POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PRACA MAGISTERSKA
na kierunku INFORMATYKA

Jakub Szymczak
Nr imm. 188178

Rok. akad. 2006/2007
Warszawa, 10.10.2006

SYSTEM WSPOMAGANIA WYZNACZANIA POŁOŻENIA ROBOTA MOBILNEGO ZA POMOCĄ OPTYCZNYCH CZUJNIKÓW ODOMETRYCZNYCH

Zakres pracy:

1. *Wstęp oraz cel pracy*
2. *Metody pomiaru odległości*
3. *Projekt czujnika optycznego*
4. *Analiza problemu*
5. *Projekt techniczny*
6. *Wykonanie testów i analiza wyników*
7. *Wnioski i podsumowanie*

Podpis i pieczątka

Kierownika Zakładu Dydaktycznego

Opiekun naukowy:
dr inż. Witold Czajewski

Termin wykonania: wrzesień 2007
Praca wykonana i zaliczona pozostaje własnością Instytutu i nie będzie zwrócona wykonawcy

Spis treści:

1. Wstęp oraz cel pracy

W dzisiejszym świecie roboty są obecne w każdej dziedzinie działalności człowieka. Wykorzystuje się je do najróżniejszych prac: powtarzalnych (rutynowych) i wymagających podejmowania decyzji w zależności od sytuacji. Twórcy SF prześcigają się w wymyślaniu nowych zastosowań dla maszyn, a inżynierowie próbują sprostać tym inwencjom niejednokrotnie z sukcesem.

Roboty możemy podzielić na różne grupy w zależności od przyjętego kryterium. Kryterium przemieszczania robotów dzieli roboty na stacjonarne i mobilne. Leonard i Durrant-Whyte w swojej pracy „Mobile Robot Localization by Tracking Geometric Beackons" określili problemy robotów w trzech pytaniach: „Where am I?" (gdzie się znajduje?), „Where am I going?" (dokąd zmierzam?), "How should I get there?" (jak się tam dostać?) [11].

W mojej pracy chciałbym się skupić na pytaniu „Where am I?". Mimo rozwoju technicznego problem samopozycjonowania robota w środowisku nie został definitywnie rozwiązany. Okazuje się, że barierą nie jest niedokładność czy szybkość układów pomiarowych tylko zjawiska fizyczne, których nie da się wyeliminować i przewidzieć w danym środowisku. W mojej pracy chciałbym zaproponować poprawę pomiaru odległości metodą nawigacji zliczeniowej (ang. dead reckoning). Była ona już wykorzystywana od wielu wieków jako alternatywna dla nawigacji astronomicznej opierającej się na gwiazdach. Polega ona na wyznaczaniu pozycji na podstawie punktu bazowego i trasy jaką pokonał robot. W robocie uczelnianym pomiar ten jest dokonywany za pomocą enkoderów umieszczonych na osi kół napędowych. W związku z tym, niedokładność pomiarów spowodowana jest głównie zjawiskiem poślizgu. W pracy chciałbym znaleźć rozwiązanie za pomocą dostępnych metod pomiarowych. W szczególności chciałbym wykorzystać urządzenia wskazujące używane w komputerach do obsługi kursora ze względu na ich możliwości wskazania translacji w dwóch prostopadłych wymiarach, dokładność i dostępność na rynku. Należy zatem stworzyć mechanizm sczytywania tych informacji i przekazania do algorytmów decyzyjnych robota. W mojej pracy ograniczę się do stworzenia aplikacji w systemie WindowsXP oraz badaniu wpływu rozmieszczenia czujników względem siebie na dokładność pomiaru translacji.

2. Układ pracy

Praca została podzielona na rozdziały i podrozdziały (w zależność od złożoności opisywanego fragmentu pracy). Zawierają one sformułowanie rozpatrywanego problemu, przegląd dostępnych i możliwych rozwiązań jak również opracowany pełny system wyznaczania położenia globalnego robota, który może być użyty jako system wspomagający mechanizm nawigacji dowolnego robota mobilnego.

W rozdziale 3 zawarłem przegląd metod wyznaczania pozycji globalnej robotów mobilnych używanych obecnie w wielu aplikacjach. Stanowią one zarówno podstawowe jak i dodatkowe metody nawigacji dla robotów i w dużej mierze wywodzą się z metod wypracowanych przez człowieka do określenia swojego położenia w środowisku. Bazują one na różnych zjawiskach fizycznych i wymagają odpowiednich czujników. Metody te zostały podzielone ze względu na wyznaczaną pozycje globalną na metody względne i bezwzględne.

Problem, który podejmuje niniejsza praca został sformułowany w rozdziale 4. Zawarłem w nim opis robota wydziałowego w aspekcie systemu nawigacji, w jaki został wyposażony oraz zjawisko poślizgu, które jest główną przyczyną powstawania błędu samopozycjonowania się robota. Rozdział ten zawiera również opis metod badawczych pomiaru i kalibracji robotów mobilnych zaproponowanych przez Borenstein w pracy [3].

Rozdział 5 zawiera opis użytych elementów i technologii przy projektowaniu czujnika odometrycznego oraz opis stworzonej na potrzeby pracy z czujnikami aplikacji i użytych w niej algorytmach. Został on podzielony na trzy podrozdziały ze względu na logiczny podział projektu na część sprzętową, część programową i część obliczeń matematycznych. Część sprzętowa zawiera informacje potrzebne do zbudowania czujnika odometrycznego na bazie myszy optycznej wraz z opisem problemów, z jakim miałem styczność w pracy oraz rozwiązaniem tych problemów. Znajdują się tu również informacje, w jaki sposób można komunikować się z czujnikiem. Wyznaczenie pozycji za pomocą czterech czujników wymaga odpowiedniej interpretacji wskazań tych czujników. W tym celu zostały zaadoptowane dwa algorytmy wyznaczania środka przemieszczenia układu sztywnego. Opis tych algorytmów znajduje się w podrozdziale 5.2. Opis stworzonej aplikacji został umieszczony w kolejnym podrozdziale. Zawiera on opis logiki biznesowej aplikacji, jak również najważniejsze klasy użyte w aplikacji.

Skuteczność czujników odometrycznych jak i optymalny rozkład względem robota zostało określone na podstawie badań. Ze względu na powtarzalność testów opracowałem odpowiednie scenariusze testowe. Opis ich został zawarty w rozdziale 6. w rozdziale tym zawarłem również informacje na temat układu optycznego, który zwiększyłby elastyczność systemu. Wyniki testów zostały zebrane w formie tabelarycznej i opisane w rozdziale 7.

Rozdział 8 zawiera podsumowanie pracy oraz zebrane wnioski na podstawie przeprowadzonych badań.

3. Metody pomiaru odległości

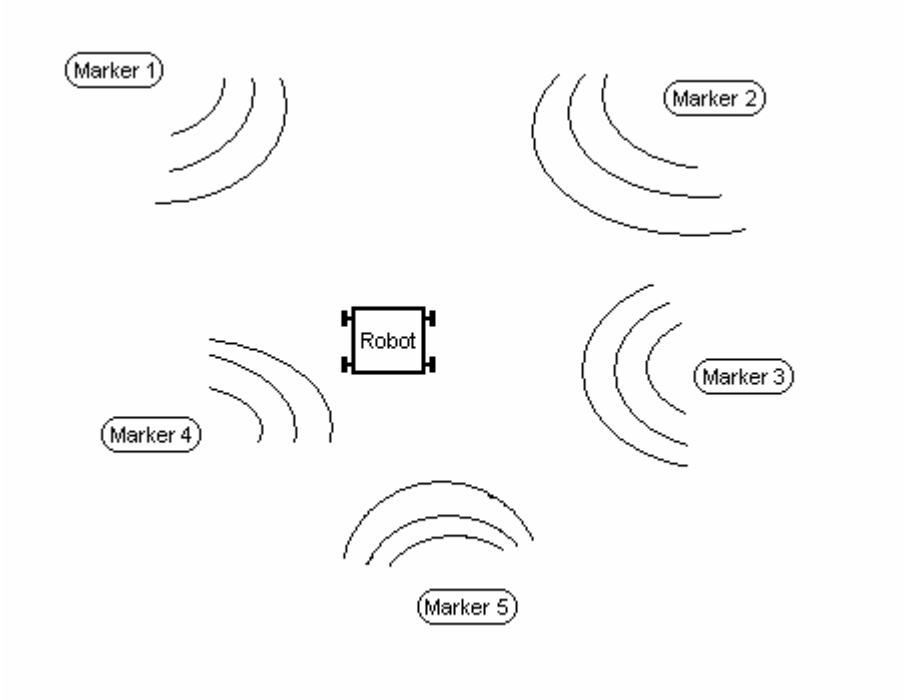
Roboty mobilne są wykorzystywane do różnych prac, a cechą, która je łączy jest zdolność do przemieszczania się. Mechanizmy poruszania są różne, od kół przez gąsienice do mechanizmów odnóży naśladujących przyrodę. W większości zastosowań od tych robotów nie wymaga się zdolności adaptacyjnych, samodzielności w podejmowaniu decyzji, co mają wykonać, a jedynie osiągnięcie zadanej precyzji, co przekłada się na możliwość wykorzystania „mapy" danego środowiska. Mapa ta jest zbiorem współrzędnych i związanych z nimi informacjami o danym punkcie w tym środowisku. Możliwe jest również dołączenie do wybranych punktów listy rozkazów dla robota, które musi wykonać po osiągnięciu zadanej pozycji. Sprowadza się to do „umiejętności" samopozycjonowania w środowisku; służą do tego mechanizmy wypracowane przez liczne doświadczenia i badania na robotach mobilnych wyposażonych w różnorodne czujniki.

Dla środowiska ściśle określonego możliwe jest zastosowanie metod wyznaczania położenia za pomocą znaczników (ang. mark [1]). Problem polega na doborze odpowiedniej liczby znaczników i rozłożeniu ich tak, aby pozycja robota mogła być wyznaczona jednoznacznie. Metody te zaliczane są do grupy pomiarów bezwzględnej pozycji (ang. absolute position measurements)[3], w skład których wchodzi:

- „aktywne przywoływanie" (ang. active beacons)
- rozpoznawanie markerów sztucznych i naturalnych (ang. artificial and natural landmark recognition)
- dopasowanie modelu (ang. model matching)

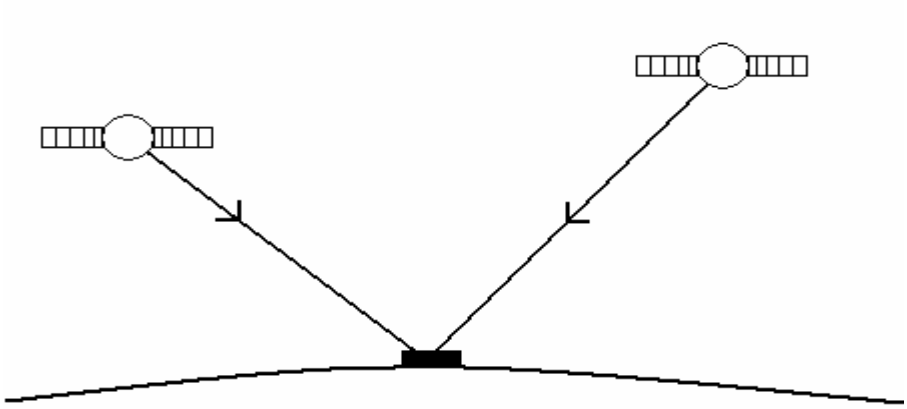
W metodzie "aktywne przywoływanie" (rys. 1) markerem jest nadajnik sygnału, w którym jest zaszyta informacja o położeniu tego markera. Na tej podstawie robot wyznacza swoje położenie. Wadą tej metody jest duży koszt eksploatacji nadajników oraz, co jest ważne dla robotów mobilnych, ograniczona przestrzeń do działania. Również dokładność wyznaczania takiej pozycji jest uzależniona od dokładności użytego sprzętu, co przekłada się na koszty takiego rozwiązania.

[1] funkcjonuje również w literaturze polskojęzycznej i jest używane wymiennie



Rys. 1. Aktywne przywoływanie

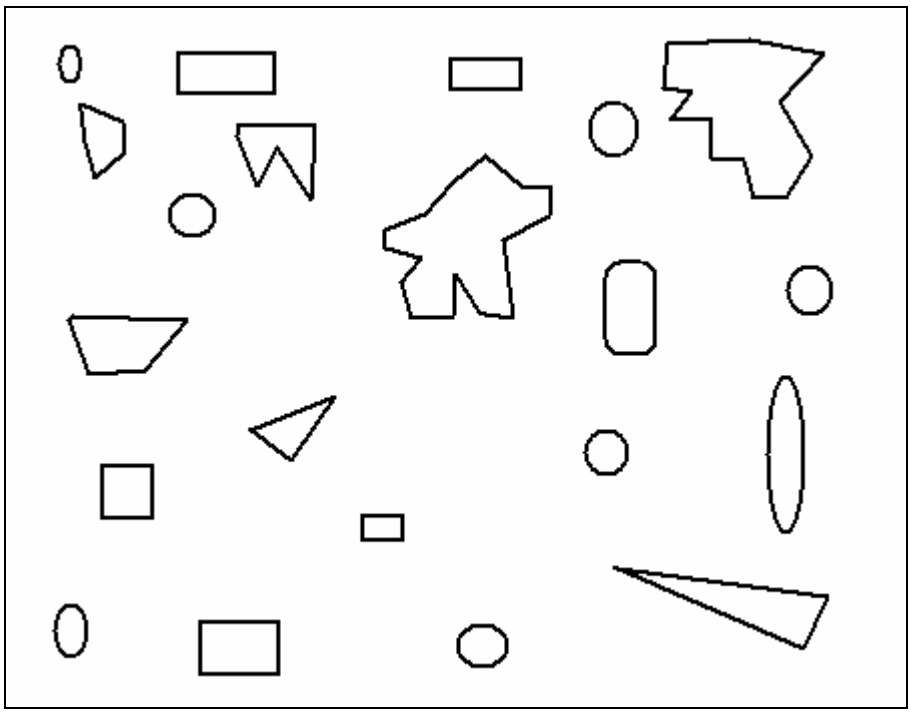
Metoda ta nie jest nowa i była już wykorzystywana od wieków przez żeglarzy, którzy obserwując gwiazdy wyznaczali swoje położenie. System latarni morskich jest też dobrym przykładem wykorzystywania tej metody w praktyce. Obecnie latarnie, obok sygnałów świetlnych, wysyłają sygnał radiowy, który zawiera informacje o identyfikatorze i położeniu danej latarni. Podobny system jest wykorzystywany w lotnictwie cywilnym i sportowym, w oparciu o maszty radiowe rozmieszczone w wybranych punktach nadające informacje o swoim położeniu. Do tej metody można też zaliczyć system nawigacji satelitarnej GPS (rys. 2), gdzie pozycja jest określana na podstawie sygnału z satelitów krążących wokół Ziemi.



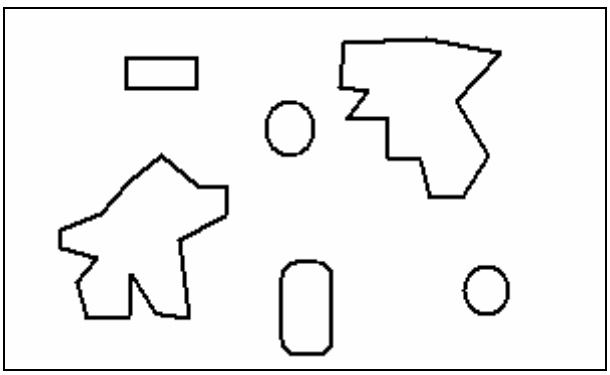
Rys. 2. Nawigacja satelitarna

Rozpoznawanie markerów polega na identyfikacji w systemie robota danego wzorca na podstawie informacji z czujników, i na tej podstawie określenie własnej pozycji. Markerami mogą być zarówno sztuczne jak i naturalne obiekty, które znajdują się w danym środowisku. Wadą takiego rozwiązania jest ograniczona liczba takich markerów, ponieważ muszą być one unikalne, aby nie było niejednoznaczności w ich wskazaniu. Poza tym należy wprowadzić do systemu robota odpowiednie wzorce. Metoda ta jest też wykorzystywana w grach terenowych.

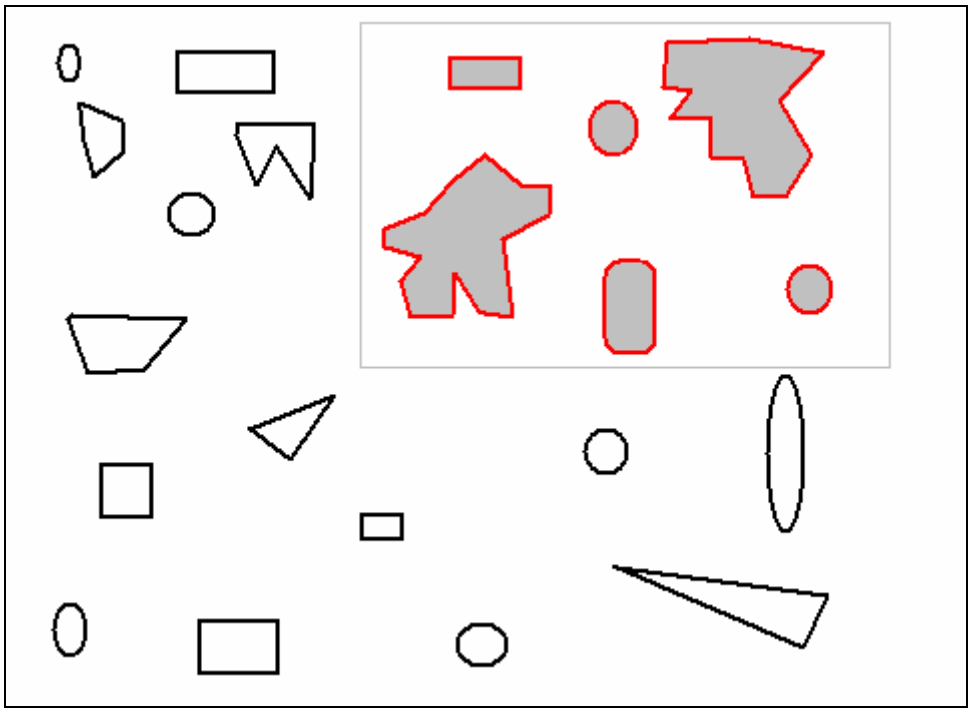
Do grupy pomiarów bezwzględnych autorzy opracowania „Where I am?" [3] zaliczyli również „dopasowanie modelu" (ang. model matching) (rys. 5). Jest to metoda polegająca na dopasowaniu informacji z czujników robota do odpowiedniego położenia na mapie. Robot tworzy tzw. mapę lokalną (rys. 4), którą próbuje dopasować do mapy globalnej (rys. 3), a wynik takiego dopasowania wyznacza położenie. Metoda ta może korzystać z pośrednich wyników powyższych metod, traktując je jako informacje z czujników robota.



