Raport z projektu indywidualnej organizacji studiów

Wojciech Ptasiński Maciej Ziaja

7 września 2020

Spis treści

Rozdział 1

Wstęp

Przedmiotem projektu jest konstrukcja zestawu wybranych algorytmów robotyki dla mobilnych pojazdów autonomicznych oraz ich symulacja w środowisku ROS (Robotics Operating System). Platforma ROS pozwala na szybkie prototypowanie oprogramowania robotów w wirtualnych środowiskach symulacyjnych z wykorzystaniem systemu operacyjnego Linux. Modelowanie obiektów i konstrukcja algorytmów w językach programowania Python oraz C++ z użyciem systemu ROS daje możliwość łatwego przenoszenia oprogramowania na systemy wbudowane.

Wybrane algorytmy obejmują tematy predykcji położenia, przetwarzania danych wizyjnych, wykrywania elementów otoczenia oraz planowania toru jazdy. Wyniki projektu mogą być podstawą do konstrukcji platformy sprzętowej realizującej algorytmy. Gotowa platforma może stanowić podstawę do dalszych prac nad wykorzystaniem sztucznej inteligencji i algorytmów sterowania w systemach mobilnych.

Dodatkowo, w ramach projektu, zrealizowano analizę biznesową produkcji robotów autonomicznych w lokalnych warunkach, na podstawie danych o istniejących firmach działających w rozpatrywanej dziedzinie.

Środowisko ROS znajduje zastosowania w przemyśle oraz jest narzędziem wykorzystywanym przy pracach nad dynamicznie rozwijającą się technologią pojazdów autonomicznych.

Rozdział 2

Organizacja

2.1 Podział zadań

Podział zadań między studentów przedstawiono w tabeli 2.1. Ilość zadań przydzielonych zadań oraz ich charakter odpowiada efektom kształcenia realizowanym przez studentów.

Wojciech Ptasiński	Maciej Ziaja
Konfiguracja środkowiska ROS	Przygotowanie skryptów konfiguracyjnych
Przygotowanie modelu robota i symulatora	Implementacja sterowników czujników IMU
Opracowanie odometrii robota	Implementacja filtru Kalmana
Implementacja sterowania robotem	Opracowanie i implementacja systemu AHRS
Analiza biznesowa	Opracowanie prototypu systemu wizyjnego
	Przygotowanie prototypu sprzętowego mikroprocesora
	Analiza biznesowa

Tablica 2.1: Tabela podziału zadań między studentów realizujących projekt

Rozdział 3

Środowisko prototypowania robotów ROS

Robot Operating System (ROS) to framework do pisania oprogramowania robotów. Jest to zbiór narzędzi, bibliotek i konwencji, które mają na celu uproszczenie zadania tworzenia złożonych zachowań robotów na wielu różnych platformach robotyki.

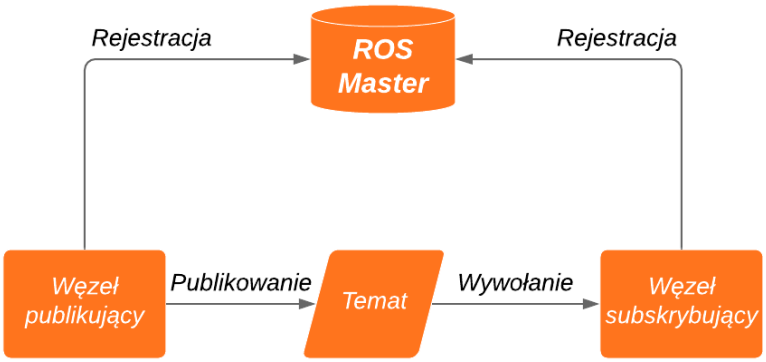
Architektura środowiska uruchomieniowego ROS to sieć procesów *peer-to-peer* (potencjalnie rozproszona na komputerach), które są luźno połączone za pomocą infrastruktury komunikacyjnej ROS.

Peer-to-peer (P2P) to model komunikacji, w którym każda ze stron ma te same uprawnienia i każda ze stron może zainicjować sesję komunikacyjną. ROS implementuje kilka różnych stylów komunikacji, w tym synchroniczną komunikację w stylu RPC za pośrednictwem usług, asynchroniczne przesyłanie danych do tematów i przechowywanie danych na serwerze parametrów.

RPC (ang. *remote procedure call*) polega na tym ,że gdy program komputerowy powoduje wykonanie procedury (podprogramu), w innej przestrzeni adresowej (zwykle na innym komputerze w sieci współdzielonej), która jest kodowana tak, jakby była normalnym (lokalnym) wywołaniem procedury, bez programisty jawnie kodującego szczegóły dotyczące zdalnej interakcji.

3.1 Komunikacja publish-subscribe

W architekturze oprogramowania na platformie ROS, komunikacja odbywa się zgodnie protokołem przesyłania wiadomości **publish-subscribe** (rys. 3.1), realizujące połączenie pomiędzy poszczególnymi **węzłami**. Protokół publish-subscribe, polega na tym, że węzły wysyłające wiadomości, nazywane **węzłami publikującymi**, nie określają do jakiego konkretnego odbiorcy, nazywanego **węzłem subskrybującym**, ma dana wiadomość trafić. W zamian, kategoryzują wysyłane wiadomości na klasy, które z kolei nazywamy **tematami**, bez żadnej informacji do jakiego, jeżeli do jakiegokolwiek węzła, ma ona trafić. Cała komunikacja odbywa się dzięki serwerowi (nazywanemu **ROS Master**), który umożliwia węzłowi subskrybującemu zidentyfikowanie źródła danych będących przedmiotem zainteresowania. Ten paradygmat daje możliwość istnienia wielu jednoczesnych węzłów publikujących i subskrybujących. Jeden węzeł publikujący może publikować dane do wielu węzłów subskrybujących lub zasubskrybować się u jednego z innych wydawców w bieżącym systemie.



