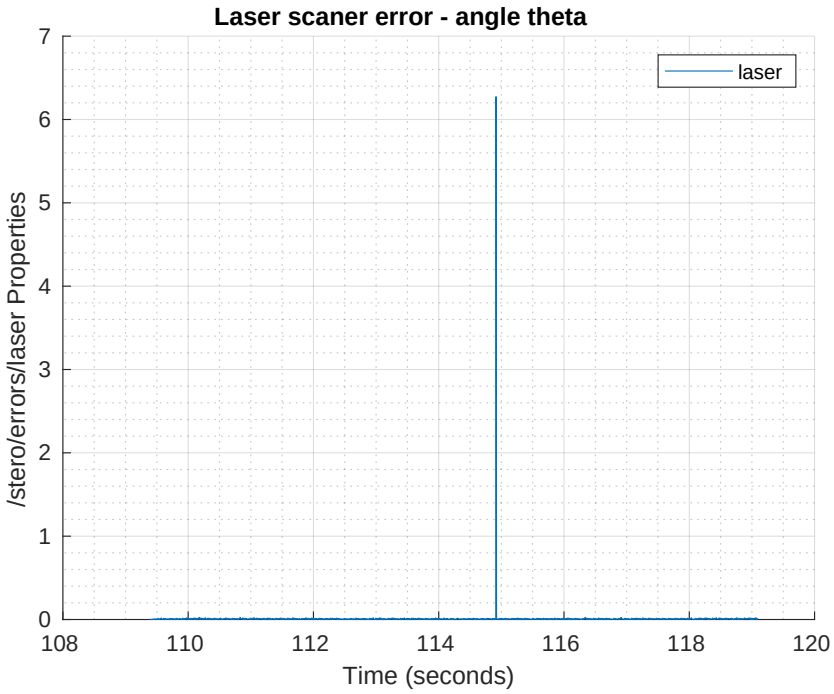
2.5.2. Test obrotu



Rysunek 2.5. Błąd odometrii we współrzędnej θ przy sterowaniu za pomocą `tune_controller` w teście obrotu



Rysunek 2.6. Błąd odometrii we współrzędnej θ przy sterowaniu za pomocą `diff_drive_controller` w teście obrotu

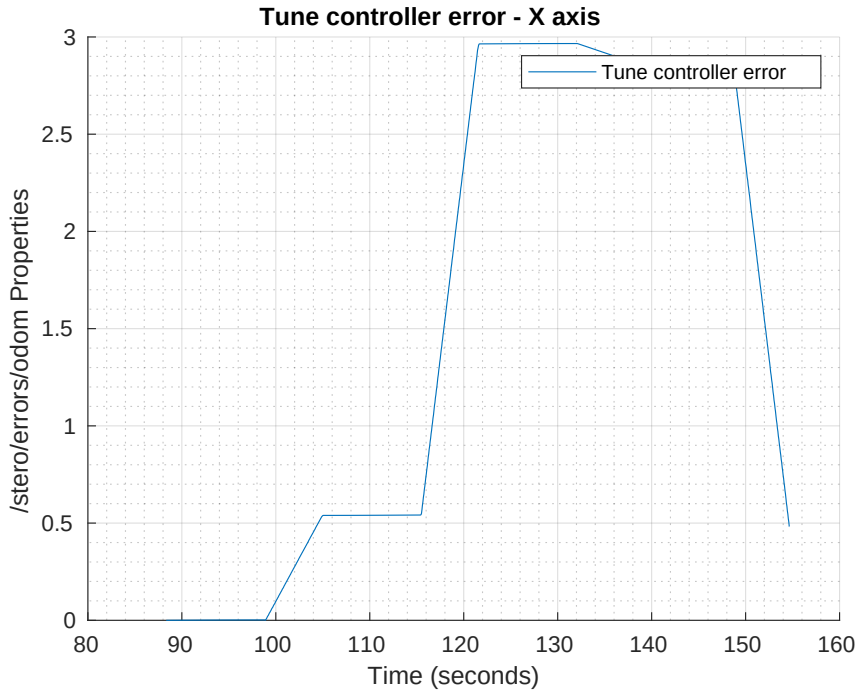


Rysunek 2.7. Błąd odometrii we współrzędnej θ uzyskany z czujnika laserowego w teście obrotu