Rys. 3. Mapa globalna



Rys. 4. Mapa lokalna



Rys. 5. Dopasowanie modelu

Wadą metod pomiaru bezwzględnego jest ograniczenie działania robotów mobilnych do zadanego środowiska, ponieważ wymaga znajomości położenia markerów (nadajników i obiektów) lub mapy danego środowiska. System GPS jest wolny od tych wad ze względu na swój globalny charakter i był wspomniany jako przykład wykorzystania metody „przywoływania". Metody te wykorzystuje się do rutynowych działań, gdzie robot pracuje w ściśle ustalonym środowisku lub nowe środowisko jest dobrze znane i wymaga jedynie wczytania koordynatów do robota.

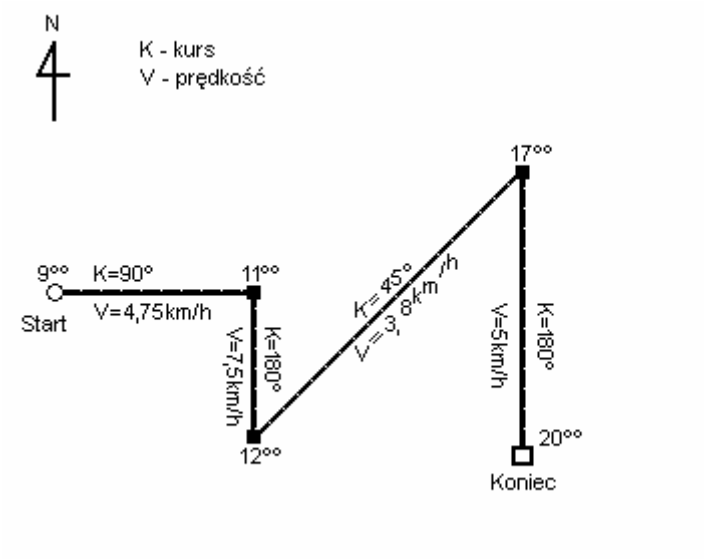
Drugą grupą metod wyznaczania położenia jest grupa pomiarów pozycji względnej (ang. relative position measurements)[3]. Wynik pomiaru jest lokalny, a położenie globalne robota jest wyznaczane jako wypadkowe wszystkich pomiarów i zadanego położenia początkowego. W praktyce stosuje się wyznaczanie położenia jako przesunięcie z ostatniej pozycji o wektor wyznaczony w danym kroku. Krokiem dla tych metod może być czas lub zdarzenie. Do tej grupy zaliczamy takie metody jak:

- odometria (ang. odometry)
- nawigacja inercyjna (ang. inertial navigation)

Odometria jest to dział miernictwa, który zajmuje się pomiarem odległości. Najbardziej popularną metodą pomiaru odległości używaną w robotyce jest tak zwana nawigacja zliczeniowa (ang. dead reckoning). Stanowi ona szkielet strategii nawigacyjnej wielu

mobilnych robotów poruszających się po powierzchni. Historycznie nawigacja zliczeniowa została wypracowana przez żeglarzy, którzy określali swoją pozycje na podstawie busoli i pomiaru przebytej trasy. Pomiar odległości na wodzie nie jest możliwy, dlatego też stosuje się metodę pośrednią jej wyznaczania. Polega ona na założeniu stałej prędkości ruchu w okresie wyznaczania położenia i wyznaczeniu odległości jako iloczynu prędkości i czasu, w jakim mierzony ruch nastąpił. Mimo dużej niedokładności pomiaru metoda ta jest używana również obecnie do wyznaczania przybliżonego położenia w przyszłości.

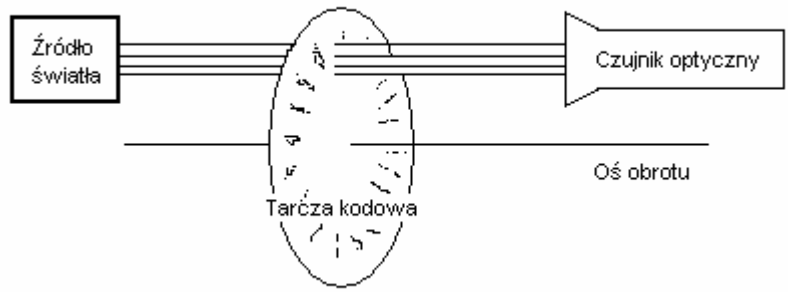
Nawigacja zliczeniowa wyznacza położenie obiektu na podstawie kierunku (kursu np. kąt względem północy geograficznej lub magnetycznej) i przebytej odległości na tym kierunku. Informacje te są nanoszone na tzw. trasę (rys. 6) na odpowiednio wyskalowanej mapie. Na tej podstawie określane jest położenie obiektu w przestrzeni.



Rys. 6. Nawigacja zliczeniowa – trasa

W robotach mobilnych metoda ta została zaimplementowana ze zmianami. Pomiar odległości jest bezpośredni za pomocą czujników, trasa jest wyznaczana w odpowiednich krokach czasowych lub zdarzeniowo. Najczęściej wykorzystywanymi czujnikami pomiaru odległości są czujniki umieszczone na kołach napędowych robota. Pomiar polega na zliczeniu liczby obrotów lub kąta obrotu danego koła, co przekłada się wprost proporcjonalnie na przebytą drogę. Jest on prosty, ale obarczony błędem kwantyzacji pomiaru i w przypadku zjawiska poślizgu nie oddaje rzeczywistego przesunięcia. Ze względu na różne sposoby poruszania się robotów stosuje się odpowiednie czujniki do pomiaru tej odległości (czujniki szczotkowe,

potencjometryczne, optyczne, magnetyczne wykorzystujące efekt Halla, indukcyjne, pojemnościowe). Najbardziej popularne i odznaczające się dużą precyzją są czujniki optyczne.

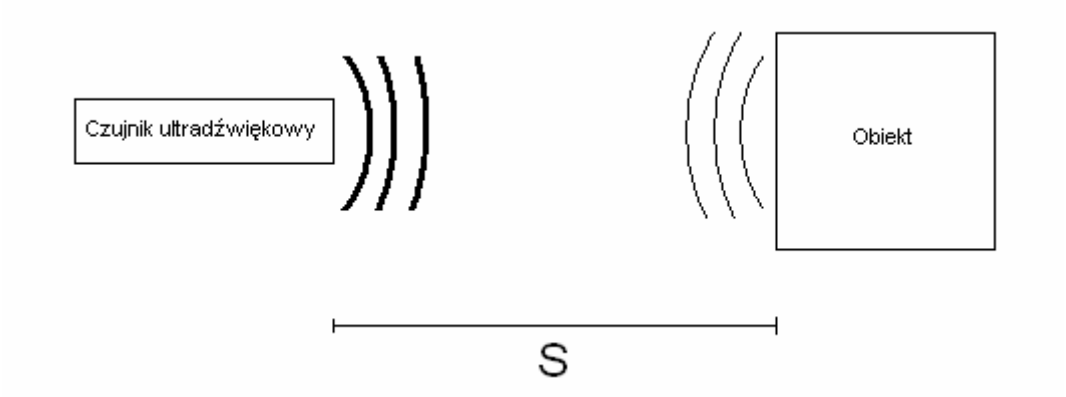


Rys. 7. Czujniki optyczne

Zbudowane są one w oparciu o odpowiednio przygotowaną tarczę umieszczoną na tej samej osi co koło jezdne robota (rys. 7), którego obroty chcemy zmierzyć. Tarcza ta ma specjalnie przygotowane otwory, która przecina wiązkę światła z nadajnika do odbiornika powodując wytworzenie sygnału. W zależności od rozwiązania może to być fala prostokątna, której impulsy należy zliczyć lub stan z sygnału n-wymiarowego, najczęściej w kodzie Gray'a (eliminacja niejednoznaczności pomiędzy sąsiednimi stanami). Dokładność jest wprost proporcjonalna do parametrów użytych czujników optycznych (rozdzielczość i częstotliwość próbkowania). Rozwiązanie takie jest jednak obarczone błędami systematycznymi wynikającymi z fizycznej różnicy pomiędzy poszczególnymi kołami robota oraz błędami nie systematycznymi wynikającymi ze zjawiska poślizgu zarówno zewnętrznego jak i wewnętrznego.

Czujniki ultradźwiękowe wykorzystują dwa zjawiska fizyczne do pomiaru odległości (pomiar pośredni): echo i efekt Dopplera. Pomiar za pomocą echa (rys. 8) polega na wyznaczeniu odległości od przeszkody w dwóch kolejnych krokach. Odległość ta wyznaczana jest na podstawie czasu, jaki upłynął od wysłania sygnału do odebrania jego części, przy założeniu, że fala ultradźwiękowa w bliskim otoczeniu rozchodzi się z tą samą prędkością oraz ciała w obrębie robota nie są idealnie czarne (nie pochłaniają fali w całości).

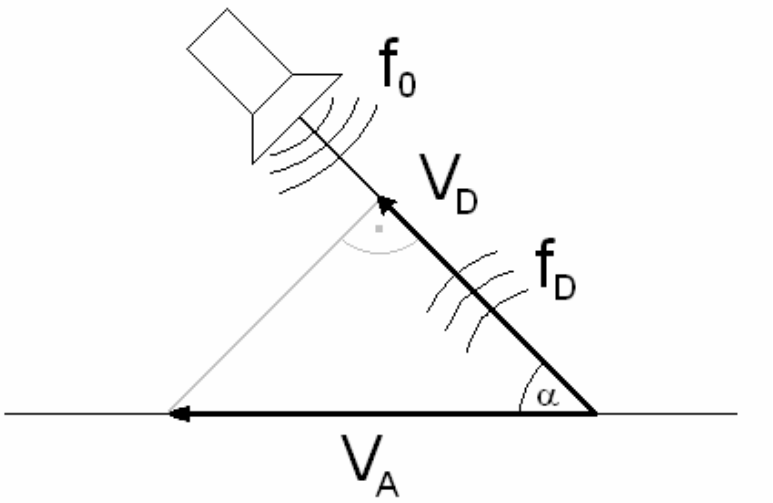
$$S = \frac{\lambda \cdot f \cdot t}{2}$$



Rys. 8. Zjawisko echa

Natomiast droga przebyta przez robota jest różnicą tych odległości. Wadą takiego pomiaru jest możliwość wystąpienia odbić od innych powierzchni, które wpływają na wiarygodność pomiaru. Metodę tę stosuje się w bardzo bliskim sąsiedztwie robota, a często w zmienionej formie służy do wykrywania przeszkód na drodze robota. Pomiar za pomocą efektu Dopplera (rys. 9) polega na pomiarze częstotliwości fali odbitej od powierzchni, po której porusza się robot, oraz względności ruchu (wzór). Wadą takiego rozwiązania jest możliwość wystąpienia zjawiska interferencji, które zmienia częstotliwość fali i przez to wprowadza błąd w pomiarze. Został on wykorzystany po raz pierwszy do kompensacji poślizgu w pojazdach rolniczych przez firmę John Deere[3].

$$V_A = \frac{V_D}{\cos\alpha} = \frac{c \cdot f_D}{2 \cdot f_0 \cdot \cos\alpha}$$



Rys. 9. Efekt Dopplera

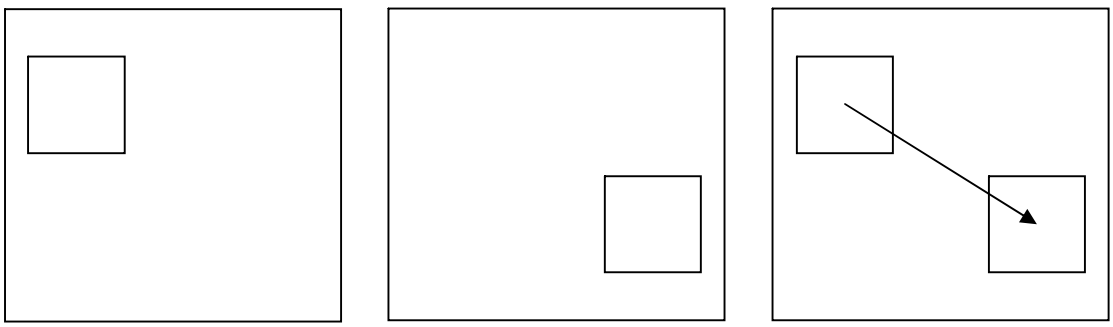
Mimo prostoty nawigacja zliczeniowa jest obarczona dodatkowo błędem skumulowanym, który zależy głównie od dokładności pomiaru w poszczególnych krokach (zdarzeniach ruchu) oraz od odległości przebytej przez robota. Dlatego też metody tej używa się jej jako uzupełnienie dla metod pomiaru pozycji bezwzględnej pomiędzy poszczególnymi markerami, jako bardziej dokładnej i wiarygodnej. Markery służą w tym przypadku do korygowania tego skumulowanego błędu pomiaru pozycji względnej.

Alternatywną metodą do nawigacji zliczeniowej jest nawigacja inercyjna opracowana na potrzeby lotnictwa oraz pocisków samonaprowadzających. Polega ona na użyciu odpowiednich czujników: żyroskopów i akcelerometrów. Położenia robota jest wyznaczane na podstawie zjawisk fizycznych: inercji obrotowej i przyspieszenia. Dokładność czujników jest uzależniona od ceny, dlatego metoda ta jest rzadko stosowana. Poza tym obarczona jest dużym błędem, gdyż wyznaczenie położenia wymaga rachunku całkowego, który sumuje błędy kolejnych kroków pomiaru. Budowa samych czujników wprowadza dodatkowo tzw. dryft, który powoduje przekłamanie wyników pomiaru. Jest to błąd systematyczny i należy go uwzględniać w wynikach pozycjonowania robota. Metoda ta jest za to wolna od zjawisk powstających na styku kół robota i powierzchni po której się porusza – samowystarczalna, ale stosunkowo wrażliwa na nierówności powierzchni ze względu na przyspieszenie ziemskie g, któremu zostaje poddawany robot.

Metody względne dają możliwość poruszania się w dowolnym środowisku. Stosuje się je również do wyznaczenia mapy nieznanego obszaru, na podstawie której możliwe będzie użycie metod bezwzględnych. Wadą tych metod jest konieczność znajomości punktu początkowego. W przeciwnym wypadku zostanie wyznaczona tylko trasa, która bez tego punktu nie określi jednoznacznie pozycji. Należy również pamiętać o określeniu punktu odniesienia do prawidłowego wyznaczenia kierunku; ma to istotne znaczenie przy tworzeniu mapy. Warunek zapewnienia współrzędnych punktu początkowego jest bardzo krytyczny, ponieważ w przypadku awarii bądź wyzerowania bufora z trasą określenie pozycji robota jest niemożliwe. Zatem informacja o ostatnim położeniu powinna być aktualizowana wraz z krokiem pomiarowym i w razie zaistnienia sytuacji błędnej mogła posłużyć jako punkt początkowy dla trasy wyznaczanej od nowa.

Istnieją również metody optyczne polegające na przetwarzaniu obrazu z kamery umieszczonej bezpośrednio na robocie (obserwacja otoczenia) lub obserwującej

środowisko działania robota (obserwacja robota). Obraz z tych kamer poddawany jest obróbce za pomocą zaawansowanych algorytmów wykorzystujących morfologię matematyczną. Wadą tych metod jest duży koszt związany z obliczeniami w przestrzeniach wielowymiarowych oraz koszt sprzętu realizujący te obliczenia (wpływający bezpośrednio na dokładność pomiaru) i urządzeń rejestrujących. Kamery umieszczone na robocie służą do wykrycia markerów, ich rozmieszczenia w przestrzeni, a następnie robot określa swoją pozycję wykorzystując metodę dopasowania modelu opisaną wyżej. Aby wykryć położenie znacznika w obrazie najczęściej wykorzystuję się algorytmy segmentacji, które znajdują kształty obiektów o tych samych właściwościach: kolor, tekstura, intensywność [5]. Metodę obserwacji otoczenia robota wykorzystuje się do wyznaczania pozycji bezwzględnej w danym środowisku na podstawie położenia wykrytego obiektu o znanych współrzędnych. Kolejnym algorytmem filtracji obrazów, który znalazł zastosowanie w samopozycjonowaniu robotów, jest algorytm dopasowania blokowego (rys. 10). Polega on na tym, że wyznacza się fragment obrazu zwany blokiem na klatce A i próbuje dopasować się go do odpowiedniego fragmentu w klatce B, która jest następna w czasie po A. Fragment klatki B, którego korelacja z blokiem wyznaczonym w klatce A jest największa, uważa się za poszukiwany blok. Na tej podstawie wyznacza się przesunięcie obiektu jako translacja bloku pomiędzy klatka A a B.



Rys. 10. Algorytm dopasowania blokowego

Wykorzystanie tego algorytmu w robotach mobilnych polega na obserwacji za pomocą kamery powierzchni, po której przesuwa się robot i analizie sąsiadujących klatek. Kolejną metodą wykorzystującą informacje zawartą w obrazie jest algorytm porównujący rozmiar obiektu [5]. Przy tych samych warunkach optycznych (stała

ogniskowa) rozmiar obiektu na obrazie jest odwrotnie proporcjonalny (ale nie liniowo) do odległości robota od tego obiektu. Pomiar w dwóch krokach pozwala na określenie odległości do tego obiektu. Użycie tego algorytmu wraz z algorytmem dopasowania blokowego pozwoli na określenie pozycji względnej robota. Szczególnym przypadkiem tej metody jest wykorzystanie jako obiektu specjalnej tarczy (przeważnie w formie szachownicy o biało-czarnych polach dla poprawy kontrastu) o znanych rozmiarach pól i określania pozycji na podstawie proporcji rozmiarów rzeczywistych i wyznaczonych za pomocą obrazu. Przykładem wykorzystania przetwarzania obrazów w pozycjonowaniu robota mobilnego może być NAVIGATOR1 opracowany w Instytucie Automatyki i Robotyki Politechniki Warszawskiej [11]. Zastosowano tu nowatorskie umieszczenie kamery wraz z lustrem kulistym, co daje możliwość obserwacji za pomocą jednej kamery całego otoczenia robota.

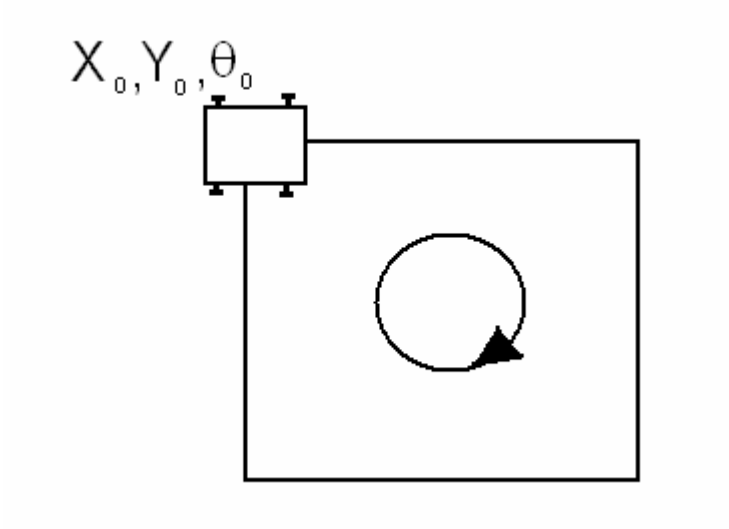
Metodą zbliżoną do obserwacji robota z użyciem kamery może być wykorzystanie systemu odbiorników sygnału nadawanego przez robota. Jest to mechanizm wykorzystywany w pelengatorach do namierzania źródła sygnału. Wymaga to systemu zewnętrznego, który będzie wyznaczał pozycje robota w danym środowisku, a następnie przekazywał ją robotowi. Wadą takiego rozwiązania jest ograniczona przestrzeń działania wynikająca z obszaru, w jakim może być dokonywany pomiar.