Rysunek 3.1: Schemat przepływowy danych w protokole publish-subscribe, na platformie ROS

W ROS węzeł jest jednostką lub procesem, który wykonuje pewną formę obliczeń i/lub przetwarzania i akwizycji sygnałów. Urządzenia i systemy wykorzystywane przez pojazd, który jest omawiany w niniejszym projekcie, można przedstawić za pomocą węzłów. Na przykład, węzeł na dane enkodera, węzeł dla urządzenia LiDAR, węzeł do autonomicznej jazdy (w tym nawigacja, planowanie trasy, unikanie przeszkód), węzeł do wizualizacji. Węzły porozumiewają się ze sobą za pomocą struktur danych (określane jako typy wiadomości) które składają się z wymaganych pól będących typu liczba całkowita, liczba zmiennoprzecinkowa, wartość logiczna czy też obiekt pewnej klasy. Jeśli węzeł chce się komunikować z innym węzłem, opublikuje wiadomość o określonej strukturze do tematu, a inny węzeł zasubskrybuje ten temat (oczekując określonego typu wiadomości i pola) i gromadząc wymagane dane. Przy każdej dostarczonej wiadomości, w węźle subskrybującym aktywowane jest wywołanie (ang. *callback*), w którym zdefiniowane są, dalsze czynności powiązane z tą wiadomością np. przetworzenie w określony sposób i przekazanie do kolejnego węzła.

3.2 Model URDF

Unified Robot Description Format (URDF) to specyfikacja XML opisująca robota. Roboty formacie URDF są reprezentowane w strukturze drzewa, które przedstawia stworzoną hierarchię połączeń pomiędzy poszczególnymi członami modelu. Specyfikacja zakłada, że robot składa się ze sztywnych elementów połączonych ze sobą. Specyfikacja obejmuje kinematyczny i dynamiczny opis, wizualną reprezentację oraz model zderzeniowy robota.

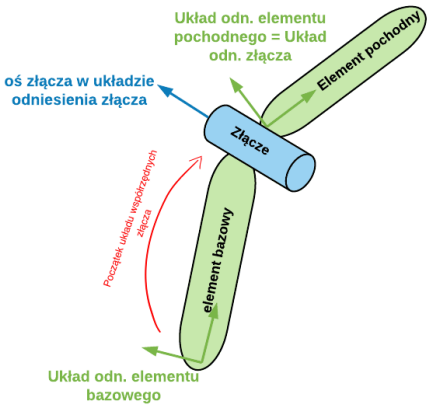
Na poziomie implementacji plik URDF jest po prostu plikiem tekstowym zawierającym znaczniki XML: określone słowa kluczowe, które ROS rozpoznaje jako część URDF.

Niektóre ze znaczników XML odnoszą się do innych plików lub modeli 3D części robotów, co pozwala korzystać z plików zewnętrznych w celu szybkiego włączenia szczegółowego kształtu wraz z kolorystyką.

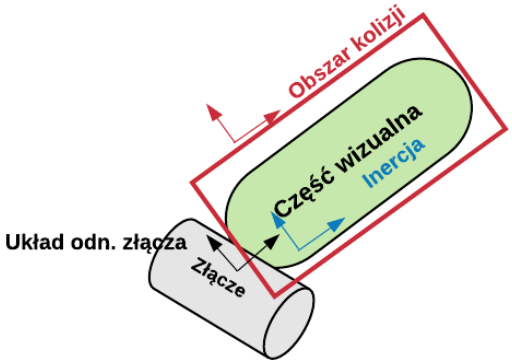
Opis robota składa się z zestawu członów(ang. *links*) i zestawu elementów łączących ze sobą człony(*ang. joints*).

Człon robota (*link*) opisuje sztywny korpus o bezwładności, cechach wizualnych i właściwościach kolizji.

Złącze(*joint*) opisuje kinematykę i dynamikę złącza, umiejscowienie połączenia, element bazowy oraz element pochodny, między którymi stworzone jest dane złącze.



(a) Złącze (*joint*)



(b) Człon robota (*link*)

Rysunek 3.2: Elementy struktury opisu robota w formacie URDF.

3.3 Środowisko symulacyjne

Symulacja robota pomaga obniżyć całkowity koszt integracji. Dzieje się tak głównie dzięki możliwości symulowania aplikacji w świecie rzeczywistym bez fizycznych kosztów związanych z tym faktem. Symulacja robota również zmniejsza, a często nawet eliminuje potrzebę wprowadzania zmian w systemie po jego zainstalowaniu, ponieważ już udowodniono, że jego procesy działają.

Spośród dostępnych otwarto-źródłowych symulatorów 3D, w celu zasymulowania pracy robota oraz pomiarów ze skanera LIDAR, wybrano symulator *Gazebo*. Głównym powodem tego wyboru jest kompatybilność z wykorzystywaną platformą ROS. Oprócz tego, Gazebo zapewnia wysokowydajne silniki fizyki, realistyczne renderowanie środowisk, w tym wysokiej jakości oświetlenie, cienie i tekstury.

Istotną cechą tego symulatora jest możliwość generowania danych czujników, opcjonalnie z szumem, ze skanerów laserowych, kamer 2D/3D, czujników w stylu Kinect, czujników kontakto-

Rysunek 3.3: Logo symulatora Gazebo 3D

wych, siły momentu obrotowego i innych.