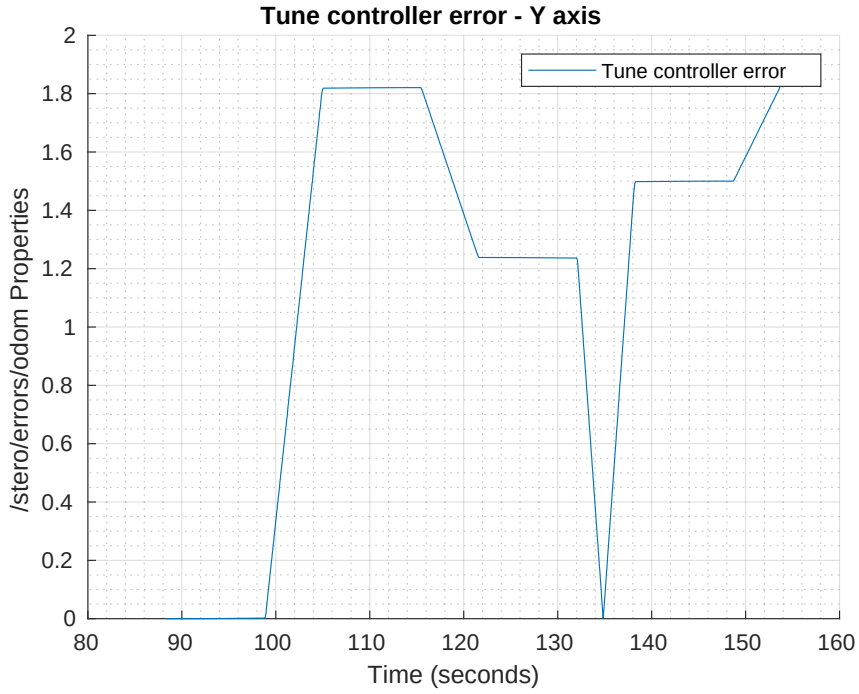
W przypadku testu obrotu, ponownie `tune_controller` wypadł najsłabiej. W trakcie obracania się, sterownik gubi kąt. Wynika to ze sposobu obracania się robota o bazie różnicowej. Pozostałe dwa sygnały utrzymują się w granicach akceptowalnych błędów, oscylując około wartości co odpowiada $0,1719$ stopnia. Charakterystyczne piki wynikają ze sposobu normalizacji kąta obrotu.