Metody opisane powyżej stanowią podstawowe sposoby określania położenia robotów mobilnych zarówno w znanym jak i nieznanym otoczeniu. Każda z tych metod nie jest wolna od błędów, zarówno samej metody jak i wynikające ze zmiennych czynników w danym środowisku. W celu ich wyeliminowania stosuję się kilka metod jednocześnie, co zmniejsza błąd globalny wyznaczania tego położenia. W zależności od zastosowania robota ustala się strategię nawigacji opartą o metodę bazową i metodę (metody) dodatkowe, służące do uwiarygodnienia i wzmocnienia pomiaru metody głównej.

4. Analiza problemu

Robot wydziałowy typu Pioneer3 firmy ActiveMedia Robotics jest to uniwersalna platforma do różnych zastosowań wyposażona w czujniki zliczające obroty kół napędowych. Dokładność odczytu jest na poziomie 1,2 mm [13]. Rozwiązanie to obarczone jest błędami systematycznymi i przypadkowymi związanymi z tą metodą. Do błędów systematycznych należą błędy wynikające z różnic fizycznych pomiędzy poszczególnymi kołami i umieszczonymi przy nich czujnikami. Błędy przypadkowe są ściśle związane z warunkami i zjawiskami fizycznymi. Należą do nich błędy wynikające z jazdy po nierównej powierzchni, przejechanie przez przeszkodę, wystąpienie zjawiska poślizgu. W każdym z tych przypadków do modułu sterującego zostaje dostarczona błędna informacja o rzeczywistej drodze, jaką pokonał robot. Powoduje to utratę informacji o aktualnym położeniu robota na mapie, i w konsekwencji błędnie wykonanie zadania. Ponieważ użyta metoda pomiaru należy do grupy pomiarów względnych bez korekcji położenia, pojedyncze błędy sumują się ze względu na sumowanie drogi przebytej przez robota. W zależności od środowiska błędy systematyczne lub pozorne stają się dominujące. Dla powierzchni regularnych wewnątrz budynku istotną rolę odgrywają błędy systematyczne, a dla powierzchni nieregularnych błędy przypadkowe wpływają znacząco na ogólny błąd wyznaczania położenia. Rozwiązaniem jest korekcja za pomocą dodatkowej metody bezwzględnego pozycjonowania robota, a rozmieszczenie znaczników odbywa się na drodze badań. Dodatkowo wielu badaczy proponuje algorytmy określania pozycji z uwzględnieniem błędu pomiaru. Pozycja tak określona wskazuje nie punkt, ale przybliżony obszar, w którym z dużym prawdopodobieństwem znajduje się robot. Wraz z korekcją położenia obszar ten zostaje zmniejszony do aktualnej pozycji. Algorytmy te są zbliżone do wyznaczania pozycji w nawigacji zliczeniowej używanej przez żeglarzy, którzy muszą dodatkowo uwzględniać dryf boczny łodzi. Korekcję błędów systematycznych można przeprowadzić w oparciu o wyznaczony przyrost. Proces taki nosi nazwę kalibracji robota mobilnego. W tym celu należy przeprowadzić szereg pomiarów drogi przebytej przez robota i wyznaczyć, albo za pomocą stałej wartości poprawki, albo za pomocą funkcji dowolnego rzędu. Nie ma wypracowanych procedur testowych na wykrywanie tych błędów, przez to informacje o próbach z jednym robotem nie dają odpowiedzi jak poprawić dokładność wyznaczania położenia u drugiego. Proste pomiary w jednym kierunku nie wykrywają wszystkich błędów, ponieważ nie uwzględniają obrotów

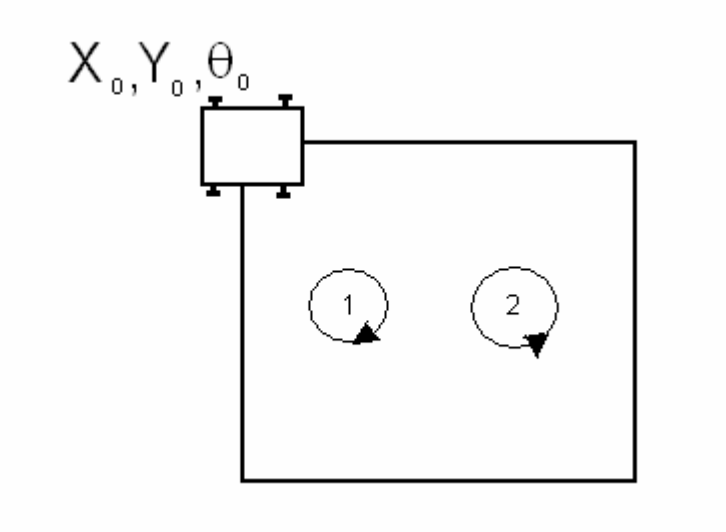
robota. W literaturze najczęstszym przykładem testowania i kalibrowania robotów jest tzw. test kwadratu (ang. the squere path experiment)(rys. 11)[3]. Polega on na przygotowaniu ścieżki dla robota (najczęściej o wymiarach 4x4 m)[2] w formie kwadratu. Następnie ustawia się robota w jednym z rogów tego kwadratu i wydaje mu się komendy na pokonanie tej ścieżki tak, aby wrócił do punktu wyjścia. Następnie mierzy się poszczególne uchybienia w przebytej trasie. Na tej podstawie można wykryć różnice i wprowadzić odpowiednie poprawki, po czym test zostaje powtórzony. Poprawki uznaje się za ostateczne, jeżeli kolejne testy wykazują błąd całkowity nie większy od przyjętego poziomu.



Rys. 11. Test kwadratu jedno-kierunkowy

Odmianą tego testu jest zaproponowany przez Borenstein'a i Feng'a test dwukierunkowy (ang. bidirectional square path experiment)(rys. 12) zwany UMBmark (University of Michigan Benchmark). Test ten pozwala wyeliminować błędy związane zarówno ze skrętem lewo- i prawo-stronnym, w przeciwieństwie do poprzedniego testu. Okazuje się, że błędy z testu w jednym kierunku nie pokrywają się z błędami w kierunku przeciwnym. Testy kwadratu zarówno jedno jaki i dwu kierunkowe nie dają jednak zadowalających efektów dla robotów z napędem rozdzielonym. Błędy przypadkowe są trudne do wykrycia, ze względu na ich jednorazowy charakter. Nie można też przeprowadzić miarodajnych testów w celu ich eliminacji.

[2] wymiary te są uzależnione od wielkości robota. Przytoczone długości odnoszą się do robotów laboratoryjnych o długości 20-60 cm.



Rys. 12. Test kwadratu dwu-kierunkowy

Niemniej jednak Borenstein zaproponował rozszerzony test UMBmark (ang. extended UMBmark). Podobnie jak w teście UMBmark robot musi pokonać wyznaczoną ścieżkę w dwóch kierunkach, z tą modyfikacją, że na ścieżce ułożone są sztuczne przeszkody wprowadzające zjawisko poślizgu. Na podstawie tego testu wyznaczają oni średni błąd orientacji bezwzględnej (ang. average absolute orientation error) jako złożenie średniego błędu w jednym i drugim kierunku błędów systematycznych i przypadkowych. Testy, mimo iż wskazują wielkość problemu nie eliminują go na tyle skutecznie, że mogą być wykorzystane jako gwarantowany sposób poprawy dokładności pozycjonowania robota. Badania przeprowadzone niezależnie przez producentów robotów mobilnych doprowadziły do wyznaczenia pewnych reguł projektowych poprawiających dokładność nawigacji zliczeniowej:

1. roboty o większym rozstawie osi są mniej podatne na błędy orientacji
2. koła w kształcie rolek, na które jest większy nacisk są bardziej podatne na poślizg w czasie zmiany kierunku jazdy
3. koła pomiarowe powinny być możliwe cienkie i nie elastyczne; pożądanym jest też, żeby pomiar odległości nie odbywał się na kołach napędowych
4. dokładność odometrii jest większa przy napędzie synchronicznym niż rozdzielnym.

Zatem oczekiwaniem jest eliminacja (jeżeli możliwa) lub redukcja błędów systematycznych i przypadkowych w robotach wykorzystujących nawigacje zliczeniową jako podstawową (i/lub jedyną) metodę nawigacji. Do redukcji błędów systematycznych używa się dodatkowych kół pomiarowych umieszczonych

bezpośrednio na robocie lub dołączanej przyczepie, systematycznej kalibracji na podstawie wyników badań (zarówno stałych wartości jak i funkcji). Redukcja błędów przypadkowych nie jest prosta. Przy założeniu, że zjawiska poślizgu występuje jako zjawisko lokalne można użyć drugiego robota sprzężonego i określać pozycję robotów jako pomiar wzajemnych położeń.

W mojej pracy chciałbym zaproponować redukcję zjawiska poślizgu za pomocą dodatkowych czujników odometrycznych niezwiązanych fizycznie z podłożem. Należy przypuszczać, że takie rozwiązanie wyeliminuje to niekorzystne zjawisko lub w dużym stopniu je zredukuje. Dodatkowo mam zamiar zbadać wpływ ilości i rozmieszczenia tych czujników na dokładność pomiarów oraz ich wiarygodność. Uwzględniając wzajemne położenie czujników jako niezmienne w czasie można wykorzystać również do uwiarygodnienia pomiarów z poszczególnych czujników.

5. Projekt techniczny

5.1. Projekt czujnika odometrycznego

W mojej pracy chciałbym wykorzystać urządzenia wskazujące – myszy komputerowe. Urządzenia te powstały w odpowiedzi na dynamicznie rozwijający się rynek programów sterowanych z poziomu menu (ang. menu driven software)[8]. Pierwsza mysz została wynaleziona przez Douglasa Engelbarta w 1963 r. [14]. Początkowo były one nieruchome z ruchomą kulą, którą użytkownik wprawiając w ruch mógł wyzwolić przesuwanie kursora na ekranie. Kolejnym krokiem było zastosowanie tzw. myszek kulkowych, w których ruch całej myszy jest rejestrowany za pomocą czujników optycznych i tarczy poruszanych za pomocą kulki umieszczonej wewnątrz urządzenia. Urządzenia te wymagały chropowatej powierzchni, która zapewniała ruch kulki. Kolejnym etapem jest zastąpienie elementów mechanicznych, które należało utrzymywać w czystości, zestawem czujnika optycznego oraz źródła światła. Urządzenia te noszą nazwę myszy optycznych. Najnowszym typem urządzeń wskazujących optycznych są myszy laserowe, gdzie źródłem światła jest promień lasera. Wraz ze zmianą technologii odczytu ruchu zwiększała się dokładność pomiaru, którą wyraża się w jednostkach CPI (ang. counts per inch). Jednostka ta mówi na ile punktów na cal jest w stanie zarejestrować mysz, lub po odwróceniu wartości jaki minimalny ruch jest w stanie dane urządzenie zarejestrować. Obecnie standardem jest wartość 400CPI (0,06mm), choć można już kupić mysz o precyzji 3200CPI (0,008mm). Podobnie zmieniał się też interfejs z komputerem, ze względu na szybkość działania oraz precyzję. Przyjęte rozwiązanie w systemach operacyjnych narzuca wymagania na format danych przekazywany z urządzenia. Każdy raport z myszy składa się z 3 bajtów. Pierwszy bajt zawiera informacje o wciśniętych przyciskach i dla protokołu PS/2 informacje o przepełnieniu rejestru danych współrzędnej X i/lub Y oraz znak danych zawartych w tych rejestrach. W interfejsie USB informacje te nie są dostępne w raporcie, ponieważ szybkość działania urządzeń wskazujących i odczytu przez hosta USB jest na tyle duża, w stosunku do szybkości poruszania myszy przez użytkownika, że pozwala na nie umieszczanie tych informacji w raporcie. Kolejne dwa bajty to informacja o wartości przesunięcia X i Y. Daje to 255 możliwości zapisu ruchu w jednym kierunku. Aby móc wskazać dodatkowo zwrot, przedział został podzielony równomiernie tak, że zakres od 1-127 oznacza kierunek dodatni (w dół lub w prawo) a zakres 128-255 oznacza kierunek ujemny (w górę lub w lewo). Jest to bezpośrednio

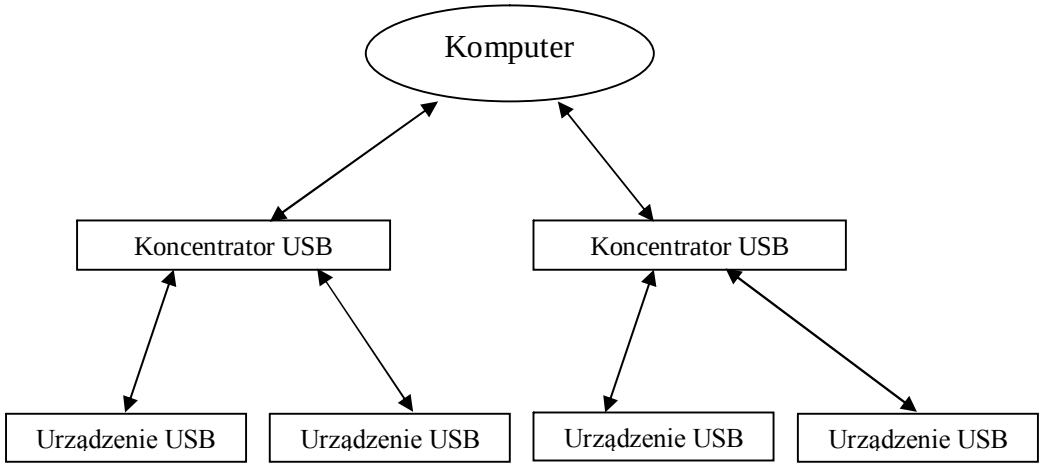
związane z przyjętymi współrzędnymi ekranu, gdzie górny lewy róg stanowi początek układu. Mysz wysyła do komputera kolejne raporty, które są umieszczane w buforze. System operacyjny musi zapewnić sobie proces, który w określonych chwilach czasowych odczyta te dane. Czas ten jest silnie deterministyczny i wynika z przyjętego protokołu transmisji (transmisja bez potwierdzenia nadawania i odbioru). W myszach opartych o interfejs USB problem został rozwiązany poprzez wprowadzenie kolejki FIFO, która zapewnia przekazanie wszystkich raportów dostarczonych przez urządzenie wskazujące. Poniżej zestawienie parametrów myszy optycznej w oparciu o układ ADNS-2610[1].

Tabela 1 Zestawienie parametrów układu ADNS-2610

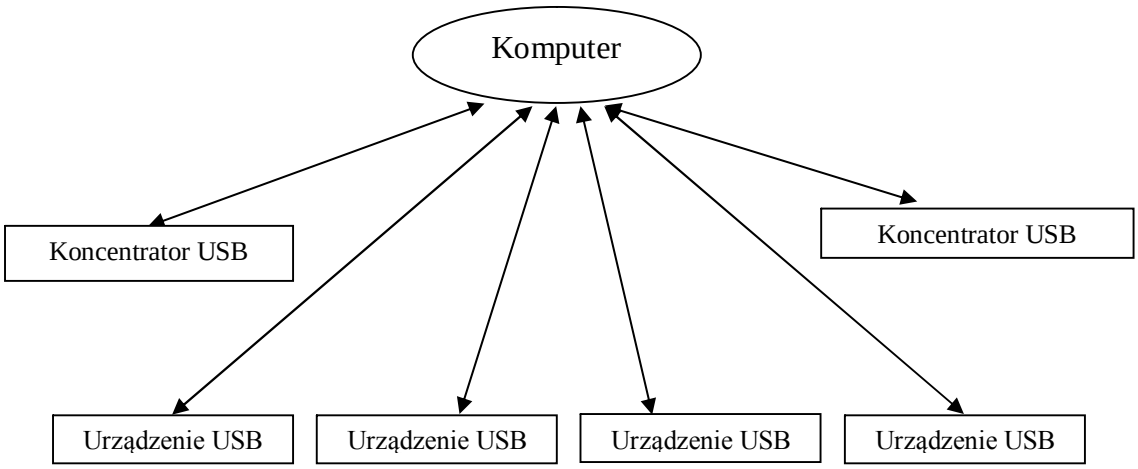
LP	Parametr	Wartość
1	Szybkość odczytu kamery	1500 klatek/s
2	Dokładność pomiaru	0,0635 mm
3	Max prędkość obiektu	30,48 cm/s
4	Max przyspieszenie obiektu	245,25 cm/s2

W mojej dalszej pracy skupię się na myszach optycznych, ponieważ celem jest stworzenie systemu czujników, które będą alternatywą dla czujników zliczających obroty kół robota. Dla tego typu urządzeń wskazujących obecnie na rynku dostępne są myszy z interfejsem PS/2 i USB. Ze względu na ograniczoną liczbę portów PS/2 w komputerze oraz konieczność wykorzystania ich przez klawiaturę i mysz do sterowania systemem i aplikacjami, zająłem się adaptacją urządzeń przez port USB. Jest to nowoczesna magistrala pozwalająca na podłączenie do 127 urządzeń do jednego kontrolera. Powstała ona w odpowiedzi na zapotrzebowanie rynku telekomunikacyjnego, i w założeniu miała służyć do transmisji danych w sieciach telekomunikacyjnych. Opis magistrali z 1998 roku [6] stał się standardem telekomunikacyjnym (v1.1) i został zaadoptowany przez kluczowych producentów sprzętu komputerowego, przez co stał się standardem komputerowym. Początkowo prędkość transmisji została ustalona na 1,5Mb/s dla urządzeń wolniejszych i 12Mb/s dla urządzeń wymagających większego strumienia. Norma z 2000 roku zwiększa znacząco prędkość transmisji do 480Mb/s wprowadzając wersje 2.0. Ze względu na kompatybilność wsteczną (urządzenia standardu 1.1 muszą prawidłowo pracować w nowych kontrolerach magistrali) możliwa jest transmisja w dwóch klasach szybkości.