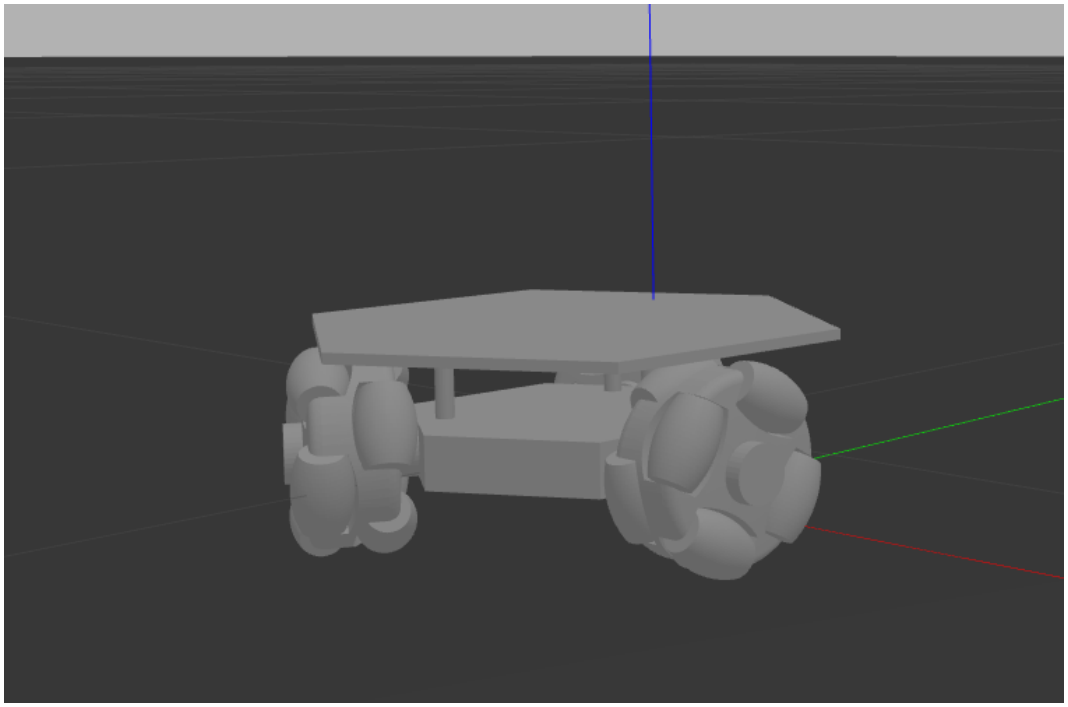
Obslugiwany przez symulator ROS format SDF opisuje obiekty i środowisko dla symulacji, wizualizacji i sterowania robotem. Pozwala dokładnie opisać wszystkie aspekty robota, a w razie potrzeby, w łatwy sposób można go przekonwertować na format URDF. Oprócz możliwości tworzenia własnych robotów Gazebo zapewnia kilka modeli, gotowych do przetestowania.

3.4 Symulacja omni robota

Na poniższym obrazku znajduje się wizualizacja omni robota, który został stworzony dla celów realizacji projektu. Początkowo poszczególne części zostały zaprojektowane jako siatka graficzna, wykorzystując do tego specjalne programy graficzne, a następnie wszystkie stworzone elementy robota zostały zintegrowane ze sobą w formacie URDF.

Wykorzystując wspomniany format (URDF), stworzono odpowiedni model, który opisuje również momenty bezwładności poszczególnych elementów, co pozwala na odwzorowanie tego, jak zachowałby się podobny robot w świecie rzeczywistym.

Na robocie znajduje się platforma, na której w dalszym etapie tworzenia modelu, został zamontowany skaner laserowy LiDAR.



Rysunek 3.4: Symulacja omni-robota

Rozdział 4

Filtr Kalmana i system AHRS

System AHRS (ang. *attitude and heading reference system*) pozwala na wyznaczenie orientacji obiektu w przestrzeni. Systemu tego typu znajdują szerokie zastosowanie w pojazdach autonomicznych.

System AHRS można podzielić na dwie główne części: system czujników oraz system predykcji orientacji w przestrzeni. Fragment systemu odpowiedzialny za działanie czujników stawia wyzwania związane z obsługą sprzętu oraz jego kalibracją. Algorytm predykcji pozwala na fuzję danych z zespołu czujników w celu określenia orientacji obiektu w przestrzeni.

Wynikiem działania systemu AHRS jest wektor trzech wartości reprezentujących obrót obiektu wokół osi trójwymiarowego układu współrzędnych. Kąty obrotu noszą nazwę *kątów Eulera*, ich ideowym odpowiednikiem w terminologii lotniczej są kąty *roll, pitch, yaw*.

4.1 Czujniki systemu AHRS i implementacja sprzętowa

W celu określenia orientacji obiektu w przestrzeni konieczny jest zestaw czujników. Do wyznaczenia pełnej orientacji w przestrzeni trójwymiarowej stosuje się zazwyczaj zestaw: akcelerometru, żyroskopu oraz magnetometru. Akcelerometr i żyroskop tworzą razem zestaw nazywany IMU (ang *inertial measurement unit*). System IMU pozwala na wyznaczenie kątów odchylenia obiektu od pionu (*roll* oraz *pitch*), co w wielu zastosowaniach jest wystarczające. Aby uzyskać pełne współrzędne orientacji obiektu w przestrzeni należy użyć także magnetometru, który pozwala określić kąt obrotu wokół osi pionowej (*yaw*).

W projekcie użyto układu *MinIMU-9 v5* firmy *Pololu* o dziewięciu stopniach swobody (trzy stopnie swobody na każdy z czujników). *MinIMU* zawiera zestaw: układ IMU *LSM6DS33* oraz magnetometr *LIS3MDL*. Komunikacja z czujnikami odbywa się poprzez magistralę *I2C*, przy czym każdy z trzech czujników ma osobny adres i wymaga oddzielnej obsługi komunikacji. Czujniki wchodzące w skład systemu AHRS wymagają kalibracji, co jest szczególnie ważne przypadku magnetometru.

4.1.1 Żyroskop

Żyroskop to przyrząd pozwalający określić prędkość obiektu kątową w trzech osiach. W spoczynku żyroskop powinien dawać zerowe odczyty ze wszystkich osi. Jest to podstawowy czujnik pozwalający określić orientację obiektu w przestrzeni. Wyznaczenie kątów obrotu wymaga jedynie całkowania odczytu z żyroskopu (ponieważ prędkość kątowa to pochodna kąta obrotu). Niestety sam żyroskop nie jest wystarczający aby precyzyjnie orientować obiekt w przestrzeni,

ze względu na zjawisko dryftu (ang. *gyroscope drift*). Żyroskop jest mało podatny na szumy pomiarowe o małej częstotliwości, natomiast jest wrażliwy na pojawianie się stałych, ale niewielkich błędów w odczytach. Ze względu na konieczność całkowania danych z żyroskopu te małe błędy są kumulowane, w wyniku czego wraz z upływem czasu, orientacja na podstawie działania żyroskopu staje się coraz mniej wiarygodna. Dlatego w celu precyzyjnego ustalania orientacji w przestrzeni konieczne jest użycie żyroskopu w parze z akcelerometrem oraz zastosowanie algorytmu fuzji danych.