2.5.3. Test kwadratu

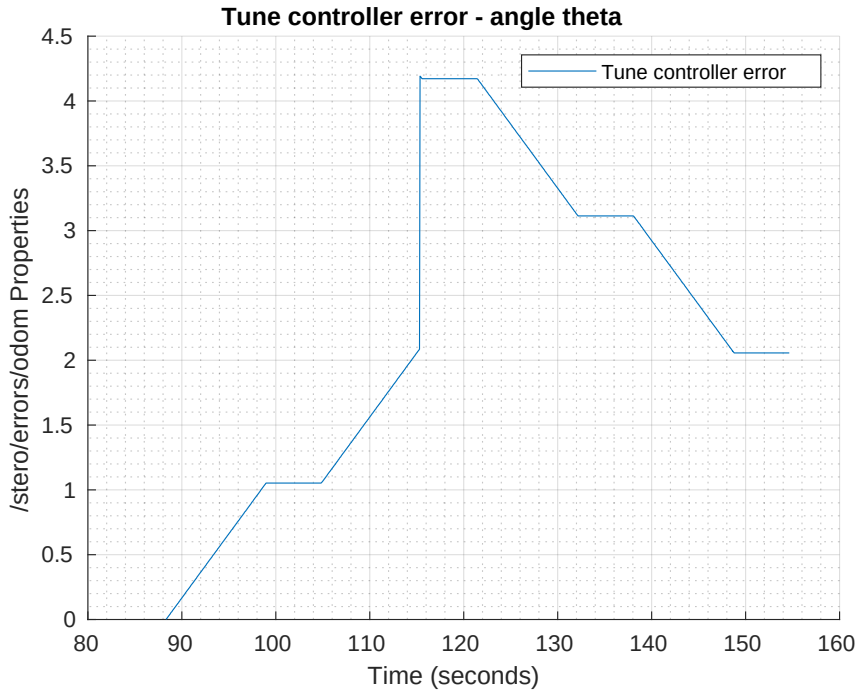
`tune_controller`



Rysunek 2.8. Błąd odometrii we współrzędnej x przy sterowaniu za pomocą `tune_controller` w teście kwadratu

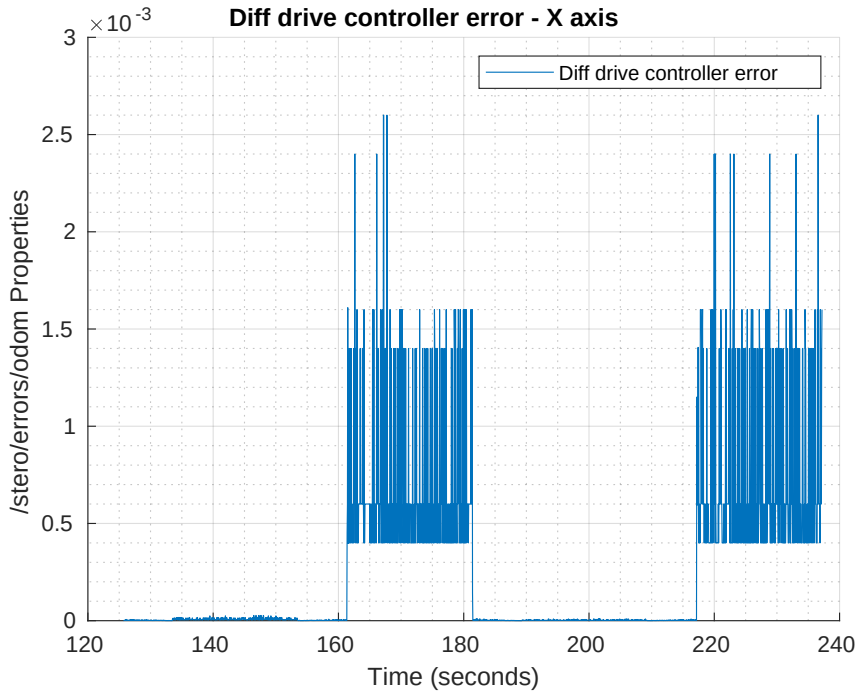


Rysunek 2.9. Błąd odometrii we współrzędnej y przy sterowaniu za pomocą `tune_controller` w teście kwadratu