Parametry transmisji są „negocjowane” podczas zgłaszania się urządzenia USB do kontrolera. Odbywa się to dwuetapowo: elektrycznie i przez zmianę szybkości transmisji. Etap elektryczny polega na „podpięciu” przez urządzenie podłączane do hosta linię D- lub D+ do dodatniego bieguna zasilania. Kontroler na tej podstawie decyduje, w jakiej wersji (1.1 czy 2.0) jest urządzenie. Drugi etap polega na rozpoczęciu transmisji z minimalną prędkością, która jest zwiększana do momentu, aż urządzenie nie będzie w stanie odebrać informacji. Maksymalna szybkość jest rejestrowana w kontrolerze jako szybkość transmisji, która będzie używana do komunikacji z urządzeniem. Transmisja danych odbywa się asynchronicznie za pomocą dwóch przewodów sygnałowych. Nie ma tu rozróżnienia na sygnał wyzwalający i sygnał danych, ponieważ przyjęto rozwiązanie różnicowe. Sygnał jest wyznaczany na podstawie różnicy stanu linii D- i D+. Dzięki takiemu rozwiązaniu uniknięto problemu synchronizacji zegarów, ich stabilności, a wiarygodność przesyłanych tak danych jest większa, gdyż transmisja różnicowa jest bardziej odporna na przekłamania, które mogą wystąpić w transmisji. Kod używany w transmisji nosi nazwę NRZI – odwrotny kod bez powrotu do zera (ang. non-return to zero inverted). Jest on kodem samosynchronizującym (nie wymaga sygnału zerowego) i łatwym do fizycznej implementacji (jako sygnał elektryczny). Dodatkowym atutem jest możliwość dołączania urządzeń w dowolnej chwili pracy komputera, bez potrzeby jego resetowania, ponieważ każde podłączenie rejestruje urządzenie w systemie dynamicznie. Kontroler odpowiedzialny za ten proces dodaje do listy urządzeń aktualny adres urządzenia oraz zapewnia dołączenie odpowiedniego sterownika. Pozawala to aplikacjom na swobodny dostęp do tych urządzeń odwołując się za pomocą tego adresu do żądanego urządzenia. Duża popularność magistrali a co za tym idzie duża liczba dostępnych urządzeń spowodowała, że w systemie WindowsXP dostępne są sterowniki do większości z tych urządzeń, co pozwala na dużą elastyczność i szybkość tworzenia aplikacji wykorzystujących urządzenia USB. Mimo fizycznej złożoności implementacji (rys. 13) aplikacja-urządzenie, tworzenie programów odwołujących się do danego urządzenia nie jest skomplikowane, ponieważ nie trzeba oprogramowywać transmisji (jak to jest w przypadku RS-232C). Za pomocą funkcji WinAPI mechanizm transmisji staje się transparentny dla programisty (rys. 14), ponieważ stworzenie tego mechanizmu jest wykonywane przez system na podstawie informacji z kontrolera magistrali USB w oparciu o dedykowane sterowniki dla urządzenia.



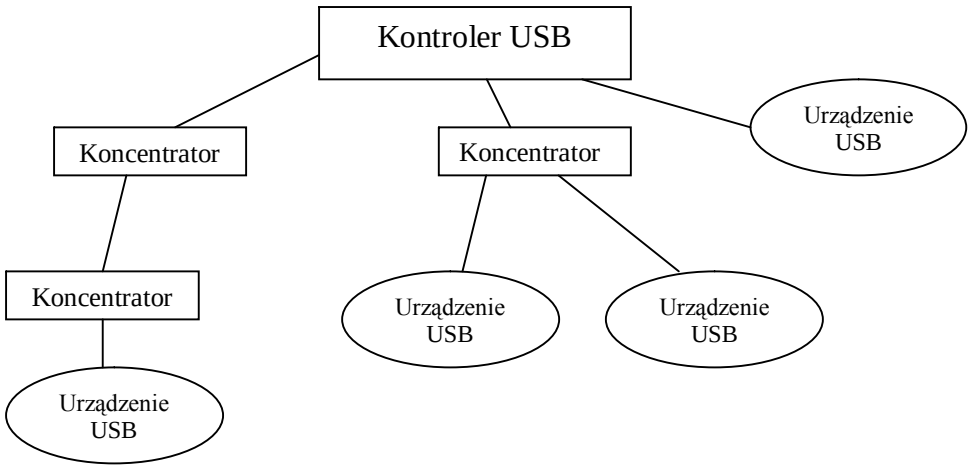
Rys. 13. Fizyczna implementacja magistrali USB



Rys. 14. Logiczna implementacja magistrali USB

Jest to możliwe dzięki przyjętemu wspólnemu rozwiązaniu dla rejestracji urządzeń w systemie WindowsXP [9]. Rozwiązanie to wymusza tworzenie sterowników w oparciu o ustalony interfejs, ponieważ komunikacja w systemie Windows odbywa się za pomocą komunikatów; w przypadku urządzeń jest to grupa komunikatów IRP, które powodują uruchomienie odpowiednich funkcji systemowych z grupy IOCTL. Kolejnym atutem magistrali USB jest możliwość stosowania koncentratorów (ang. hub). Koncentrator umożliwia podpięcie do jednego portu kilku urządzeń, stanowiąc jego

fizyczną multiplikacje. Nie zmienia to jednak głównego ograniczenia na liczbę urządzeń podpiętych do jednego kontrolera (128), ponieważ każdy koncentrator też jest takim urządzeniem, przez co zmniejsza liczbę dodatkowych urządzeń. Zaletą koncentratorów jest możliwość ich kaskadowego połączenia. Poniższy rysunek (rys. 15) ilustruje przykładowe połączenie.



Rys. 15. Kaskada koncentratorów USB

Urządzenia podpinane do magistrali przeważnie korzystają z energii elektrycznej dostarczanej z portu. W procesie inicjalizacji urządzenie określa, jaki prąd będzie pobierać z portu (jest to wielokrotność 2 mA ale nie więcej niż 500 mA). Urządzenie, które zgłosi zapotrzebowanie na energię przekraczające możliwości portu nie zostanie zarejestrowane w systemie. Ograniczenie to zostało wprowadzone w celu eliminacji możliwego uszkodzenia elektrycznego portu oraz zapewnieniu prawidłowego działania zarejestrowanych urządzeń. Dla urządzeń wymagających większej energii (np. dysk zewnętrzny) wprowadza się dodatkowe źródło zasilania. Dodatkowo możliwe jest użycie koncentratorów aktywnych, posiadających własne zasilanie. Rozwiązane takie jest dedykowane dla urządzeń mobilnych (laptop, PDA), gdzie ograniczenie strat energii wpływa na czas pracy urządzenia.

Rozszerzeniem klasy urządzeń USB są urządzenia klasy HID (ang. human input device), do których należą urządzenia wskazujące. Proces instalacji takiego urządzenia rozpoczyna się w momencie podłączania do komputera (tak jak urządzenia USB). Pomiędzy hostem USB a urządzeniem wysyłane są informacje, zawierające dane o sprzęcie. Na ich podstawie urządzenie jest rejestrowane w systemie, dołączany jest

sterownik obsługi i tworzony stos do wymiany danych [2][9]. Urządzenia klasy HID zostały podzielone na grupy, ze względu na przeznaczenie:

HID_DEVICE_SYSTEM_MOUSE	Urządzenie wskazujące (mysz)
HID_DEVICE_SYSTEM_KEYBOARD	Klawiatura (oraz klawiatura numeryczna)
HID_DEVICE_SYSTEM_GAME	Joystick lub game pad
HID_DEVICE_SYSTEM_CONTROL	Urządzenie systemowe
HID_DEVICE_SYSTEM_CONSUMER	Pozostałe

W procesie rejestracji urządzenia klasy HID są przypisywane do jednej z wyżej wymienionych grup, za pośrednictwem identyfikatora przekazanego w danych inicjujących w tzw. opisie raportu (ang. report descriptor). Następnie na podstawie opisu raportu tworzony jest protokół wymiany danych. Komunikacja z urządzeniem wymaga posiadania systemowego identyfikatora (uchwyt ang. handle) do pliku wymiany danych. Uzyskanie tego identyfikatora następuję poprzez próbę otwarcia pliku o adresie, który został przypisany w procesie rejestracji urządzenia, za pomocą funkcji WinAPI *CreateFile*. Funkcja ta jest zaimplementowana w bibliotece systemowej *kernel32.dll*. Uzyskanie adresu urządzenia w systemie wymaga wykonania poniższych kroków (za pomocą dedykowanych bibliotek: *hid.dll, setupapi.dll, kernel32.dll*)[2]:

1. Uzyskanie GUID (globalny unikalny identyfikator urządzenia) klasy HID za pomocą funkcji *HidD_GetHidGuid*
2. Uzyskanie struktury danych, przechowującej informacje o urządzeniach klasy HID za pomocą funkcji *SetupDiGetClassDevs*
3. Uzyskanie struktury danych, przechowującej informacje o pojedynczym urządzeniu za pomocą funkcji *SetupDiEnumDeviceInterfaces*
4. Uzyskanie adresu urządzenia w systemie za pomocą funkcji *SetupDiGetDeviceInterfejsDetail*

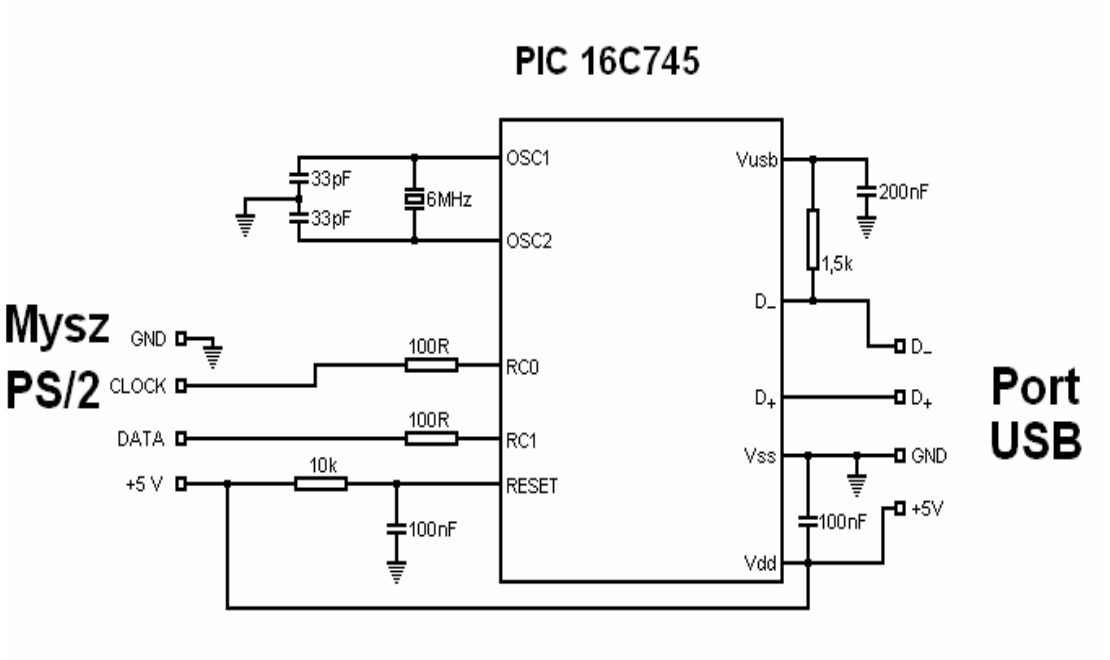
Sposób ten nie daje jednak dostępu do wybranego urządzenia. Należy zatem kolejno sprawdzić czy żądane urządzenie znajduje się pod uzyskanym adresem na podstawie danych urządzenia dostępnych strukturze urządzenia zarówno klasy USB jak i HID. Informacje, które mogą zidentyfikować urządzenie jednoznacznie są identyfikatory producenta i produktu (VendorID i ProductID), nazwa producenta i urządzenia oraz wersja. Wadą takiego rozwiązania jest możliwość jednoznacznego identyfikowania urządzenia w systemie, jeżeli jest pojedyncze. Dwa urządzenia takie same (mające te

same identyfikatory producenta i produktu) mogą być w tym algorytmie nierozróżnione, mimo iż system zarejestrował je jako osobne urządzenia. Rozwiązaniem tego problemu jest dołączenie kolejnych urządzeń po identyfikacji poprzednich, ponieważ identyfikator do komunikacji z urządzeniem w systemie jest nadawany raz przy rejestracji i nie zmienia się przez okres podłączenia urządzenia do magistrali (jest unikalny globalnie - GUID). Informacje o identyfikatorze producent i produktu są pozyskiwane za pomocą funkcji *HidD_GetAttributes* i *HidP_GetCaps*. Sama komunikacja z urządzeniem odbywa się już za pomocą funkcji WinAPI: *ReadFile* i *WriteFile* według przyjętego protokołu. System WindowsXP zapisuje identyfikatory otwartych plików do specjalnej struktury, dlatego po skończeniu pracy z urządzeniem powinno się zamknąć używany plik i zwolnić uchwyt do niego. Pozwoli to na bezproblemowy dostęp innym aplikacjom do naszego urządzenia oraz pozwoli zwolnić pamięć operacyjną zajmowaną przez strukturę do obsługi tego pliku.

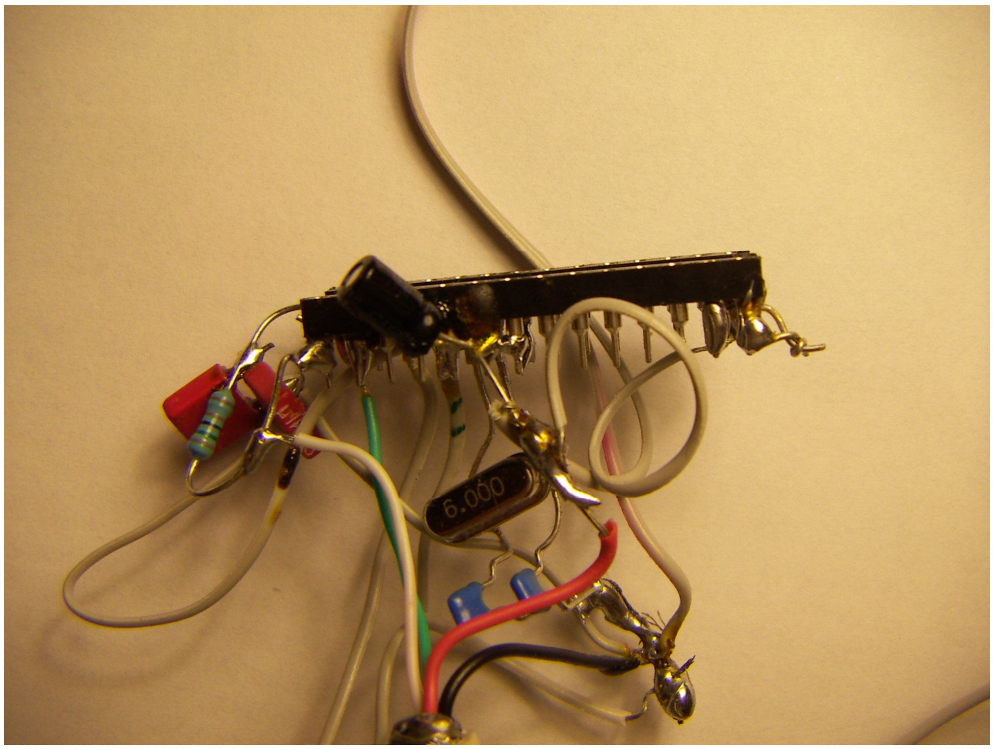
Ograniczeniem systemu WindowsXP, które nie pozwala użyć bezpośrednio myszy USB jako czujników, jest przypisywanie urządzeń do grup klasy HID. Co prawda system pozwala na dołączenie kilku myszy, ale ze względu na ich przeznaczenie wątek obsługi wejścia systemu wywłaszcza każde nowo podłączone urządzenie wskazujące i mimo identyfikacji myszy w klasie HID nie jest ona dostępna dla innych aplikacji [9]. Grupa HID_DEVICE_SYSTEM_GAME nie ma już takich ograniczeń i dla aplikacji użytkownika urządzenia te są w pełni dostępne. Skłoniło mnie to do poszukiwania rozwiązania, które z jednej strony zmieni przeznaczenie urządzenia w systemie, a z drugiej będzie transparentne dla raportów z myszy (będzie je przekazywało w niezmienionej postaci). Zmiana sposobu przypisywania do grupy jest zaszyta w sterowniku klasy HID (*hidclass.sys*). Jest to sterownik systemu WindowsXP i prawa autorskie oraz patentowe nie pozwalają na ingerencje w kod sterownika. Poza tym źródła tego sterownika nie są udostępnione. Rozwiązanie, które przyjąłem polega na użyciu translatora w oparciu o mikrokontroler, który „zgłosi" się do systemu jako urządzenie innej grupy oraz będzie przekazywało raporty z myszy bezpośrednio do komputera. Ze względu na trudno dostępne układy mikrokontrolerów posiadające wbudowane dwa porty USB, zdecydowałem się na użycie myszy z protokołem PS/2. Na rynku dostępne są translatory PS/2-USB służące do podpinania klawiatury i myszy PS/2 do portu USB, ale nie nadają się do tego zastosowania. System wykrywa je jako urządzenia z odpowiednich grup urządzeń wskazujących lub sterujących. Protokół PS/2 nie wymaga specjalnie dedykowanego układu portu szeregowego (SP). Ze względu na

potrzebę wykrywania zbocza narastającego wystarczy użyć układu porównującego (CCP), a w pętli głównej programu mikrokontrolera zapewnić cykliczne sprawdzanie stanu na dedykowanych portach. Układem posiadającym te właściwości jest mikrokontroler PIC16C745 firmy Microchip, którego użyłem do budowy translatora PS/2 - USB. Program wsadowy mikrokontrolera napisałem w oparciu o informacje umieszczone przez producenta na stronie http:\\www.microchip.com w opracowaniu [4]. Przy użyciu dedykowanego programatora oraz programu IcProg kod został załadowany do mikrokontrolera. Układ ten posiada pamięć programu typu OTP (ang. one time programable) i jest możliwe tylko jednorazowe wgranie kodu. Dodatkowo układ wyposażony jest w złącze ICSP (ang. in circuit serial programming), które pozwala na programowanie mikrokontrolera w docelowym układzie bez potrzeby jego przekładania do programatora. W celu załadowania kodu mikrokontroler musi przejść w stan programowania. Odbywa się to za pomocą podania odpowiedniego napięcia na nóżkę Vpp (w przypadku mikrokontrolera PIC16C745 jest to 11,75-12,25V). Na nóżki PGC i PGD podawane są sygnały zegara i danych zawierające kod programu od adresu 0h do maksymalnego w danym układzie. Program *IcProg* pozwala na używanie programatorów typu JDM do programowania mikrokontrolerów rodziny PIC. Są one bardzo popularne i tanie w stosunku do dedykowanego programatora firmy Microchip (koszt około 1000PLN). Wykorzystują one właściwości portu RS-232C i podwajacza napięcia do uzyskania napięcia programowania. Parametry tego źródła mogą okazać się niewystarczające ze względu na duży pobór prądu podczas programowania. Powoduje to wyjście mikrokontrolera ze stanu programowania oraz dla układów OTP może wiązać się ze stratą mikrokontrolera. Zaleca się na czas programowania dołączania dodatkowego źródła zasilania (większość profesjonalnych lub półprofesjonalnych programatorów posiada taką opcje).