4.1.2 Akcelerometr

Akcelerometr to czujnik pozwalający mierzyć przyspieszenie kątowe obiektu. W stanie spoczynku osie poziomie akcelerometru powinny wskazywać zerowe odczyty, natomiast oś pionowa powinna wskazywać wartość przyspieszenia ziemskiego.

Akcelerometr jest wolny od zjawiska dryftu, natomiast jest wrażliwy na szumy pomiarowe o dużych częstotliwościach. Przeciwna natura niedokładności akcelerometru oraz żyroskopu sprawia, że najlepiej pracują one w parze, z użyciem algorytmu fuzji danych.

W czujnikach IMU często dokonuje się kalibracji błędu zera (addytywnego). Należy wykonać to w trakcie spoczynku systemu na płaskiej powierzchni. Oś pionowa akcelerometru nie powinna być kalibrowana.

4.1.3 Magnetometr

Magnetometr mierzy indukcję pola magnetycznego w trzech kierunkach. Ponieważ między biegunami Ziemi przebiegają linie pola magnetycznego, to magnetometr może być użyty do wyznaczenia orientacji obiektu w przestrzeni (jako kompas). Oś magnetometru leżąca w kierunku północ-południe wskazuje wartości maksymalne indukcji, osie prostopadłe wskazują wartości minimalne.

Kluczowym zagadnieniem jest kalibracja magnetometru, który jest obciążony zarówno błędem addytywnym (nazywanym z ang. *hard iron*), jak i multiplikatywnym (nazywanym z ang. *soft iron*). Odczyty magnetometru są wrażliwe na zakłócenia powstałe w wyniku działań innych części układów robotyki. Z tego powodu kalibracja magnetometru jest koniecznością, magnetometr nieskalibrowany jest obciążony błędami uniemożliwiającymi poprawne działanie systemu AHRS.

4.1.4 Kalibracja czujników

W przypadku czujników IMU kalibracja polega na uśrednieniu odczytu n wartości w trakcie spoczynku na równym podłożu. Kolejne odczyty należy korygować o zapisaną wartość średniej.

W przypadku magnetometru w czasie kalibracji należy nim poruszać, tak aby każda oś jego pomiaru obróciła się w przestrzeni o kąt pełny. W przypadku poprawnie skalibrowanego magnetometru pomiary o maksymalnych wartościach powinny mieć na wykresie kształt okręgu ze środkiem w początku układu współrzędnych. W przypadku wystąpienia błędu *hard iron* środek okręgu jest przesunięty, natomiast w wyniku działania błędu *soft iron* następuje zniekształcenie okręgu.

Ważne jest aby kalibracji dokonać w odpowiedniej kolejności, najpierw należy wyznaczyć *hard iron*, a potem *soft iron*. Obliczenie błędu multiplikatywnego wymaga, aby błąd addytywny był już zniwelowany. Błąd *hard iron* jest średnią ze skrajnych wartości n pomiarów, co przedstawiono na wzorze 4.1, gdzie $\vec{m_n}$ to wektor pomiarów z czujnika.

$$\vec{e_{hard}} = \frac{max\,(\vec{m_n}) + min\,(\vec{m_n})}{2} \tag{4.1}$$

Błąd *soft iron* należy wyznaczyć według wzorów od 4.3 do 4.4.

$$\vec{r} = \frac{max\left(\vec{m_n}\right) - min\left(\vec{m_n}\right)}{2} \quad \vec{r} = [r_x, r_y, r_z] \tag{4.2}$$

$$r_a = \frac{r_x + r_y + r_z}{3} \tag{4.3}$$

$$\vec{e_{soft}} = \left[\frac{r_a}{r_x}, \frac{r_a}{r_y}, \frac{r_a}{r_z}\right] \tag{4.4}$$

Ostatecznie odczyty magnetometru należy korygować według wzoru

$$\vec{m_{fixed}} = (\vec{m} - \vec{e_{hard}}) \cdot \vec{e_{soft}} \tag{4.5}$$

4.1.5 Implementacja obsługi czujników w systemie operacyjnym Linux

Obsługę czujników zdecydowano się oprzeć na systemie Linux, który jest również bazą dla środowiska ROS. Jądro systemu Linux udostępnia dostęp do magistrali komunikacji *I2C* poprzez wywołania systemowe (ang. *system calls*). Urządzenia peryferyjne w systemach pochodnych systemu *UNIX* są reprezentowane przez pliki znakowe. W celu nawiązania połączenia z czujnikiem należy otworzyć plik urządzenia znakowego *I2C* za pomocą wywołania `open()`. Pliki urządzeń peryferyjnych znajdują się w katalogu `/dev/`. Połączenie nawiązuje się wywołaniem `ioctl(device_handle, I2C_SLAVE, i2c_address);`, gdzie `device_handle` to deskryptor otwartego pliku urządzenia znakowego, `I2C_SLAVE` to stała określająca tryb działania urządzania, a `i2c_address` to adres urządzenia podłączonego do magistrali *I2C*. Wymiana z urządzeniami odbywa się poprzez zapis i odczyt danych z pliku urządzenia, za pomocą wywołań `read()` oraz `write`. Istotne jest aby odczytywać i zapisywać odpowiednią liczbę bajtów, oraz poprawnie skalować odczyty. Informacje na temat formatu przesyłanych danych znajdują się w notach katalogowych czujników. Istnieją dodatkowe biblioteki ułatwiające pracę z urządzeniami *I2C* (np. *SMBUS*), jednak opisany sposób jest najbardziej uniwersalny. Utworzona biblioteka komunikacji z czujnikami *MinIMU-9 v5* będzie działała na każdym urządzeniu z systemem Linux, którego jądro obsługuje komunikację *I2C*.