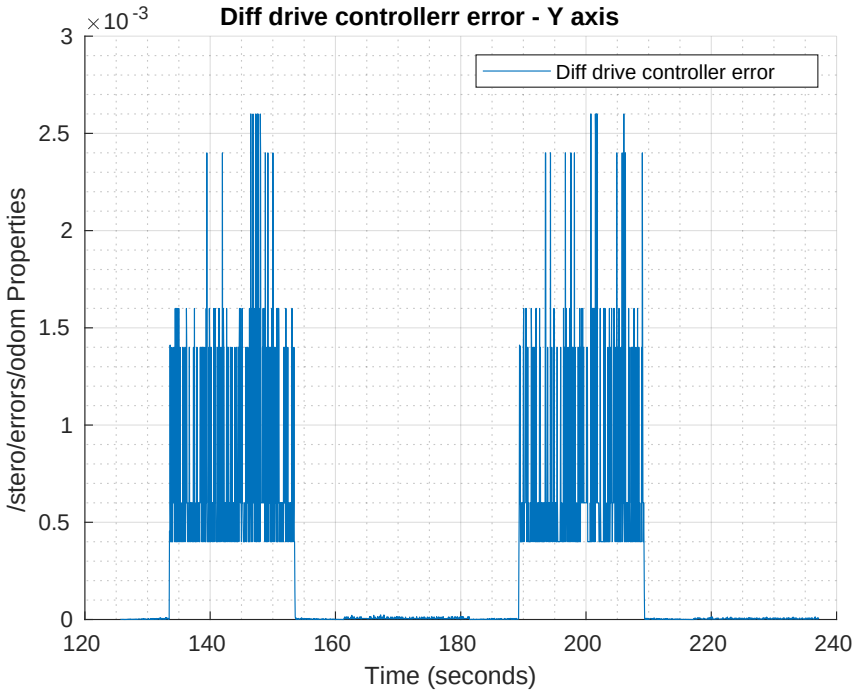


Rysunek 2.10. Błąd odometrii we współrzędnej θ przy sterowaniu za pomocą `tune_controller` w teście kwadratu

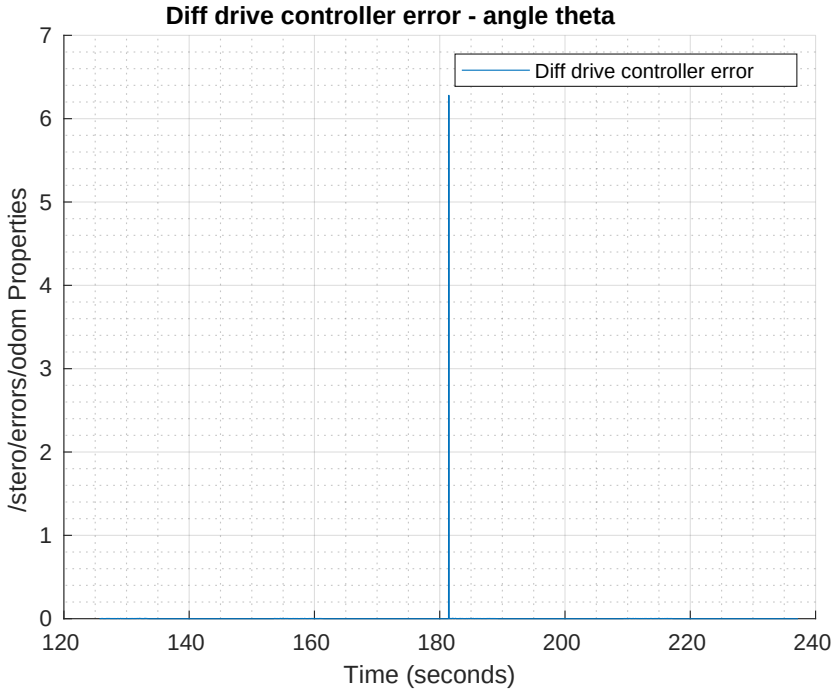
`diff_drive_controller`



Rysunek 2.11. Błąd odometrii we współrzędnej x przy sterowaniu za pomocą `diff_drive_controller` w teście obrotu