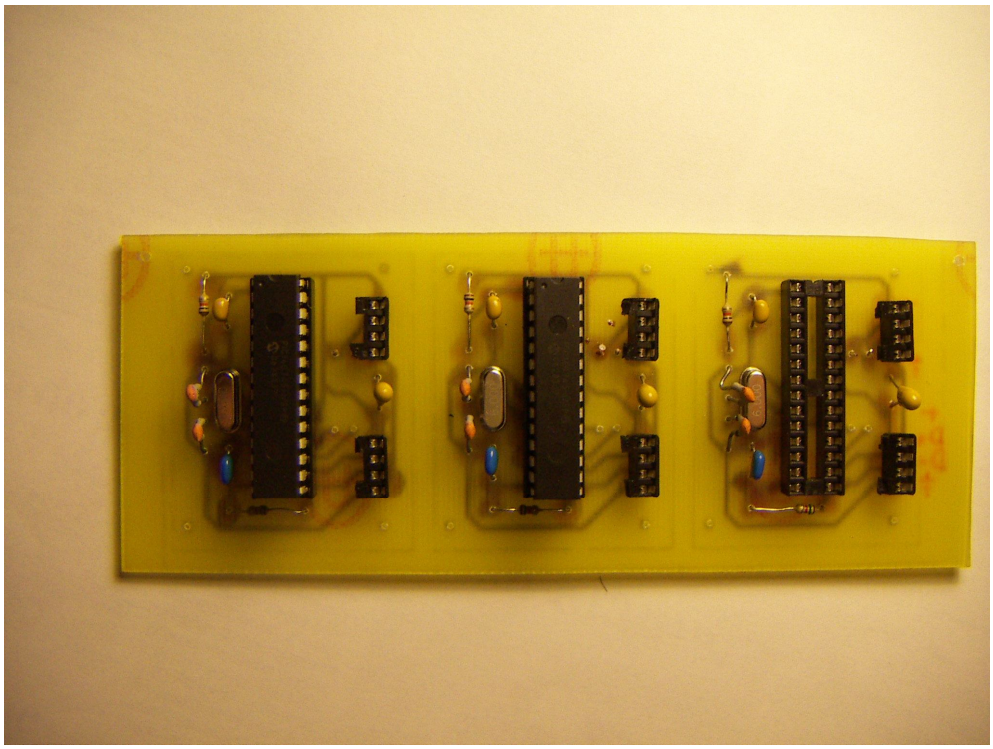
Zaprogramowany mikrokontroler został umieszczony wraz z elementami elektronicznymi w dedykowanym układzie (rys. 16). Wersja prototypowa (rys. 17) została wykonana w oparciu o technikę tzw. „pajączka", która pozwala na swobodną zmianę połączeń oraz elementów w układzie. Daje to dużą swobodę i możliwości w fazie projektowej. Badania i testy z prototypem pozwoliły na opracowanie docelowej płytki drukowanej (rys. 18), która po zmontowaniu układu powoduje zwiększenie wytrzymałości układu translatora. Jest to ważne ze względu na charakter działania czujników umieszczonych na robocie mobilnym.



Rys. 16. Schemat ideowy translatora USB - PS/2



Rys. 17. Wygląd prototypu translatora



Rys. 18. Wygląd docelowy trzech translatorów

Zadaniem stworzonego w pracy translatora jest zgłoszenie urządzenia w systemie jako urządzenia grupy HID_DEVICE_SYSTEM_GAME oraz przekazywanie raportów z myszy do komputera. Rozwiązanie takie pozwoliło obejść problem ograniczonego dostępu do urządzenia wskazującego w systemie WindowsXP. Zatem czujnik odometryczny używany w dalszych badaniach jest zbudowany w oparciu o mysz optyczną na PS/2, translatora PS/2 – USB. Daje to uniwersalność i niezależność od liczby dostępnych portów dla urządzeń.

5.2. Model obliczeń pozycji globalnej

Wyznaczenie pozycji globalnej [3] wymaga sformułowania funkcji położenia czujników do wyznaczania tej pozycji. Rozmieszczenie czujników odometrycznych względem robota nie jest silnie zdeterminowane, dlatego też należało zbudować uniwersalny model matematyczny, który pozwoli na testowanie różnych ustawień tych czujników bez potrzeby modyfikacji zarówno aplikacji jak i samego modelu.

[3] położenie środka robota w przestrzeni

Algorytm wyznaczania środka przesunięcia układu sztywnego

Autorzy w pracy [7] zaproponowali algorytm FCR (ang. finite centre of rotation) wyznaczania środka obrotu układu sztywnego przy jego przemieszczaniu w oparciu o punkty rozkładu tego układu. Środek ten jest wyznaczany jako aproksymacja średniokwadratowa na podstawie dopasowania rozkładu czujników z poprzedniego kroku do rozkładu odczytanego za pomocą czujników. Metoda ta uwzględnia możliwe zaszumienia wszystkich punktów pomiarowych, ale nie jest odporna na kumulacje błędów pomiaru, ponieważ wyznacza pozycje na podstawie dwóch sąsiednich pomiarów. Obliczenie są wykonywane w przestrzeni liczb zespolonych, dlatego należy współrzędne rozkładu zamienić według wzoru:

$$c_k(t) = x_k(t)+y_k(t)*i$$

gdzie k=1,2,3,...,n

Wyznacznie wektora przesunięcia punktu środkowego rozkladu wymaga odwrócenia macierzy współrzędnych rozkładu jednego z kroków. Macierz ta jest macierzą prostokątną, dlatego autorzy [7] proponują wykonać operacje pseudoinwersji. Wykorzystują oni do tego przekształcenia funkcja *pinv* w środowisku Matlab, która bazuje o algorytm Moore-Penrose. Algorytm ten wykorzystuje rozkład SVD (ang. singular value decomposition) macierzy według wartości szczególnych. Dla liczb zespolonych rozkład ten wymaga dużej złożoności obliczeniowej ze względu na sam rozkład, jak i również na obliczenia w przestrzeni zespolonej. Rozwiązaniem tego problemu może być metoda Greville [10], która jest uogólnioną metodą odwracania macierzy dla algorytmu Moore-Penrose. Jest ona prostsza numerycznie, a dla wektora zespolonego, który jest rozpatrywany w tym przypadku, sprowadza się do jednego kroku wyznaczenia macierzy pseudoodwrotnej, dlatego została ona zaimplementowana w mojej pracy.

Algorytm w oparciu o punkty sprzężone

W mojej pracy przyjąłem model wektorowy polegający na wyznaczaniu pozycji globalnej jako najbardziej prawdopodobnej z serii pozycji wyznaczonych na podstawie dwóch punktów sprzężonych. Zostało to podyktowane faktem, że przy każdym pomiarze otrzymujemy serię czterech rozwiązań, na podstawie których należy

wyznaczyć pozycję globalną. Wartość oczekiwana określona jako średnia arytmetyczna z tej serii nie jest dobrym rozwiązaniem tego zagadnienia, ponieważ nie uwzględnia ona błędów przypadkowych i jest ściśle związana z rozkładem naturalnym, w którym elementy serii są wynikiem powtarzalnych doświadczeń (proces pomiaru translacji pojedynczego czujnika robota nie można zamodelować takim rozkładem). Dlatego też, przyjąłem rozwiązanie bazujące na medianie wektorowej. Rozwiązanie takie zwiększa wiarygodność pozycji globalnej robota, eliminując przypadkowy błąd pomiaru czujnika. Ze względu na parzystą liczbę wektorów rozwiązań cząstkowych rozwiązanie najmniej prawdopodobne nie jest brane do wyznaczenia wartości środkowej. Odrzucone rozwiązanie jest wyznaczane na podstawie sumy odległości rozwiązania od innych rozwiązań przy założeniu bliskości sąsiedztwa.

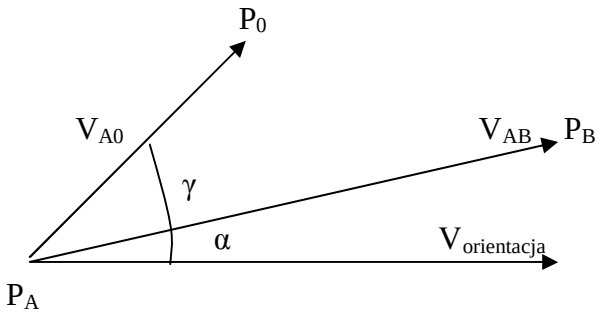
$$\underset{k}{\exists!}\ f_{celu}(k) = \max\left(\sum_{i=1}^{n} \left|X_k - X_i\right|\right)$$

Pozostałe trzy rozwiązania są uważane za wiarygodne z tym samym prawdopodobieństwem. Pozwala to wyeliminować tylko jedno ze wskazań czujników, które w oparciu o wyniki pozostałych uważa się za błędne. Jeżeli wszystkie wskazania czujników są poprawne to odrzucenie jednego pomiaru nie ma wpływu na wyznaczenie pozycji globalnej robota. System taki jest wykorzystywany w różnych metodach arbitralnych, gdzie wyniki najmniej prawdopodobne są odrzucane (przykładem może być system not w skokach narciarskich, gdzie notę maksymalną jak i minimalną się odrzuca). Algorytm taki nie eliminuje jednak błędów przypadkowych, które mogą wystąpić jednocześnie dla kilku czujników.

Po odrzuceniu jednego rozwiązania wyznaczam z pozostałych medianę wektorową w analogiczny sposób. Pozycją globalną robota jest rozwiązanie, którego suma odległości od pozostałych jest najmniejsza; rozwiązanie takie uważa się za środkowe dla serii.

$$\underset{p}{\exists!}\ f_{celu}(p) = \min\left(\sum_{\substack{i=1 \\ i\neq k}}^{n} \left|X_p - X_i\right|\right)$$

Rozwiązania cząstkowe [4] są wyznaczane na podstawie dwóch czujników sprzężonych ze sobą, przy założeniu stałego ich rozkładu względem robota. Poniższy rysunek (rys. 19) ilustruje rozkład punktów i wektorów służący do wyznaczania tego rozwiązania.



Rys. 19. Metoda wyznaczania rozwiązania cząstkowego.

Pozycje globalną P_0 w tym rozwiązaniu wyznacza się jako przesunięcie o wektor V_{A0} względem współrzędnych czujnika P_A, które są ustalane i aktualizowane w procesie pomiaru jego translacji.

$$\mathbf{P_0 = P_A + V_{A0}}$$

Należy zatem wyznaczyć ten wektor. Ze względu na niezmieniającą się pozycję czujnika względem środka robota w lokalnym układzie odniesienia, długość tego wektora jest znana. Należy jedynie wyznaczyć kąt tego wektora względem przyjętego kierunku na mapie. Kierunek ten jest prezentowany na rysunku jako wektor $V_{orientacja}$. Jest on obliczany jako suma dwóch kątów: na podstawie współrzędnych dwóch sprzężonych czujników P_A i P_B kąt α względem przyjętego kierunku $V_{orientacja}$ oraz stałej poprawki o kąt γ, która jest niezmienna i wynika z rozkładu czujników względem siebie oraz środka robota.

$$\gamma = \arctan\left(\frac{y_B - y_A}{x_B - x_A}\right)$$

Wyznaczenie współrzędnych wektora V_{A0} jest realizowane za pomocą przejścia z współrzędnych biegunowych na współrzędne kartezjańskie.

[4] wyznaczenie pozycji globalnej robota na podstawie współrzędnych czujnika

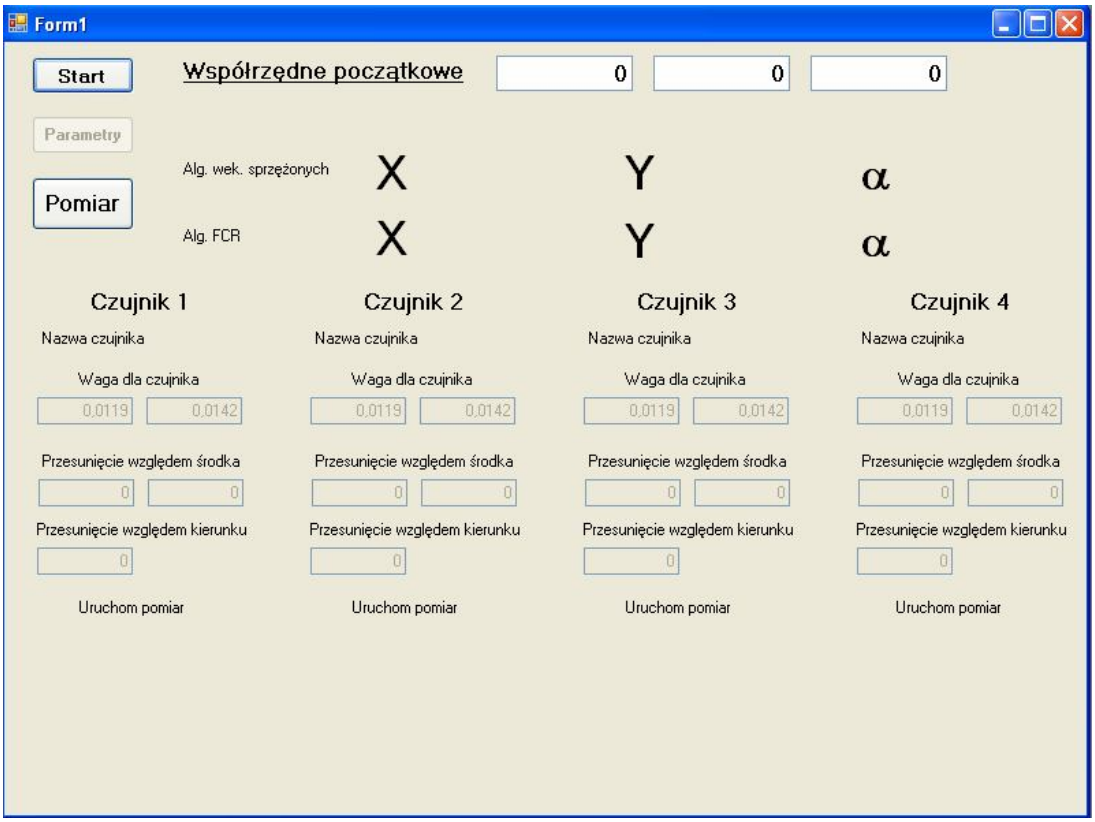
$$\mathbf{V_{A0}x = |V_{A0}| \cos(\alpha+\gamma)}$$
$$\mathbf{V_{A0}y = |V_{A0}| \sin(\alpha+\gamma)}$$

W mojej pracy przyjąłem takie pary czujników sprzężonych: 1 z 2, 2 z 3, 3 z 4 i 4 z 1. Powiązania te są niezmienne w aplikacji, przez co wymuszają używania czterech czujników.

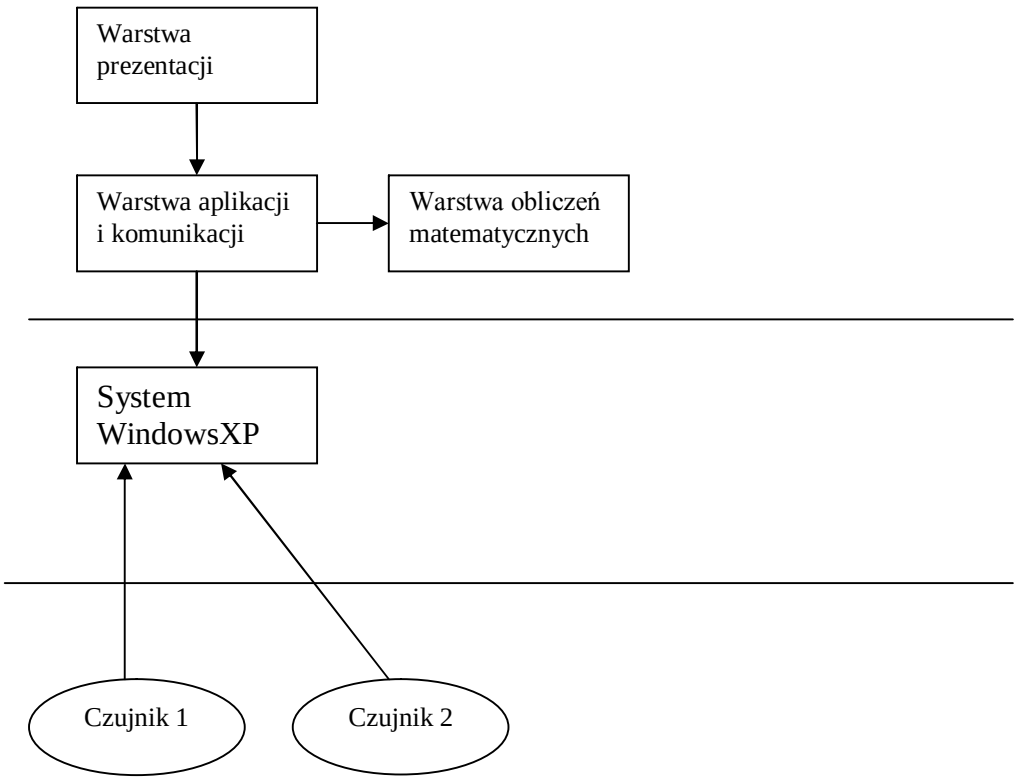
Moduł obliczeń został zaimplementowany jako osobna klasa, w celu logicznego rozdzielenia pomiędzy częścią obsługi czujników a częścią wyznaczania rozwiązania pozycji globalnej.

5.3. Program MouseReader

Czujnik odometryczny opisany w poprzednim podrozdziale stanowi urządzenie zewnętrzne. Uzyskanie informacji z tego czujnika jest zadaniem komputera, do którego ten czujnik został podpięty. Należało zatem stworzyć aplikację, która będzie w stanie odebrać informacje z urządzenia, a następnie przetworzyć tę informację w żądany sposób. Powinna ona obsłużyć kilka czujników niezależnie i dać możliwość ich dowolnego rozmieszczenia w robocie. Dlatego też zdecydowałem się w mojej pracy na stworzenie aplikacji wielowątkowej w oparciu o technologię .NET. Wybór został podyktowany prostotą tworzenia aplikacji okienkowych (dla systemu Windows XP), jaką daje środowisko Visual Studio .NET. Aplikacje takie charakteryzują się obsługą zdarzeniową, związaną z jednej strony z interakcją z użytkownikiem, a z drugiej, z reakcją na zdarzenia systemowe. Program został napisany w języku C#, który jest nowoczesnym językiem w pełni obiektowym. Aplikację podzieliłem na trzy logiczne części: prezentacji za pomocą GUI (Windows Form), aplikacji i komunikacji oraz obliczeń matematycznych. Wygląd aplikacji przedstawia rysunek poniżej.