Program napisano w języku C++ odpowiednio opakowując kod języka C. Błędy zgłaszane przez system *errno* obsłużono przez wyjątki. Adresy rejestrów i urządzeń *I2C* zapisano w silnie typowanych wyrażeniach wyliczeniowych, które parametryzują szablony klas urządzeń magistrali *I2C*. Oprogramowanie obsługi czujników udostępniono w postaci statycznie łączonej biblioteki programistycznej.

4.2 Filtr Kalmana

Filtr Kalma jest algorytmem estymacji stanu modelu dynamicznego. Algorytmy tego typu określają wewnętrzny stan obiektu na podstawie pomiarów wejścia i wyjścia tego układu. Filtr wykonuje estymacje rekurencyjnie, na podstawie wcześniejszych obliczeń, a szacunki są optymalne, zakładając że błędy pomiarowe mają rozkład normalny oraz układ jest liniowy. Jeżeli układ dyskretny opisany jest równaniami 4.6 i 4.7, gdzie:

$x\left(t\right)$ to wektor stanu w funkcji czasu,

$y\left(t\right)$ to wektor wyjścia układu (pomiaru) w funkcji czasu,

A to macierz systemowa (przejścia),

B to macierz wejścia,

H to macierz wyjścia (pomiaru),

$v(t), u(t)$ to wektory szumu, kolejno procesu oraz pomiaru.

$$x(t+1) = Ax(t) + Bu(t) + v(t) \tag{4.6}$$
$$y(t) = Hx(t) + w(t) \tag{4.7}$$

Działanie filtru Kalmana składa się z dwóch faz: fazy predykcji (*a priori*) oraz aktualizacji (*a posteriori*). Faza predykcji (ekstrapolacji) polega na wyznaczeniu stanu układu w nowej chwili czasu na podstawie modelu układu oraz wcześniejszego stanu. W czasie predykcji wyznaczana jest również nowa macierz niepewności (kowariancji estymacji). Predykcja odbywa się według wzorów 4.8 i 4.9, gdzie:

$\hat{x}_a(t)$ to estymacja *a priori* wektora stanu w funkcji czasu,

$\hat{x}_p(t)$ to estymacja *a posteriori* wektora stanu w funkcji czasu,

$P_a(t)$ to macierz kowariancji estymacji stanu *a priori* w funkcji czasu,

Q to macierz kowariancji szumu procesu.

$$\hat{x}_a(t) = A\hat{x}_p(t-1) + Bu(t-1) \tag{4.8}$$
$$P_a(t) = AP_p(t-1)A^T + Q \tag{4.9}$$

Kolejnym etapem jest aktualizacja, która odbywa się na podstawie pomiaru wyjścia układu. Ma ona za zadanie poprawić estymację stanu, uwzględniając różnicę między rzeczywistym pomiarem wyjścia, a jego wartością obliczoną *a priori* przy pomocy stanu z fazy predykcji. Kluczowym elementem filtru Kalmana jest *wzmocnienie Kalmana*, które reprezentuje balans ufności w predykcję *a priori* (według modelu) oraz w pomiary wyjścia *a posteriori*. W trakcie fazy aktualizacji, wyznaczane jest także nowe *wzmocnienie Kalmana*. Intuicyjnie można powiedzieć, że wzmocnienie jest wyznaczane tak, by najlepiej balansować pomiędzy przewidywaniem stanu według modelu matematycznego (niedoskonałego, a także nie obejmującego zakłóceń procesu) oraz przewidywaniem według pomiaru (obarczonego szumem pomiarowym). Faza aktualizacji odbywa się zgodnie ze wzorami od 4.10 do 4.12, gdzie:

$K(t)$ to *wzmocnienie Kalmana* w funkcji czasu,

$\hat{P}_p(t)$ to macierz kowariancji estymacji stanu *a posteriori* w funkcji czasu,

R to macierz kowariancji szumu pomiarowego.

$$K(t) = P(t)H^T \cdot \left(HP_a(t)H^T + R\right)^{-1} \tag{4.10}$$
$$\hat{x}_p(t) = \hat{x}_a(t) + K \cdot (y(t) - H\hat{x}_a(t)) \tag{4.11}$$
$$P_p(t) = (I - K(t)H) \cdot P_a(t) \tag{4.12}$$

4.2.1 Zastosowanie filtru Kalmana w fuzji danych pochodzących z czujnika IMU

Filtr Kalmana można zastosować, aby oszacować orientację obiektu w przestrzeni na podstawie odczytów z czujnika IMU. Fuzja danych przy pomocy filtru pozwala zniwelować niedoskonałości żyroskopu (dryft) oraz akcelerometru (szumy pomiarowe). Zazwyczaj dane z żyroskopu są używane w roli wejścia systemu, na etapie predykcji, a pomiary z akcelerometru są wykorzystywane podczas fazy aktualizacji. Czujniki IMU pozwalają jedynie na określenie kątów *roll* oraz *pitch*, w dalszej części raportu zostanie opisany sposób wyznaczenia kąta *yaw* na podstawie odczytów magnetometru.

4.2.2 Dane wejściowe filtru Kalmana

Na podstawie danych z akcelerometru można obliczyć dwa z kątów opisujących orientację obiektu w przestrzeni (wykorzystując fakt, stałego przyspieszenia ziemskiego w pionowej osi inercyjnego układu odniesienia). Kąty *roll* i *pitch* są wyznaczane według wzorów 4.13 oraz 4.14, gdzie gdzie: ϕ i θ to kolejno kąty *roll oraz pitch*, a a to odczyty akcelerometru.

$$\tilde{\phi}_a = arctan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right) \cdot \frac{180}{\pi} \tag{4.13}$$

$$\tilde{\theta}_a = arctan\left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}\right) \cdot \frac{180}{\pi} \tag{4.14}$$

Prędkości kątowe odczytane z żyroskopu nie mogą być użyte wprost, ze względu na powiązanie osi odczytu w układzie odniesienia. Obrót układu wokół jednej z jego osi może powodować zmianę orientacji obiektu w wielu osiach zewnętrznego układu odniesienia. Transformacji prędkości kątowych z układu żyroskopu do układu inercyjnego należy dokonać według wzoru 4.15, gdzie: ϕ, θ, ψ to kolejno kąty *roll, pitch oraz yaw*, a g to odczyty żyroskopu.