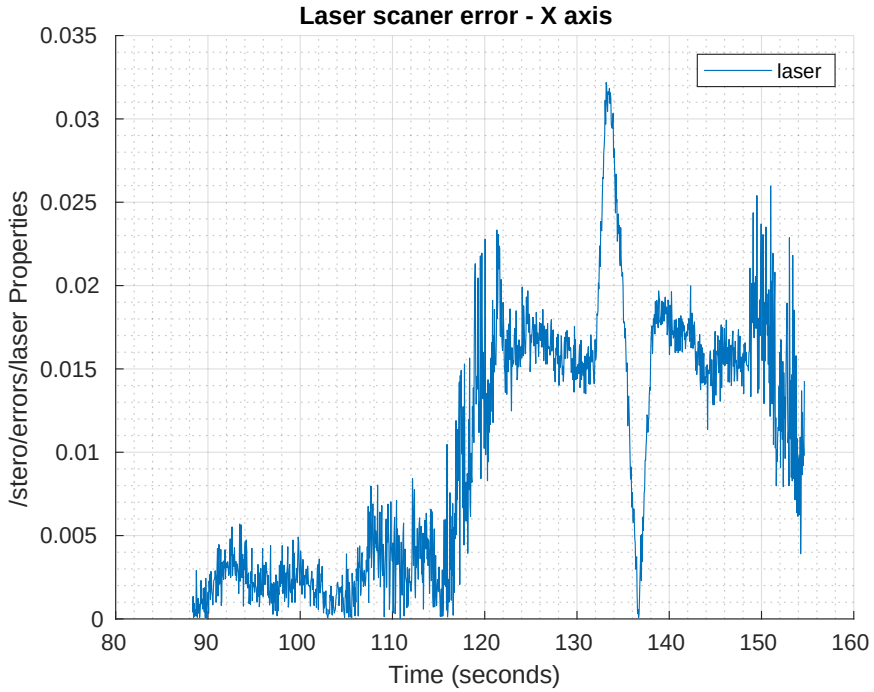


Rysunek 2.12. Błąd odometrii we współrzędnej y przy sterowaniu za pomocą `diff_drive_controller` w teście obrotu

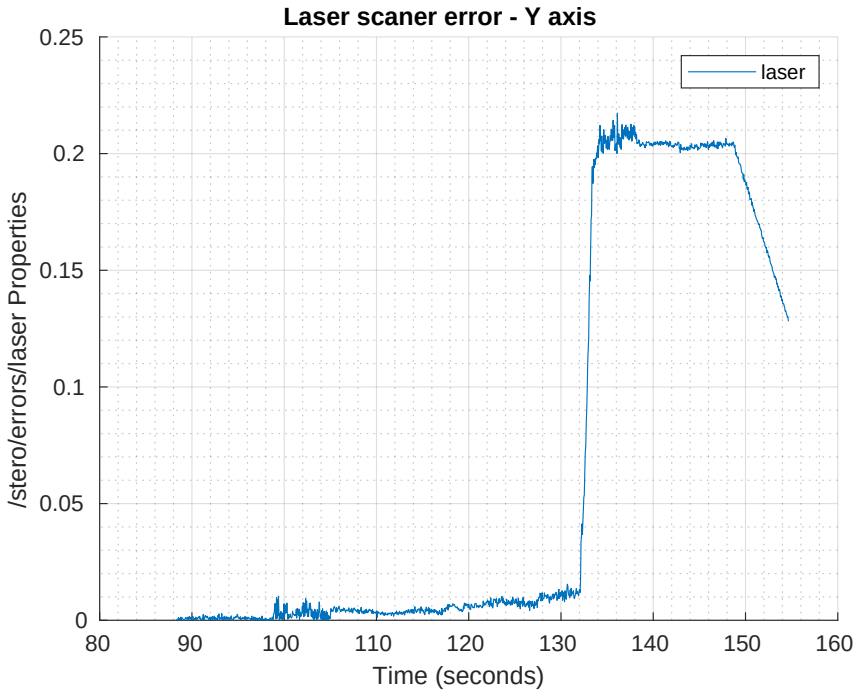


Rysunek 2.13. Błąd odometrii we współrzędnej θ przy sterowaniu za pomocą `diff_drive_controller` w teście obrotu