Rys. 20. Wygląd aplikacji

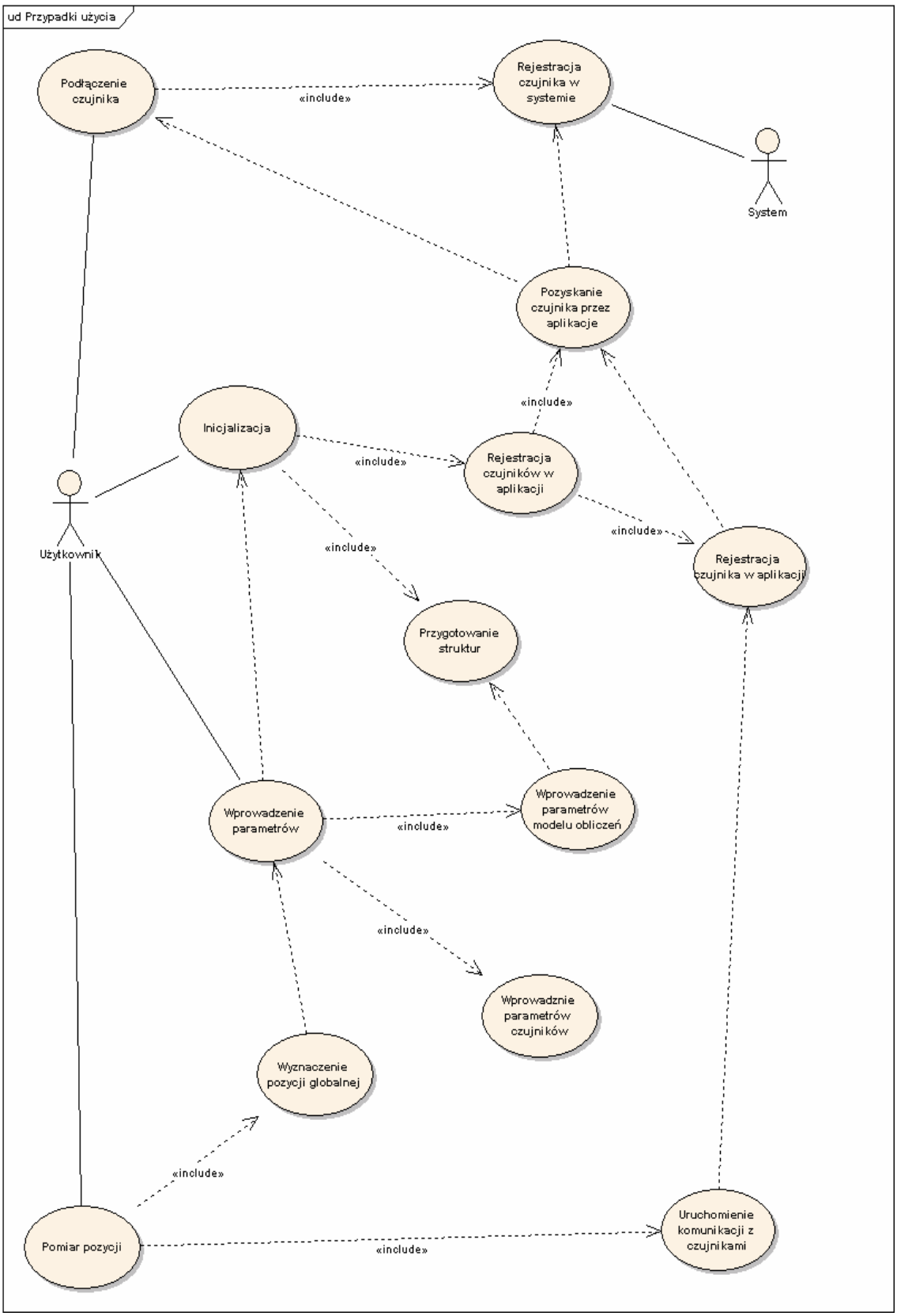


Rys. 21. Podział logiczny aplikacji

Głównymi zadaniami części prezentacji jest interakcja z użytkownikiem oraz wyświetlenie wyników pomiarów i obliczeń. Zdarzenia wywoływane przez użytkownika w GUI są przekazywane i obsługiwane w warstwie logiki biznesowej, a wyniki są prezentowane użytkownikowi, jako rezultat wywołanego zdarzenia zarówno użytkownika jak i robota.

Część obliczeń matematycznych służy do implementacji modelu rozkładu czujników oraz operacji związanych z wyznaczeniem pozycji robota na podstawie pomiarów z czujników. Algorytm obliczeniowy został opisany w poprzednim podrozdziale.

Logikę biznesową zaszytą w aplikacji można przedstawić na poniższym diagramie przypadków użycia.



Rys. 22. Diagram przypadków użycia

Aktorem jest tu użytkownik, który decyduje o inicjalizacji pomiaru, parametrach tego pomiaru oraz uruchomieniu pomiaru. Dodatkowo na diagramie zamieściłem przypadek *Podłączenie czujnika*, który odbywa się poza samą aplikacją, ale ma bezpośredni wpływ na jej działanie. W tym przypadku bierze udział również drugi aktor, którym jest sam system operacyjny. Jego rola ogranicza się do rejestracji podłączonego urządzenia w systemie i przypisania go do odpowiedniej klasy urządzeń HID. Mechanizm ten został już opisany w podrozdziale 5.1 (Projekt czujnika odometrycznego) i nie jest implementowany przeze mnie, ponieważ stanowi integralną część działania systemu operacyjnego (rejestracja urządzeń podłączanych do magistrali USB). Działanie mojej aplikacji sprowadza się do zamodelowania trzech podstawowych przypadków użycia: *Inicjalizacja*, *Wprowadzenie parametrów* oraz *Pomiar pozycji*.

Inicjalizacja

Do uruchomienia tego przypadku służy przycisk *Start*, który rejestruje zdarzenie od użytkownika rozpoczęcia pracy z aplikacją. Wynikiem reakcji na to zdarzenie jest stworzenie odpowiednich struktur do przechowywania danych oraz pozyskanie danych potrzebnych do komunikacji z czujnikami i ich prezentacji w programie. Operacje te można zapisać jako osobne przypadki użycia, które stanowią integralną część przypadku *Inicjalizacja*.

Scenariusz tego przypadku jest sekwencją *Przygotowanie struktur* i *Rejestracji czujników w aplikacji*. Kolejność ta jest istotna, ponieważ proces rejestracji czujników wymaga odpowiednich struktur do przechowywania danych. Wynikiem tego przypadku użycia jest rejestracja czujników w aplikacji, a w części prezentacyjnej pojawienie się odpowiednich nazw czujników pobranych z fizycznych realizacji.

Wprowadzenie parametrów

Przyjęty model obliczeń wymaga podania przez użytkownika parametrów dla czujników oraz pozycji bezwzględnej i orientacji robota. Służą do tego celu odpowiednie pola edycyjne w aplikacji. Za pomocą przycisku *Parametry* dane te są odpowiednio propagowane w aplikacji do odpowiednich struktur. Dla testowych czujników pola odpowiedzialne za kalibracje przyjmują wartości domyślne i są efektem przeprowadzonych badań.

Pomiar pozycji

Przypadek ten realizuje główny proces pomiaru translacji i wyznaczenia pozycji globalnej. Realizowany jest on za pomocą uruchomienia synchronicznej komunikacji z czujnikami i uaktualnianiu pozycji (*Uruchomienie komunikacji z czujnikami*) oraz cyklicznie wyznaczanej pozycji globalnej (*Prezentacja wyników pomiaru*). Podejście takie rozdziela odczyt z fizycznego czujnika od wyznaczania tej pozycji, a buforem jest w tym przypadku struktura czujnika. Komunikacja z czujnikiem odbywa się za pomocą funkcji *Read* biblioteki *mcHID.dll,* a aktualizacja polega na dodaniu translacji do poprzedniej pozycji. Funkcja ta jest synchroniczna i wymaga oczekującego raportu z czujnika; w innym przypadku oczekuje na pojawienie się takiego raportu. Problem ten w systemach wielozadaniowych można rozwiązać poprzez stworzenie wątku, do którego zostanie przypisany ten proces. Wątek jest najmniejszą jednostką wykorzystania procesora przez program i posiada mechanizmy współdzielenia zasobów (takich jak otwarte pliki, obszar danych) [12]. Przełączanie procesora pomiędzy wątkami realizuje sam system operacyjny, który jest odpowiedzialny również za rozwiązywanie problemów współrzędności. Dzięki temu praca samej aplikacji nie jest zakłócana przez oczekiwanie na raport z czujnika. Dodatkowo obiekt czujnika stanowi bufor do przechowywania informacji o położeniu i odczyt jak i zapis może odbywać się niezależnie i nie wpływa na wyznaczanie pozycji globalnej robota. Wyznaczanie samej pozycji globalnej odbywa się cyklicznie na podstawie współrzędnych poszczególnych czujników. Proces te został opisany w podrozdziale 5.2 (Model obliczeń matematycznych). Do jego cyklicznego wywoływania użyłem dedykowanego komponentu .NET – *Timer*. Wykonuje on polecenia zawarte w jego procedurze *OnTimerTick* z odpowiednim okresem powtarzalności zapisanym w atrybucie *Interwal*. Dla mojej aplikacji przyjąłem wartość 100 ms, która jest wystarczająca do prezentacji aktualnej pozycji robota.

Podział aplikacji na części przekłada się również na podział kodu na odpowiednie klasy. Dzięki temu uzyskujemy przejrzystość zarówno logiczną jak i fizyczną. Dodatkowo inne aplikacje mogą skorzystać z tych klas bez potrzeby ponownego implementowania ich zawartości. Istotne jest tylko dostosowanie interfejsu tej aplikacji do wystawionego interfejsu klas.

Klasa *Czujnik*

Klasa *Czujnik* przechowuje współrzędne czujnika oraz dane potrzebne do komunikacji z fizycznym czujnikiem. Dzięki temu fizyczny czujnik dla aplikacji staje się transparentny. Dodatkowo dzięki parametrom *WagaX* i *WagaY* fizyczne wskazania czujnika są odpowiednio skalowane na potrzeby wyznaczania pomiaru. Wartości te są wynikiem badań i testów kalibracji robota i ściśle zależą od użytych czujników odometrycznych.

```
Class Czujnik
    {
        private int Uchwyt;
        private string Nazwa;
        private double WspX;
        private double WspY;
        private double WagaX;
        private double WagaY;

        public int UstawUchwyt(int uchwyt);
        public int DajUchwyt(ref int uchwyt);
        public int UstawNazwe(string nazwa);
        public int DajNazwe(ref string nazwa);
        public int UstawWspolrzednePoczatkowe(double X, double Y,
                                              double dX, double dY,
                                              double wX, double wY);
        public int DajWspolrzedne(ref double X, ref double Y);
        public int UstawWspolrzedne(int X, int Y);
    }
```

Atrybut *Uchwyt* służy do przechowywania wskaźnika do pliku komunikacyjnego z fizycznym czujnikiem. W standardzie WINAPI jest on 32-bitowym adresem fizycznym w pamięci operacyjnej początku pliku w obszarze stosu zajmowanym przez urządzenie fizyczne (czujnik).

Atrybut *Nazwa* jest wykorzystywany do przechowywania ciągu znaków, który jest pobierany w procesie rejestracji czujnika w aplikacji z dedykowanego miejsca w opisie urządzenia. Ciąg ten jest dodatkową informacją, którą uzupełnia się w celu identyfikacji urządzenia w systemie jako bardziej zrozumiałą i intuicyjną dla człowieka.

Atrybuty *WspX* i *WspY* służą do przechowywania współrzędnych czujnika. Stanowią one bufor pomiędzy dwoma procesami: uaktualniania i odczytu. Dzięki temu problem synchronicznego procesu aktualizacji nie wpływa na proces odczytu danych.

Atrybuty *WagaX* i *WagaY* są używane jako mnożniki do przeskalowania wskazań z czujnika do jednostek metrycznych; ze względu na pokonywaną odległość testowego robota przyjąłem centymetr jako jednostkę podstawową.

Procedura *UstawUchwyt* służy do przekazania identyfikatora (wskaźnika) do pliku komunikacyjnego z czujnikiem. Jest ona wywoływana w procesie rejestracji czujnika do zapamiętania uchwytu pozyskanego z systemu.

Metoda *DajUchwyt* jest procedurą odwrotną do *UstawUchwyt* i służy do pobierania identyfikatora pliku komunikacyjnego czujnika. Identyfikator ten jest przekazywany przez wskaźnik do parametru określonego typu. W C# taki parametr jest poprzedzany słowem kluczowym *ref* zarówno w definicji jak i w wywołaniu procedury. Możliwe jest również zwrócenie wartości za pomocą funkcji, jednak ogranicza ono liczbę tak zwracanych elementów. Dodatkowo parametr ten jest często rezerwowany do obsługi wywołania procedury dla zwracania kodu błędu lub poprawności wykonania. Dlatego wszystkie procedury używane w programie są zaimplementowane zgodnie z tą zasadą.

Procedury *UstawNazwe* i *DajNazwe* są używane odpowiednio do wprowadzania i pobierania nazwy przypisanej do czujnika. Nazwa ta w aplikacji służy do prezentacji użytkownikowi przypisania fizycznego czujnika do identyfikatora w systemie.

Metoda *UstawWspolrzednePoczatkowe* jest wywoływana w procesie przekazywania parametrów początkowych dla czujnika. Na podstawie parametrów są wyznaczane współrzędne początkowe dla czujnika w oparciu o współrzędne środka robota (parametry X i Y) i przesunięcia (parametry dX i dY), które informuje o położeniu czujnika względem środka robota. Parametry wX i wY stanowią odpowiednio skalę w osi 0X i 0Y i służą do zamiany wskazań czujnika na przesunięcie w jednostkach metrycznych długości.

Procedura *DajWspolrzedne* zwraca aktualne bezwzględne współrzędne czujnika. Jest wywoływana w procesie wyznaczania pozycji globalnej robota do pobierania aktualnych danych z czujnika.

Metodą odwrotną jest *UstawWspolrzedne*, która służy do aktualizacji współrzędnych w procesie odczytu raportów z czujnika. Parametry wejściowe są typu całkowitego, ponieważ w takiej postaci są uzyskiwane z raportów. Dzięki atrybutom WagaX i WagaY są one odpowiednio skalowane i dodawane do współrzędnych wyznaczonych w poprzednim kroku. Rozwiązanie takie jest bardziej elastyczne i pozwala na łatwą zmianę skali lub użycie myszy o innej rozdzielczości. Wartości wag wynikają bezpośrednio z rozdzielczości czytnika użytego w urządzeniu i mogą być obliczane na podstawie parametrów myszy lub wyznaczane w procesie kalibracji robota.

Klasa *WyznPozycji*

Klasa *WyznPozycji* służy do przechowywania modelu matematycznego do obliczeń oraz operacji związanych z wyznaczaniem pozycji globalnej robota. Przyjęte rozwiązanie i stałą liczbę czujników przekłada się bezpośrednio na atrybuty zaimplementowane w klasie.

```
class WyzPozycji
{
    private double katABdoA0;
    private double dlugoscA0;
    private double katBCdoB0;
    private double dlugoscB0;
    private double katCDdoC0;
    private double dlugoscC0;
    private double katDAdoD0;
    private double dlugoscD0;

    private int DajPozWzgPkt(double x1, double y1,
                             double x2, double y2,
                             double katPoprawka,
                             double dlugoscWekPopr,
                             ref double X, ref double Y);
    private int WybierzPozycje(double Xa, double Ya,
                               double Xb, double Yb,
                               double Xc, double Yc,
                               double Xd, double Yd,
                               ref double X, ref double Y);
    public int UstawParametry(double katAB, double dlA0,
                              double katBC, double dlB0,
                              double katCD, double dlC0,
                              double katDA, double dlD0);

    public int DajPozycjeGlobalna(double Ax, double Ay,
                                  double Bx, double By,
                                  double Cx, double Cy,
                                  double Dx, double Dy,
                                  ref double X, ref double Y);
}
```

Atrybuty *katABdoA0*, *katBCdoB0*, *katCDdoC0* i *katDAdoD0* służą do przechowywania kąta przesunięcia między kierunkiem wyznaczonym przez dwa sprzężone ze sobą czujniki, a wektorem przesunięcia jednego z czujników sprzężonych w stosunku do środka robota. Jest on używany w procesie wyznaczania pozycji globalnej robota. Długość wektora przesunięcia odpowiedniego czujnika względem środka robota jest przechowywana w atrybutach *dlugoscA0*, *dlugoscB0*, *dlugoscC0*, *dlugoscD0*.

Procedura *DajPozycjeGlobalna* służy do wyznaczania pozycji globalnej robota na podstawie współrzędnych czujników. Jest ona wywoływana w procesie określenia bezwzględnej pozycji robota. Działanie jej opiera się w oparciu o procedury *DajPozWzgPkt* i *WybierzPozycje*. Metoda *DajPozWzgPkt* oblicza pozycję globalną na podstawie współrzędnych dwóch czujników sprzężonych. Wyznaczone w ten sposób

rozwiązania cząstkowe są przekazywane do procedury *WybierzPozycje*, która służy do wskazania rozwiązania najbardziej prawdopodobnego według algorytmu opisanego w podrozdziale 5.2.

Procedura *UstawParametry* jest wykorzystywana w procesie przekazywania parametrów początkowych do modelu obliczeń, na podstawie danych wprowadzonych w GUI przez użytkownika.

Klasa *MouseReader*

Klasa *MouseReader* stanowi główną klasę aplikacji, ponieważ skupia w sobie logikę biznesową oraz wiąże część komunikacji z czujnikami i część obliczeniową. Wystawia ona niezbędne metody do obsługi całego procesu wyznaczania pozycji globalnej robota na podstawie dedykowanych czujników odometrycznych.

```
class MouseReader
 {
     private static Czujnik[] CzujnikiLista = new Czujnik[8];
     private static int CzujnikiLiczba;
     private Thread[] WatkiLista = new Thread[8];
     private WyzPozycji kalkulator = new WyzPozycji();

     public int MouseReaderInit(int hwnd)
     public int MouseReaderExit()
     public int ZmianaParametrow(double X, double Y,
                                 double Xa, double Ya,
                                 double wXa, double wYa,
                                 double katA,
                                 double Xb, double Yb,
                                 double wXb, double wYb,
                                 double katB,
                                 double Xc, double Yc,
                                 double wXc, double wYc,
                                 double katC,
                                 double Xd, double Yd,
                                 double wXd, double wYd,
                                 double katD)

     public int DajIloscCzujnikow()
     public int DajNazweCzujnika(int NrCzujnika, ref string Nazwa)
     public int DajWspolrzedneCzujnika(int NrCzujnika,
                                       ref double X, ref double Y)
     public int PomiarStart()
     public int PomiarStop()
     public int DajPozycjeGlobalna(ref double X, ref double Y)
     public int DajStatusCzujnika(int NrCzujnika,
                                  ref string Status)
     private static int AktualizujPozycje(int NrCzujnika)
     public static void Czujnik1Aktualizacja()
     public static void Czujnik2Aktualizacja()
     public static void Czujnik3Aktualizacja()
     public static void Czujnik4Aktualizacja()
     public static extern int Connect(int pHostWin);
```

```
        public static extern int Disconnect();
        public static extern int GetItem(int pIndex);
        public static extern int GetItemCount();
        public static extern int Read(int pHandle,
                                      ref byte pData);
        public static extern int GetVendorID(int pHandle);
        public static extern int GetProductID(int pHandle);
        public static extern int GetVendorName(int pHandle,
                                               string pText,
                                               int pLen);
        public static extern int GetProductName(int pHandle,
                                                ref byte pText,
                                                int pLen);
    }
```

Atrybut *CzujnikiLista* jest tablicą do przechowywania obiektów klasy *Czujnik*. W mojej pracy wykorzystuję tylko cztery czujniki, dlatego też rozmiar tej struktury nie jest parametryzowany. Przy wykorzystaniu innej liczby czujników należy go zmienić. Procedura uzupełniająca tę tablicę jest dynamiczna i uwzględnia ten rozmiar. W polu *CzujnikiLiczba* przechowywana jest rzeczywista liczba zarejestrowanych czujników w aplikacji. Z każdym fizycznym czujnikiem jest skojarzony wątek. Analogicznie są one przechowywane w tablicy wątków *WatkiLista*, której rozmiar jest uzależniony od ilości czujników. Użycie tablic zostało podyktowane prostotą i szybkością działania w stosunku do struktur dynamicznych oraz ich stałym rozmiarem. Dostęp do elementów jest indeksowany od 1 tak, aby by on intuicyjny przy implementacji GUI. Dodatkowym atrybutem jest klasa do obliczeń matematycznych i wyznaczania pozycji - *kalkulator*. Model ten jest przystosowany dla czterech czujników, ale może być zastąpiony przez klasę bardziej generyczną o dynamicznej strukturze ustalanej w momencie inicjalizacji. Klasa ta musi jedynie wystawić odpowiedni interfejs do współpracy z pozostałymi klasami aplikacji.