$$\begin{bmatrix} \dot{\phi}_g \\ \dot{\theta}_g \\ \dot{\psi}_g \end{bmatrix} = \begin{bmatrix} 1 & sin\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi sec\theta & cos\phi sec\theta \end{bmatrix} \cdot \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} \tag{4.15}$$

4.2.3 Równania filtru Kalmana dla fuzji danych

Na podstawie opisu teoretycznego filtru Kalmana oraz przygotowanych danych z czujników można przygotować ostateczną wersję systemu podstawiając konkretne macierze systemowe i wektory z równań: 4.16 i 4.17 do równań od 4.6 do 4.12. Symbol Δt oznacza okres próbkowania czujników.

$$A = \begin{bmatrix} 1 & -\Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} \Delta t & 0 \\ 0 & 0 \\ 0 & \Delta t \\ 0 & 0 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.16}$$

$$\vec{x}(t) = \begin{bmatrix} \hat{\phi}(t) \\ b_{\hat{\phi}(t)} \\ \hat{\theta}(t) \\ b_{\hat{\theta}(t)} \end{bmatrix} \quad \vec{u}(t) = \begin{bmatrix} \dot{\phi}_g(t) \\ \dot{\theta}_g(t) \end{bmatrix} \quad \vec{y_t} \begin{bmatrix} \tilde{\phi}_a(t) \\ \tilde{\theta}_a(t) \end{bmatrix} \tag{4.17}$$

Macierze Q (kowariancja szumu procesu) oraz R (kowariancja szumu pomiaru) są dobierane przez użytkownika. Są to zazwyczaj macierze diagonalne, im większa wartość na ich przekątnej, tym większe zakłócenia w danej części układu. Początkowa wartość na przekątnej P reprezentuje wstępną ufność w dokładność modelu.

4.2.4 Algorytm kompasu oraz kompensacja przechylenia

Zgodnie z wcześniejszym opisem magnetometru można użyć jako kompasu w celu wyznaczenia kąta *yaw*. Jednak w przypadku przechylenia układu czujników w pozostałych kątach płaszczyzna kompasu (magnetometru) zostanie odchylona od płaszczyzny podłoża, co spowoduje zakłamanie odczytów. Aby zniwelować ten problem można użyć wyliczonych uprzednio kątów *roll* oraz *pitch*. Służy do tego formuła *kompensacji przechylenia* (ang. *tilt compensation*, którą przedstawia wzory od 4.18 do 4.20.

$$h_x = m_x cos\hat{\theta} + m_y sin\hat{\theta} sin\hat{\phi} + m_z sin\hat{\theta} cos\hat{\phi} \tag{4.18}$$

$$h_y = m_y cos\phi - m_z sin\phi \tag{4.19}$$

$$\psi = atan2(-h_y, h_x) \tag{4.20}$$

4.3 Implementacja programistyczna systemu AHRS

System AHRS zaimplementowano w języku C++ w postaci biblioteki programistycznej. Aby umożliwić jak najszersze zastosowanie opracowanego kodu zadbano o jego wysoką przenaszalność. Aby, umożliwić działanie programu na urządzaniach wbudowanych *bare metal* w programie unikano polegania na wyjątkach oraz dynamicznej alokacji pamięci. Większość klas i funkcji jest parametryzowana przez szablony w czasie kompilacji, co pozwala na generowanie szybkiego kodu (kosztem hipotetycznego zwiększenia rozmiaru plików wykonywalnych).

Rozdział 5

Odometria robota

Do wyznaczenia odometrii pojazdu wymagane są enkodery, dlatego w celu stworzenia projektu, który będzie łatwy do integracji z rzeczywistym pojazdem, zasymulowano pomiary z enkodera. Pomiary z enkodera pobierane są w ten sposób, że porównywane są w czasie kolejne dwa odczyty z kąta obrotu osi koła. Na tej podstawie otrzymujemy prędkość kątową danego koła.

Część symulująca odczyt z autoenkodera została zaimplementowana jako osobny węzeł w architekturze platformy ROS. Węzeł autoenkodera komunikuje się z węzłem który jest odpowiedzialny za wyznaczanie poszczególnych parametrów, określających odometrię pojazdu. Parametry, czyli model kinematyczny robota, został opisany w kolejnym podrozdziale.

5.1 Model kinematyczny robota

Równania kinematyczne odnoszące się do punktu odniesienia robota:

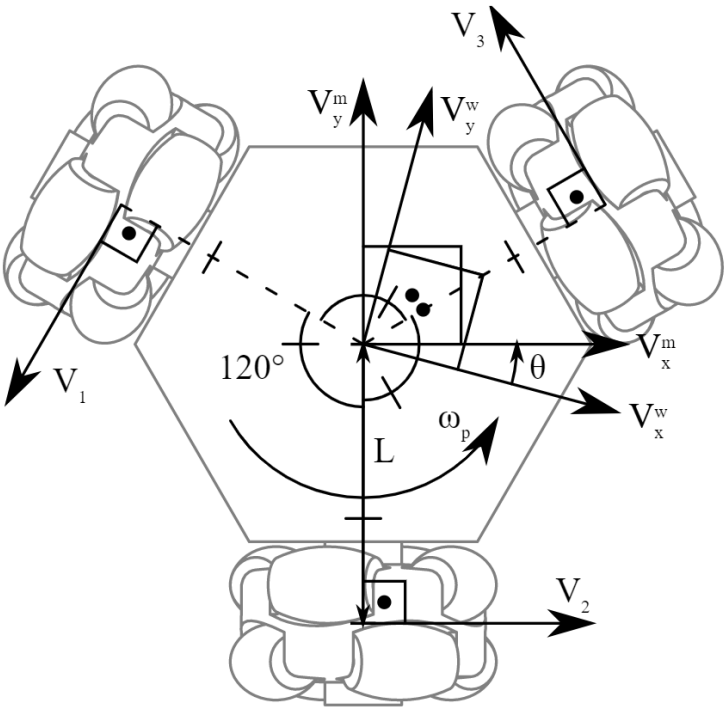
$$V_x^m = \frac{2 \cdot V_2 - V_1 - V_3}{3}$$
$$V_y^m = \frac{\sqrt{3} \cdot V_3 - \sqrt{3} \cdot V_1}{3}$$
$$\omega_p = \frac{V_1 + V_2 + V_3}{3 \cdot L}$$