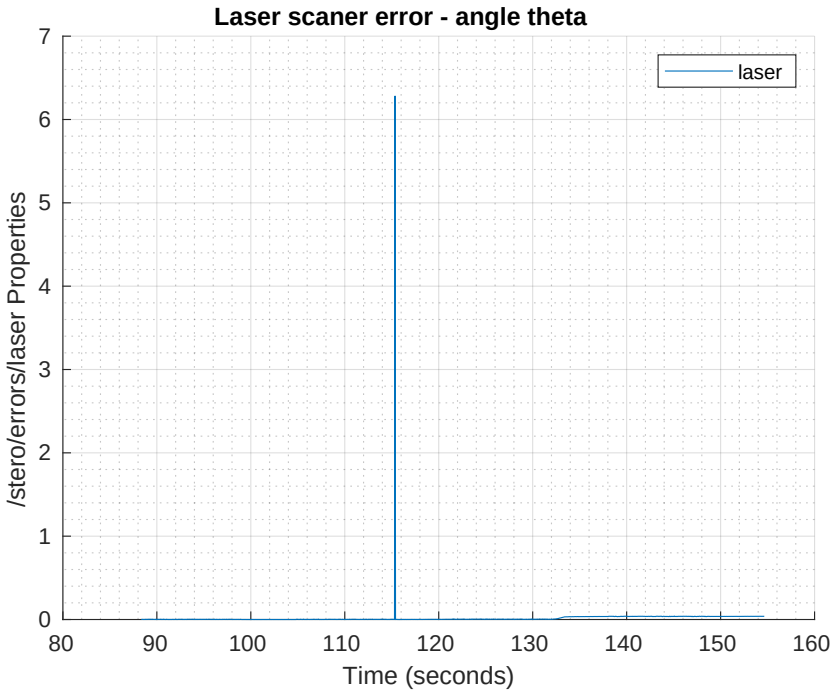
`laser_scan_matcher`



Rysunek 2.14. Błąd odometrii we współrzędnej x uzyskany z czujnika laserowego w teście kwadratu



Rysunek 2.15. Błąd odometrii we współrzędnej y uzyskany z czujnika laserowego w teście kwadratu



Rysunek 2.16. Błąd odometrii we współrzędnej θ uzyskany z czujnika laserowego w teście kwadratu

Wnioski

Na podstawie wyżej przedstawionych wykresów można jednoznacznie stwierdzić że sterownik `diff_drive_controller` jest zdecydowanie lepszy od sterownika `tune_controller`. Dzieje się tak ponieważ sterownik `tune_controller` nie został odpowiednio dostrojony. Poprawę jakości odometrii z tego sterownika można osiągnąć poprzez poprawienie konfiguracji parametrów, co zostało pozostawione ochotnikom (z racji że czas przed sesją jest niezwykle cennym surowcem, szczególnie na tym wydziale, tym razem wyjątkowo ochotnikami nie zostaliśmy).

Pomiary pozycji względnej za pomocą czujnika laserowego LIDAR uzyskują zaskakująco dobrą dokładność. Zmierzone błędy w złożonym zadaniu jazdy po kwadracie utrzymują się poniżej rozsądnego poziomu. Duży skok błędu wzdłuż współrzędnej y wynika z tego że w tej osi nie było żadnych ścian w pobliżu robota, co sprawiło że pomiar był mniej dokładny. Takie ściany robot miał przed sobą wzdłuż osi x, co pozwoliło na osiągnięcie znacznie mniejszych błędów.

W przypadku tak bogatej platformy jaką jest robot Elektron, najlepsze rezultaty możemy osiągnąć poprzez wsparcie sterownika `diff_drive_controller` pomiarami ze skanera laserowego. Połączenie danych z obu źródeł ma potencjał aby stać się źródłem bardzo dokładnego wskaźnika względnej pozycji, co jest kluczowe w skutecznym nawigowaniu robotem mobilnym o tak skomplikowanej bazie.

3. Laboratorium 2

3.1. Stworzone środowisko i jego mapa

3.2. Przykładowe ścieżki zaplanowane w środowiskach

3.3. Pliki uruchomieniowe symulacji

4. Projekt 2

4.1. Struktura sterownika robota

4.2. Opis działania węzła planującego

4.3. Pliki konfiguracyjne map kosztów oraz lokalnego planera

4.4. Wyjaśnienie zastosowanych parametrów

Dlaczego taki parametr ustawiono i dlaczego taka wartość?

4.5. Weryfikacja działania

Zrzuty ekranu z zaplanowaną i wykonaną ścieżką (Typ wizualizacji: Odometry): http://wiki.ros.org/rviz/DisplayTypes/Odometry