Procedura *MouseReaderInit* jest używana w procesie rejestracji czujników do ich pozyskiwania z systemu i mapowania w aplikacji za pomocą dedykowanej klasy *Czujnik*. Parametrem tej metody jest wskaźnik do systemu, w jakim pracuje aplikacja. Jest on niezbędny do podłączenia się do klasy HID w celu pozyskania odpowiednich danych (adresu, identyfikatora pliku komunikacji). Wykorzystuje ona funkcje biblioteki *mcHID.dll – Connect, GetItem, GetItemCount, GetVendorID, GetProductID, GetProductName*. Funkcja *Connect* służy do podłączenia do klasy HID w systemie i zakończenie jej wywołania sukcesem daje możliwość wywołania kolejnych funkcji pozyskania urządzeń z systemu. *GetItemCount* zwraca ilość urządzeń podpiętych

aktualnie do systemu i zarejestrowanych w klasie HID. W mojej pracy wykorzystuję tę wartość do sterowania pętlą pozyskiwania adresów fizycznych czujników w systemie. Pozyskiwanie tych wskaźników jest realizowane za pomocą *GetItem*. W celu wybrania odpowiedniego urządzenia każdy fizyczny czujnik odometryczny ma ustawione odpowiednie wartości w polach informacyjnych *VendorID* i *ProductID*. Na podstawie tych danych aplikacja rejestruje urządzenie jako czujnik, zaś samo pozyskanie tych wartości odbywa się za pomocą funkcji *GetVendorID, GetProductID*. Dodatkowo przy użyciu funkcji *GetProductName* pobieram nazwę z urządzenia w celu późniejszej prezentacji użytkownikowi aplikacji.

Metoda *MouseReaderExit* jest używana w procesie ponownego rejestrowania czujników w programie lub przy wyjściu z niego. Użycie tej funkcji jest podyktowane potrzebą wyczyszczenia odpowiednich struktur do wymiany danych z klasą HID oraz dobrą praktyką zwalniania nieużywanej pamięci operacyjnej w systemie procesorowym. Procedura ta używa funkcji *Disconnect* z biblioteki *mcHID.dll*.

Metoda *DajIloscCzujnikow* służy do zwracania informacji o liczbie czujników zarejestrowanych w aplikacji. Może być ona wykorzystywana do sterowania w sekcjach warunkowych kodu części prezentacyjnej. Procedura *DajNazweCzujnika* jest wykorzystywana do pozyskania nazwy czujnika na podstawie identyfikatora w programie.

Procedura *ZmianaParametrow* jest używana do przekazania warunków początkowych zarówno dla czujników jak i modelu do wyznaczania pozycji globalnej robota. Wywołanie tej procedury jest bezpośrednią reakcją na zdarzenie użytkownika wynikające z wprowadzenia parametrów zerowych w GUI.

Uruchomienie procesu związanego z pomiarem translacji odbywa się przez wywołanie procedury *PomiarStart*. Uruchamia ona poszczególne wątki i przypisuje im odpowiednie metody: *Czujnik1Aktualizacja*, *Czujnik2Aktualizacja*, *Czujnik3Aktualizacja*, *Czujnik4Aktualizacja*. Aby procedurę można przypisać do wątku nie może ona mieć parametrów wejściowych. Metody te służą jedynie do wywołania funkcji *AktualizujPozycje* z odpowiednim parametrem – numerem czujnika w aplikacji. Natomiast procedura ta synchronicznie pobiera raport z fizycznego czujnika funkcją *Read* i wywołuje procedurę uaktualniania pozycji bezwzględnej czujnika. Procedura *PomiarStop* służy do zatrzymywania procesu aktualizowania pozycji poszczególnych czujników.

Do wyznaczania pozycji globalnej robota należy wywołać funkcję *DajPozycjeGlobalna*. Jest ona wywoływana cyklicznie przez część prezentacji. Dodatkowo procedura *DajStatusCzujnika* zwraca aktualny status dla zadanego czujnika. Informacja ta jest wykorzystywana w procesie zatrzymywania pomiaru do informowania użytkownika o potrzebie odwieszenia danego wątku za pomocą wystawienia sztucznego raportu z czujnika odometrycznego. Obie te metody są uruchamiane w GUI za pomocą dedykowanego komponentu do okresowych wywołań – *Timer*.

6. Wykonanie testów

6.1. Kalibracja czujników odometrycznych

Wykorzystanie czujników odometrycznych w robocie wymaga określenia odpowiednich współczynników skalujących wskazania czujników na metryczne przesunięcie, które interpretują algorytmy sterujące robota. Proces taki nazywamy kalibracją i polega on na uśrednieniu wyników z kilku pomiarów odległości i wskazań czujnika. Pomiar ten można wykonać na klika sposobów:

- zmierzyć odległość, jaką pokonał robot przy określonym wskazaniu czujnika
- odczytać wskazanie czujnika po przebyciu określonej odległości przez robota
- dokonać obu pomiarów niezależnie (długość drogi i wskazania czujnika)

Na podstawie tych dwóch danych można wyznaczyć skalę jako stosunek odległości do wskazań czujnika. Wartość ta jest używana w procesie obliczeń matematycznych do aktualizowania współrzędnych czujnika. Może być również obliczona na podstawie rozdzielczości myszy podanej przez producenta. Skala ta wpływa bezpośrednio na dokładność wyznaczania pozycji globalnej, ponieważ proces wyznaczania tej pozycji kumuluje błędy pojedynczych pomiarów położenia czujników. Przyjęcie większej precyzji nie usuwa tego błędu, ale w znacznym stopniu go redukuje. Kalibracje można wykonywać w dwóch kierunkach jednocześnie. Należy jednak pamiętać, że droga przebyta przez robota uwzględniana w obliczeniach jest rzutem drogi rzeczywistej na odpowiedni kierunek. Czujniki odometryczne użyte w pracy wyznaczają przesunięcie w dwóch prostopadłych kierunkach, które do wyznaczenia pozycji robota w danym środowisku jest wystarczające. Skala jest określana dla każdego kierunku. Wartość tej skali nie zależy od rozkładu czujników względem robota i raz wyznaczona nie zmienia się. Aplikacja MouseReader wymaga tej skali jako parametr początkowy dla czujnika. W mojej pracy przeprowadziłem kalibracje mierząc drogę i wskazania czujników. W tabeli poniżej zostały zebrane dane reprezentacyjne serii.

Tabela 2. Dane z kalibracji czujników

Lp.	Długość	Wskazanie	Skala
współrzędna y			
1	10 cm	702	0,0142
2	10 cm	708	0,0141
3	10 cm	714	0,0140
4	10 cm	706	0,0141
5	10 cm	710	0,0141
współrzędna x			
6	14 cm	1173	0,0119
7	14 cm	1170	0,0119
8	14 cm	1166	0,0120
9	14 cm	1164	0,0120
10	14 cm	1161	0,0120

6.2. Rozkłady czujników

Jednym z zagadnień w mojej pracy jest zbadanie wpływu rozmieszczenia czujników na dokładność pomiarów. Badania zostały wykonane w oparciu o przyjęty model obliczeń matematycznych, dla następujących rozkładów czujników:

- czujniki ułożone w linii środka robota
- czujniki rozmieszczone w wierzchołkach trójkąta i boku
- czujniki rozmieszczone w wierzchołkach trapezu
- czujniki rozmieszczone w wierzchołkach deltoidu

Rozkłady te uwzględniają wszystkie cztery czujniki. Należy jednak spełnić warunek dla wyznaczania pozycji o nie pokrywaniu się współrzędnych czujników ze środkiem robota. Wynika to z przyjętego modelu obliczeń, który bazuje na wektorach, ponieważ dla dwóch punktów pokrywających się nie można jednoznacznie określić wektora (dwa punkty pokrywające się wyznaczają rodzinę wektorów). Ze względu na rozmiar samych czujników nie da się ich rozłożyć w tych samych punktach, zatem należy spełnić warunek o nie pokrywaniu z środkiem robota. Aplikacja do odczytu danych z czujników nie kontroluje tych ograniczeń na pokrywanie, dlatego wyniki wyznaczania pozycji globalnej mogą być przypadkowe i niewiarygodne. Warunek ten nie jest trudny

do sprawdzenia, ale ze względu na możliwości fizycznego rozkładu czujników względem robota postanowiłem go nie implementować.

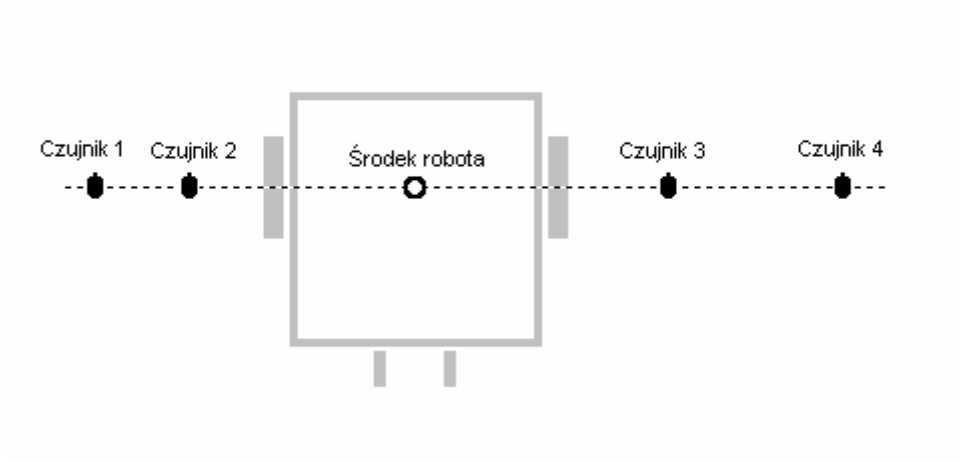
W mojej pracy użyłem następujących rozkładów w oparciu o figury geometryczne:

- linia
- trójkąt
- pseudotrójkąt
- prostokąt
- trapez
- deltoid

Położenie środka robota mobilnego nie zależy od przyjętego rozkładu i może być dowolnie określone. Rozwiązanie takie pozwala na dużą elastyczność w doborze tego rozkładu oraz położenia czujników względem robota.

Linia

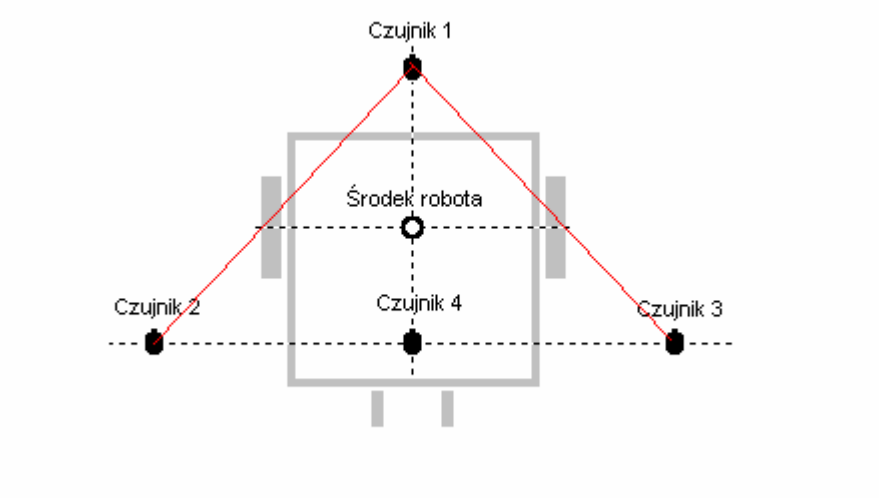
Czujniki w tej konfiguracji umieszczone są na linii środka robota prostopadłej do kierunku poruszania się robota. Środek robota stanowi jeden z punktów na tej linii. Położnie czujników w tym rozkładzie może być zarówno równomierne, jaki i dowolne w stosunku do środka linii. W badaniach przyjąłem rozkład dowolny, ponieważ rozkład równomierny jest przypadkiem takiego rozkładu. Dodatkowo położenie czujnika w rozkładzie daje kilka wariantów rozłożenia czujników. Te stopnie swobody nie mają jednak wpływu na sam pomiar i wyznaczanie pozycji globalnej, pod warunkiem prawidłowo zdefiniowanych i wprowadzonych parametrów początkowych. Rysunek poniżej przedstawia testowany wariant rozkładu czujników w linii.



Rys. 23. Rozkład czujników w linii.

Trójkąt

Rozkład czujników w tej konfiguracji przedstawia rysunek poniżej. Czujniki są umieszczone w wierzchołkach trójkąta, a dodatkowy czujnik jest umieszczony w jednym z jego boków.

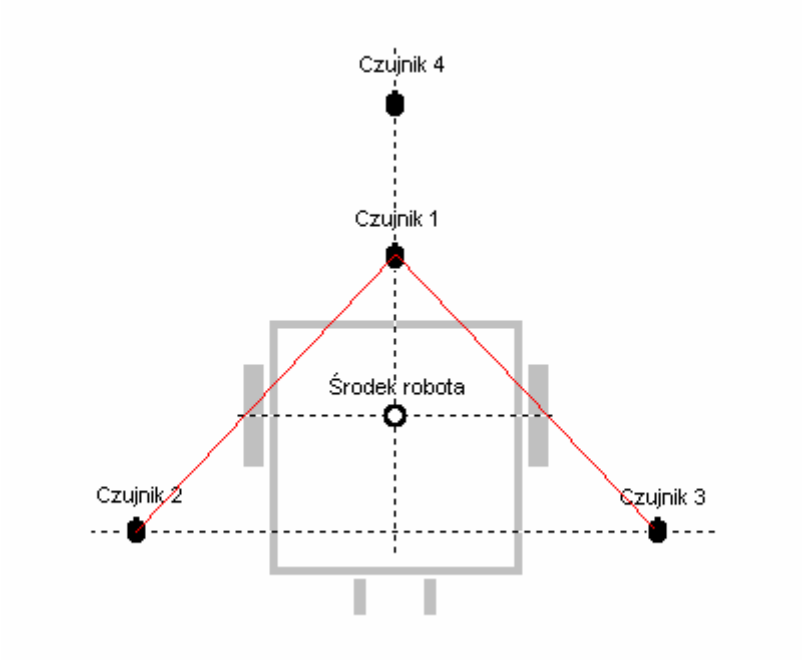


Rys. 24. Rozkład czujników w trójkąt.

Kolejność czujników przedstawiona na rysunku jest jednym z możliwych ich rozmieszczeń. Poza tym położenie czwartego czujnika nie jest deterministyczne i może być zmienione w ramach jednego boku oraz pomiędzy nimi. Rodzaj trójkąta jest również dowolny. W badaniu przyjąłem trójkąt ostrokątny o różnych długościach boków.

Pseudotrójkąt

Rozkład ten jest analogiczny do rozkładu trójkąta ze zmianą położenia czwartego czujnika. Znajduje się on na kierunku poruszania się robota w dowolnej odległości. Rozkład ten przedstawia poniższy rysunek.

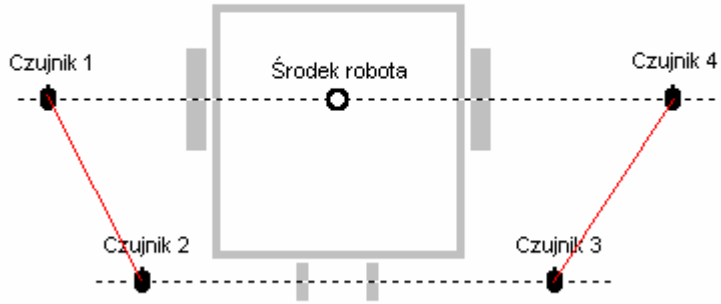


Rys. 25. Rozkład czujników w pseudotrójkąt.

Położenie czujnika 4 w tym rozkładzie czyni wskazanie czujnika 1 bardziej wiarygodnym. Możliwe jest również umiejscowienie czujnika 4 na przedłużeniu innego wierzchołka, ale wydaje się to nie zasadne i nie optymalne, ponieważ nie wzmacnia wskazań czujników w tych wierzchołkach. Rozkład taki nie był badany w ramach pracy, jako niepoprawiający wyznaczanie pozycji globalnej robota.

Prostokąt i trapez

Rozkład prostokąt jest szczególnym przypadkiem rozkładu trapez, dlatego zostanie on opisany w tym kontekście. Czujniki rozmieszczone są w czterech wierzchołkach trapezu; przedstawia to rysunek poniżej.

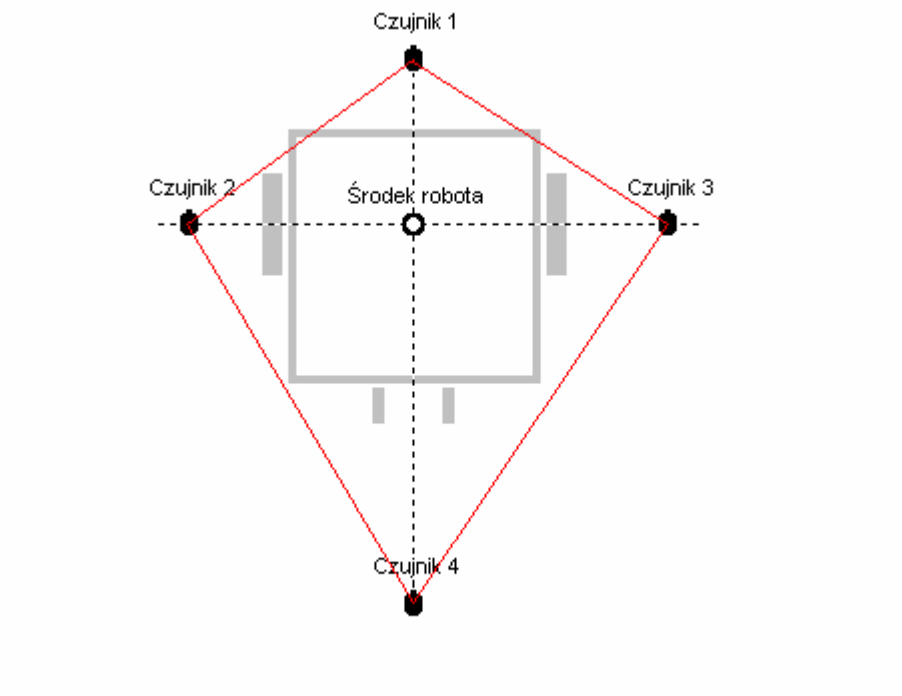


Rys. 26. Rozkład czujników w trapezie (prostokącie).