Równania kinematyczne odnoszące się do punktu odniesienia świata:

$$V_x^w = \cos\theta \cdot V_x^m - \sin\theta \cdot V_y^m$$
$$V_y^w = \sin\theta \cdot V_x^m + \cos\theta \cdot V_y^m$$

Kinematyka odwrotna w układzie odniesienia robota:

$$V_x^m = \cos\theta \cdot V_x^m + \sin\theta \cdot V_y^m$$
$$V_y^m = -\sin\theta \cdot V_x^m + \cos\theta \cdot V_y^m$$



Rysunek 5.1: Schemat ukazujący poszczególne wektory prędkości

Kinematyka odwrotna w układzie odniesienia świata:

$$V_1 = -\frac{V_x^m}{2} - \frac{\sqrt{3} \cdot V_y^m}{2} + L \cdot \omega_p$$
$$V_2 = V_x^m + L \cdot \omega_p$$
$$V_3 = -\frac{V_x^m}{2} + \frac{\sqrt{3} \cdot V_y^m}{2} + L \cdot \omega_p$$

Za pomocą powyższych wzorów modelu kinematycznego pojazdu, zrealizowano odometrię oraz sterowaniem robotem.

Rozdział 6

System wizyjny

6.1 Sprzęt wizyjny

Jako bazę do wyznaczania ścieżki robota w terenie zdecydowano się użyć kamery *Raspberry Pi Camera V2*, która oferuje obraz wideo w rozdzielczości *full HD* oraz zdjęcia w rozdzielczości do *4K*. Kamera jest wyposażona we wbudowany obiektyw szerokokątny, co jest korzystne w zastosowaniach robotyki.

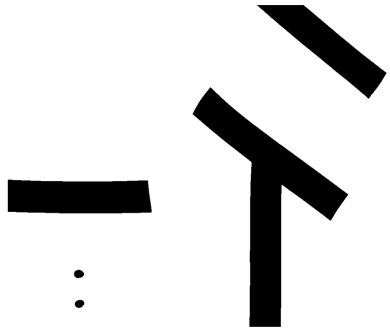
6.2 Wyznaczanie przestrzeni ruchów robota

Kamerę przygotowano do wykorzystania w celu wykonywania zdjęcia labiryntu, w którym poruszać może się robot. Obraz z kamery należy przetworzyć aby uzyskać mapę przestrzeni, po której może poruszać się robot. Biała przestrzeń na mapie powinna oznaczać miejsca, w których może znaleźć się środek robota, czarne pola to miejsca zabronione (przeszkody oraz ich margines). Budowę mapy wykonano w następujących krokach:

- binaryzacja zdjęcia z kamery,
- wykrycie położenia robota oraz jego promienia na podstawie markerów,
- usunięcie obrysu robota z przestrzeni zabronionej,
- naniesienie marginesu przeszkód.

Ze względu na zdalny charakter pracy w semestrze letnim 2020 przygotowano prototypowy model labiryntu. Podstawą przetwarzania obrazu labiryntu jest binaryzacja metodą *isodata*. Oparcie wizji na algorytmie binaryzacji wymaga odpowiedniego kontrastu zdjęcia oraz dobrego oświetlenia labiryntu. *Isodata* jest adaptacyjną metodą binaryzacji zdjęć, która gwarantuje, że wartość progowania jest średnią ze średnich wartości ekspozycji podzielonych fragmentów zdjęcia. Taki algorytm binaryzacji odpowiada algorytmowi klasteryzacji *k-średnich*, gdzie k wynosi dwa. Przykładowy labirynt po binaryzacji przedstawiono na rysunku 6.1.

W następnym kroku algorytm wizyjny przeprowadza wykrycie pozycji i rozmiaru robota na podstawie dwóch czerwonych markerów umieszczonych na jego obwodzie. Punkty o charakterystycznym kolorze najlepiej wykrywać w przestrzeni barw HSV (barwa, nasycenie, jasność ang. *hue, saturation, value*). W omawianej przestrzeni kolory są reprezentowane przez wartość kąta; od zera do kąta pełnego, przy czym wartości reprezentuje często się jako ułamek dziesiętny z maksymalnej wartości. Kolor czerwony jest w takim systemie opisywany przez kąty zbliżone do zera.



Rysunek 6.1: Przykładowy labirynt poddany binaryzacji

(a) Wykryte markery i ich centroidy

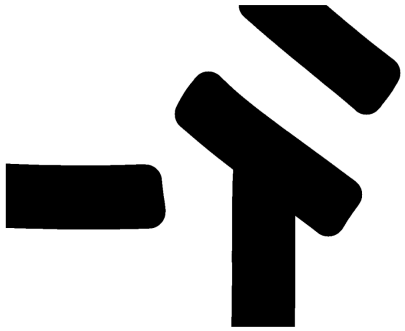
(b) Labirynt po usunięciu markerów

Rysunek 6.2: Wykrywanie oraz usuwanie markerów z mapy labiryntu

Markery zdecydowano się kwalifikować w przedziale odcieni czerwonego od 0,9 do 1 oraz 0 do 0,1. Wartości nasycenia oraz jasności markerów są akceptowane dla wartości większych od 0,3. Po progowaniu pozycji markerów dokończono proces ich segmentacji przez etykietowanie oraz obliczenie parametrów obszaru zajmowanego przez markery. Wyznaczono ich centroidy; punkt między centroidami markerów to środek robota. Po wyznaczeniu pozycji robota można usunąć jego kształt z mapy przez odjęcie maski jego obrysu z obrazu. Wyniki omówionych operacji przedstawiono na rysunkach 6.2a oraz 6.2b.

Ostatni etap przygotowania mapy polega na dodaniu do ścian labiryntu marginesu, który zapewni, że krawędź robota nie zetknie się z przeszkodami. W tym celu dokonano procesu erozji binarnej przestrzeni dozwolonej ruchu robota. Margines ścian wyznaczono na podstawie średnicy robota wyznaczonej przy segmentacji markerów. Erozji dokonano używając jądra w kształcie okręgu o promieniu równym ponad połowie promienia robota. Ostatecznie powstaje mapa binarna, na której kolor biały oznacza przestrzeń w której może znaleźć się środek robota, przedstawiono ją na rysunku 6.3

Do wykonania wymienionych operacji użyto funkcji z biblioteki *scikit-image* dla języka Python.



Rysunek 6.3: Przykładowy labirynt z naniesionym marginesem ścian

Rozdział 7

Analiza biznesowa

W ramach projektu dokonano analizy możliwości biznesowych związanych z robotyką mobilną w regionie Górnego Śląska. Pełną treść analizy zamieszczono w osobnym dokumencie.

7.1 Przegląd dostępnych źródeł finansowania

Przeanalizowano ofertę dofinansowania projektów oraz przedsięwzięć biznesowych: zarówno krajowych, jak i zagranicznych. W ramach przeglądu wzięto pod uwagę oferty aktualne oraz niedawno zakończone. Dofinansowania w ramach programów rozwoju oraz inkubatorów charakteryzowały się atrakcyjnymi ofertami, niektóre z nich nie wymagały wkładu własnego, inne uzależniały go od wysokości kwoty pomocy. Wkład własny większości rozpatrywanych programów nie przekraczał parunastu procent. Oferty proponowały wsparcie w wysokości od parudziesięciu do paruset tysięcy złotych. Częstym wymogiem uczestnictwa w programie była gotowości prototypu produktu. Wsparcie w projektach jest ograniczone do małych firm, przy czym pożądana jest współpraca z uczelnią. W ramach przeglądu analizowano programy takie jak: *POWER*, *Program Akceleracyjny KPT* oraz *Program Inteligentny Rozwój*.

7.2 Przykłady rozwijających się firm na rynku robotyki

Spośród polskich działalności przeanalizowano rozwój firm takich jak *KP Labs* oraz *Future Processing*. Zwrócono uwagę na źródła finansowania projektów. Produkty o wysokiej innowacyjności i aspekcie naukowym (takie jak w przypadku firmy *KP Labs*) uzyskały bardzo wysokie dofinansowania unijne, przekraczające połowę wartości poszczególnych przedsięwzięć. Produkty związane na przykład z kosmonautyką mogły liczyć na dotacje wielkości milionów złotych.

Zdecydowano się także na analizę zagranicznych firm, które w ostatnich latach odniosły sukces na arenie międzynarodowej. Zwrócono uwagę na działalność skandynawskiej firmy *Antmicro*, która posiada polskie filie. Wiele z wykorzystywanych przez firmę technologii należy do grupy *open source* oraz *open source hardware*. Ciekawym przypadkiem okazała się inicjatywa *CommaAI*, której twórcą jest znany programista Georg Hotz. Jego firma skupia się na wyposażaniu istniejących samochodów, w systemy jazdy autonomicznej. *CommaAI* oferuje oprogramowanie oraz moduły współpracujące z czujnikami i kamerami wbudowanymi w samochody oraz smartfony. Jest to ciekawe podejście wykorzystujące wykosi poziom cyfryzacji otaczających technologii. Takie podejście ułatwia tworzenie produktów oraz pozwala na oferowanie ich w konkurencyjnych cenach.

7.3 Analiza SWOT potencjalnego przedsięwzięcia

W ramach analizy przeprowadzono analizę *SWOT* hipotetycznego przedsięwzięcia w dziedzinie robotyki mobilnej w warunkach biznesowych Górnego Śląska. Analiza skupia się na możliwościach założenia działalności biznesowej przez absolwentów Politechniki Śląskiej.

Mocne strony

- potencjalni członkowie zespołu, tzn. absolwenci kierunków technicznych są zapoznani z bieżącymi, nowoczesnymi technologiami,
- potencjalni członkowie zespołu rozumieją potrzeby nowoczesnego przemysłu.
- dobry wewnętrzny kontakt absolwentów z uczelnią.

Słabe strony

- potencjalni członkowie zespołu (absolwenci, doktoranci) mają małe doświadczenie biznesowe i menadżerskie.

Szanse

- zainteresowanie medialne i społeczne nowymi technologiami,
- liczne projekty dofinansowania; unijne oraz krajowe,
- możliwość współpracy z jednostkami naukowymi i uczelniami badawczymi.

Zagrożenia

- nieprzygotowanie polskiego rynku na adaptowanie nowych technologii,
- prawdopodobny kryzys,
- ryzyko przejęcia firmy przez zagraniczny koncern, ze względu na brak bezpośredniego odbiorcy zaawansowanych produktów na rodzimym rynku.

Rozdział 8

Podsumowanie

W ramach projektu zaprojektowano model 3D robota mobilnego, a następnie stworzono symulację, która pozwala na implementację oraz testowanie poszczególnych algorytmów.

Wykorzystując platformę ROS stworzono architekturę węzłów, która pozwoli na łatwą integrację z rzeczywistym obiektem, co może być podstawą do kontynuowania projektu.

Za pomocą wyznaczonego modelu kinematycznego pojazdu stworzono węzeł, który określa odometrie pojazdu, na podstawie symulowanych danych z enkodera.

Opracowano system AHRS bazujący na filtrze Kalmana. System wyznaczania orientacji w przestrzeni zaimplementowano i przetestowano z użyciem systemu mikroprocesorowego.

Wykonano prototyp systemu wizyjnego wyznaczającego przestrzeń dozwoloną robota w labiryncie.

W raporcie załączono wnioski wynikające z przeprowadzonej analizy rynku pojazdów autonomicznych oraz analizę ryzyka przedsięwzięcia tego typu.

Spis rysunków