Rozkład ten wydaje się najbardziej optymalny dla robotów mobilnych poruszających się za pomocą dwóch kół napędowych oraz koła (kół) skrętnych przy użyciu czujników odometrycznych wskazujących translacje w dwóch kierunkach. Krótsza podstawa powinna być umieszczana bliżej kół skrętnych, ze względu na lepszą rejestracje obrotów robota, ponieważ przy krótszym promieniu skrętu robota zmiany położenia czujników umieszczonych na dłuższej podstawie są wskazywane przy użyciu większej wartości wskazania. Przekłada się to bezpośrednio na mniejszy błąd pomiaru czujnika i w konsekwencji na mniejszy błąd globalny wyznaczania położenia.

Deltoid

Deltoid jest figurą geometryczną, składającą się z dwóch par przylegających boków o równych długościach. Czujniki w tym rozkładzie umieszczone są w wierzchołkach, co przedstawia poniższy rysunek. Rozkład ten wydaje się optymalny do robotów mobilnych z dwoma kołami napędowymi i kołem (kołami) skrętnym. Położenie czujnika 1 powinno znajdować się bliżej kół skrętnych, ponieważ przy małym promieniu skrętu robota czujnik 4 powinien wskazywać większą wartość translacji. Dzięki temu zmniejsza się błąd pomiaru i wyznaczania położenia.



Rys. 27. Rozkład czujników w deltoidzie.

Generalnie rozkład czterech czujników może być dowolny w stosunku do robota i jego środka. W mojej pracy przyjąłem rozkłady pokrywające się z wybranymi figurami geometrycznymi. Dzięki temu nazwa rozkładu kojarzy się z odpowiednim rozkładem fizycznym i porównywanie wyników wyznaczania tej pozycji jest łatwiejsze. Testy zostały wykonane w oparciu o symulacje ruchu robota przy użyciu odpowiedniej konstrukcji pozwalającej na swobodny rozkład czujników w przestrzeni. Rozwiązanie staje się przez to uniwersalne, niezależne od fizycznej realizacji robota mobilnego. Wadą zaś takiego badania jest brak możliwości odpowiedniego przetestowania odporności systemu wspomagającego na wystąpienie zjawiska poślizgu.

6.3. Scenariusze testowe

Do zbadania wpływu rozkładu czujników na wyznaczanie pozycji globalnej robota stworzyłem scenariusze testowe, ponieważ pojedyncze próby musiały być wykonane w tych samych warunkach dla każdego z rozkładów. Podstawą scenariusza testowego była wyznaczona trasa, jaką musiał pokonać robot wyposażony w odpowiedni zestaw czujników. W badaniach przyjąłem również podział na scenariusze proste i złożone. Scenariusze proste dotyczyły wykonania pomiarów podczas wykonywania pojedynczego manewru. Natomiast scenariusze złożone składały się z scenariuszy prostych, następujących po sobie w założonej sekwencji. Scenariusze proste

obejmowały ruch w zadanym kierunku, zarówno w jedną jak i w drugą stronę. Czujniki wykorzystane w testach dokonują pomiaru translacji w dwóch kierunkach prostopadłych równocześnie, dlatego też badania z nimi obejmowały pomiary przesunięcia prostopadłego do kierunku jazdy robota. W rzeczywistości robot wydziałowy *Pionier 3* nie jest w stanie wykonać takiego przemieszczenia, ponieważ koła napędowe nie są skrętne, ale może nastąpić poślizg poprzeczny, który zmieni trasę robota. Scenariusz złożony dla przesunięcia obejmował również przesunięcie w dwóch kierunkach jako złożenie dwóch ruchów prostych (w jednym kierunku) oraz ruchu w dwóch kierunkach jednocześnie. Scenariusze dla obrotów obejmowały obrót w miejscu względem środka robota oraz względem jednego ze skrajnych czujników. Obejmuje to dwa przypadki obrotu robota: obrót w miejscu i jazda po łuku.

6.4. Wpływ optyki na pomiary

W oparciu o teorię optyki można wyznaczyć parametry układu optycznego: ogniskowa, głębia ostrości. Ogniskowa decyduje o fizycznym rozmieszczeniu soczewki względem czujnika. Natomiast głębia ostrości reguluje zakres odległości obiektów od czujnika optycznego, w którym są one widzialne z tą samą ostrością. Parametr ten określa granicę nierówności powierzchni dla danego układu optycznego, po przekroczeniu której czujnik nie jest w stanie właściwie zmierzyć translacje. Myszy optyczne użyte w pracy posiadają głębie ostrości dającą zakres 2mm, co dla wykorzystania jako urządzenia wskazujące w komputerze jest wystarczające, ale może nie być dla robota poruszającego się po powietrzni z większymi zagłębieniami. Rozwiązaniem tego problemu jest większa głębia ostrości, jednak wiąże się to z mniejszym światłem, jakie dociera do czujnika ze względu na zmniejszanie się otworu względnego układu optycznego. W związku z tym należy dostosować oświetlenie obserwowanej powierzchni. Problem ten można rozwiązać zmieniając układ optyczny, pozwalający na lepsze oświetlenie czujnika.

7. Wyniki testów

Testy zostały wykonane na parkiecie, gdzie nierówności powierzchni były na poziomie 2 mm, a faktura drewna miała charakterystyczne linie słoi. Ze względu na podłączenie czujników do komputera stacjonarnego i użytej długości kabla [5] (testy wykazały, że przedłużenie kabla o 3 m uniemożliwiało wykrycie czujnika w systemie) badania zostały wykonane na ograniczonej powierzchni. Testy wykonane zostały w oparciu o rozkłady czujników opisanych powyżej. Rozmieszczenie fizyczne czujników w poszczególnych rozkładach zostały zebrane w poniższej tabeli, gdzie środek robota znajduje się w punkcie (0,0).

Tabela 4. Rozmieszczenie czujników względem środka robota

Rozkład	Czujnik 1		Czujnik 2		Czujnik 3		Czujnik 4	
	punkt	kąt wg 2	punkt	kąt wg 3	punkt	kąt wg 4	punkt	kąt wg 1
Linia	(-30,0)	0	(-7;0)	0	(7;0)	180	(29;0)	0
Trójkąt	(0;10)	38,6	(-12;-5)	22,6	(12;-5)	-22,6	(0;-5)	0
Pseudotrójkąt	(0;10)	38,6	(-12;-5)	22,6	(12;-5)	65,4	(0;15)	0
Trapez	(-20;0)	45	(-10;-10)	45	(10;-10)	90	(20;0)	0
Deltoid	(0;15)	33,7	(-10;0)	0	(10;0)	-26,5	(0;-5)	0

Badania zostały przeprowadzone zgodnie z opisanymi poniżej scenariuszami. W ramach każdego scenariusza testy były powtarzane 30 krotnie, a wyniki uśrednione i skrajne zostały zebrane w tabelach.

Scenariusz 1 - Poruszanie się po linii prostej w kierunku jazdy robota

W tym scenariuszu od robota mobilnego wymaga się jazdy zarówno w przód jak i w tył po drodze o ściśle określonej długości. Na podstawie pomiarów z czujników i wyznaczonej pozycji globalnej ocenia się skuteczność metody jako błąd względny w stosunku do pokonanej odległości rzeczywistej. Odmianą tego scenariusza może być ruch składający się z przesunięcia w przód i w tył o tą samą odległość, a ocenie podlega różnica pomiędzy rzeczywistą pozycją globalną a pozycja wskazaną przez czujniki. Podczas badania został zasymulowany ruch robota za pomocą przesunięcia czujników

[5] długość kabla połączeniowego do portu USB wpływa na szybkość działania i utratę danych. Maksymalna długość podana w specyfikacji magistrali wynosi 5 m, ale zależy to od pobieranej mocy układu.

na odległość 50 cm. Wskazania w kierunku prostopadłym nie zostały uwzględnione. Służyły one do porównania jakości próby, tzn. odchylenia od zadanego kierunku (w przeprowadzonych badaniach metody wskazywały przesunięcie na poziomie 2 mm, co daje wydłużenie drogi o $4*10^{-4}$ cm i zostało pominięte). Poniższa tabela zawiera wyniki wskazań czujników.

Tabela 4. Wyniki pomiarów dla scenariusza 1 (odległość 50 cm)

Rozkład	**Wynik** [cm]			**Błąd względny** [%]	
	min	**max**	**śred**	**max**	**śred**
Linia	48,2	51,6	49,2	3,6	1,6
Trójkąt	48,5	51,1	49,6	3,0	0,8
Pseudotrójkąt	48,4	51,4	49,5	3,2	1,0
Trapez	48,4	51,2	49,5	3,2	1,0
Deltoid	48,6	51,0	49,6	2,8	0,8

Scenariusz 2 – Ruch czujników prostopadły do kierunku jazdy robota

Scenariusz ten został dodany w celu przetestowania ruch czujników prostopadłego do kierunku jazdy robota. Robot wydziałowy nie jest w stanie wykonać takiego ruchu samodzielnie, ale może taki ruch wystąpić podczas jazdy (dryft boczny) lub w wyniku działania siły bocznej (poślizg poprzeczny). W tym scenariuszu wykonuje się ruch czujnikami w kierunku poprzecznym do kierunku, w jakim poruszałby się robot zarówno w jedną jak i w drugą stronę na ustaloną odległość. Ocenie podlega pomiar w wyznaczonym kierunku oraz określany jest błąd względny pomiaru. Modyfikacja tego scenariusza polegająca na wykonaniu ruchu w obie strony pozwoli na zbadanie różnicy pomiędzy rzeczywistym położeniem, a położeniem wskazanym przez czujniki. Badania te nie uwzględniają przesunięcia w kierunku jazdy robota. Wskazania te zostały uwzględnione do porównania jakościowego przesunięcia. Podobnie jak w scenariuszu 1 przesunięcie to wprowadza błąd marginalny i został on pominięty.

Tabela 5. Uśrednione wyniki pomiarów dla scenariusza 2 (odległość 50 cm)

Rozkład	**Wynik** [cm]			**Błąd względny** [%]	
	min	**max**	**śred**	**max**	**śred**
Linia	48,1	51,8	49,0	3,8	2,0
Trójkąt	48,4	51,3	49,2	3,2	1,7
Pseudotrójkąt	48,3	51,5	49,0	3,3	2,0
Trapez	48,3	51,3	49,1	3,3	1,8
Deltoid	48,5	51,2	49,2	3,1	1,7

Scenariusz 3 – Obrót względem środka robota o ustalony kąt

Obrót jest jednym z podstawowych manewrów robota mobilnego. Od precyzji wykonania tego obrotu zależy kierunek jazdy i wyznaczenie pozycji globalnej robota. Scenariuszu ten polega na wykonaniu obrotu wokół środka robota o zadany kąt i wyznaczenie błędu względnego pomiaru.

Tabela 6. Uśrednione wyniki pomiarów dla scenariusza 3 (kąt 90 °)

Rozkład	**Wynik** [°]			**Błąd względny** [%]	
	min	**max**	**śred**	**max**	**śred**
Linia	84,8	93,8	87,3	5,8	3,0
Trójkąt	85,3	93,3	87,7	5,2	2,6
Pseudotrójkąt	85,1	93,4	87,6	5,4	2,7
Trapez	85,4	93,2	87,7	5,1	2,6
Deltoid	85,8	92,9	87,9	4,6	2,3

Scenariusz 4 – Obrót względem czujnika o ustalony kąt

Scenariusz ten jest odmianą scenariusza 3. Oś obrotu jest umieszczona w czujniku. Przypadek ten jest odpowiednikiem obrotu robota, gdzie koła nie obracają się z tą samą prędkością. Badanie polega na obrocie robota o zadany kąt. Ocenie podlega wskazanie kąta obrotu i obliczenie błędu względnego wskazania z czujników do zadanego kąta.

Tabela 7. Uśrednione wyniki pomiarów dla scenariusza 4 (kąt 90 °)

Rozkład	Wynik [°]			Błąd względny [%]	
	min	max	śred	max	śred
Linia	85,1	93,5	87,8	5,4	2,4
Trójkąt	85,5	93,0	88,0	5,0	2,2
Pseudotrójkąt	85,3	93,1	87,9	5,2	2,3
Trapez	85,6	92,9	88,1	4,8	2,1
Deltoid	85,9	92,8	88,3	4,5	1,9

Scenariusz 5 – Jazda złożona

Scenariusz ten może zawiera dowolny zestaw poprzednich scenariuszy. W mojej pracy ograniczyłem się do połączenia scenariusza 1 z 3, czyli jazda w jednym kierunku z obrotem o ustalony kąt. Ocenie podlega różnica pomiędzy punktem końcowym a wyznaczonym za pomocą wskaźników z czujnika. Błąd jest wyznaczony jako błąd bezwzględny odległości wyznaczonego punktu od punktu, który miał być osiągnięty. Jest to promień wyznaczający obszar w wokół celu, w którym znajduje się robot po jego osiągnięciu.

Tabela 8. Uśrednione wyniki pomiarów dla scenariusza 5 (50 cm,90 °,50 cm)

Rozkład	Wynik [punkt]			Błąd bezwzględny [cm]		
	min	max	śred	min	max	śred
Linia	(-49,3;50,4)	(-48,4;51,9)	(-49,2;50,7)	0,8	2,5	1,1
Trójkąt	(-49,6;50,3)	(-48,5;51,4)	(-49,4;50,6)	0,5	2,1	0,8
Pseudotrójkąt	(-49,5;50,4)	(-48,3;51,4)	(-49,3;50,5)	0,6	2,2	0,9
Trapez	(-49,7;50,3)	(-48,6;51,5)	(-49,5;50,5)	0,4	2,1	0,7
Deltoid	(-49,7;50,2)	(-48,8;51,3)	(-49,6;50,3)	0,4	1,8	0,5

8. Wnioski i podsumowanie

Wyznaczenie pozycji globalnej robota mobilnego jest operacją decydującą dla sterownia robota. Algorytmy decyzyjne i sterujące robota wykorzystują ta pozycje do określenia położenia na zadanej mapie, która przechowuje informacje o zadaniach do wykonania. Precyzja wyznaczenia tego położenia określa przydatność danej aplikacji robota mobilnego do powierzonego zadania. Dlatego też roboty wyposaża się w kilka metod określania położenia redukując błędy pomiarowe poszczególnych metod. Metody te wykorzystują różne czujniki od urządzeń mechanicznych, poprzez sensory wysyłające i odbierające sygnały, do zaawansowanych kamer z układami optycznymi. Informacje zbierane przez czujniki przekazywane są w postaci surowej i muszą być przekształcone na odpowiednie dane dla algorytmów decyzyjnych. Metody przytoczone w pracy zawierają wszystkie te elementy potrzebne do wyznaczenia pozycji globalnej robota mobilnego. Pożądane jest użycie kilku metod, które są odporne na różne zjawiska tak, aby przy zaistnieniu jednego z nich pozostałe metody dały poprawne wyniki. Dodatkowo stosuje się rachunek błędów, który redukuje błąd globalny wyznaczania pozycji.

Czujnik odometryczny stworzony i przetestowany w pracy okazał się być wystarczający do zastosowania w badaniach z robotem wydziałowym, ze względu na szybkość działania. Precyzja czujnika jest uzależniona od precyzji użytej myszy optycznej, a rozwój technologii sprawił, że na rynku są już dostępne urządzenia wskazujące o większej precyzji. Decydującym elementem układu czujnika jest zastosowany układ optyczny. Rozbudowa jego pozwoli na lepsze wykorzystanie samego czujnika. Chodzi tu głównie o powierzchnie nieregularne, których poziom nierówności jest większy o zastosowanej głębi ostrości układu optycznego. Dla tych powierzchni wskazania czujnika są przypadkowe i trudne do wykrycia. W środowisku naturalnym zdarzają się również takie zdarzenia losowe, które powodują przesuwanie się „podłoża” (dla czujnika optycznego). Przykładem takiego zjawiska może być burza piaskowa. Wskazania czujnika są przez to zafałszowane, a błędny pomiar jest trudny do wykrycia. Dlatego należy stosować metody, które aproksymują pozycje globalną i są odporne na przypadkowe błędy pojedynczych czujników. Algorytmy użyte w pracy do wyznaczania pozycji globalnej robota mobilnego na podstawie współrzędnych czujników okazały się

wystarczające do zastosowania w robocie laboratoryjnym. Błąd względny wyznaczony w badaniach był na poziomie 0,5 % dla ruchu prostoliniowego i 1,2 % dla obrotu. Dla rozkładu w trapez i deltoid algorytm FCR okazał się być lepszy od zaproponowanego algorytmu wektorowego. Różnice we wskazaniach były rzędu 10^{-1} % błędu względnego. Natomiast dla rozkładu w linię algorytm ten okazał się być gorszy zarówno dla ruchu prostoliniowego jak i obrotowego.

Jednym z celów pracy było zbadanie wpływu rozmieszczenia czujników na dokładność pomiarów. Po przeprowadzeniu zaproponowanych w pracy scenariuszy okazało się, że najlepszym rozkładem dla robota wydziałowego jest rozkład w trapez i deltoid. Dla zmniejszenia błędów powstających przy obrocie słuszne jest umieszczanie czujników na dłuższej podstawie (dla trapezu) przy osi kół napędowych oraz umieszczenie wierzchołka (dla deltoidu) przy parze dłuższych boków z przodu robota przed kołami napędowymi. Jest to bezpośrednio związane z łukiem, po jakim porusza się czujnik. Jeżeli ramię będzie dłuższe to dla danego kąta długość łuku będzie większa. Zatem czujnik powinien wskazać większą wartość, a w przypadku błędu na poziomie dokładności czujnika (nie zliczenie jednego kroku) błąd całkowity pomiaru będzie mniejszy.

Przeprowadzone scenariusze nie uwzględniały powierzchni nieregularny i nierównych, ponieważ optyka dedykowana dla myszy optyczny nie jest wystarczająca do ich wykonania na tych powierzchniach. W pracy umieściłem opis teoretycznego układu optycznego, w który można wyposażyć czujniki odometryczny, aby był bardziej uniwersalny i w określonym stopniu odporny na błędy wynikające z rodzaju powierzchni.

System wspomagania wyznaczania pozycji globalnej robota stworzony prze ze mnie i opisany w pracy może być potraktowany jako metoda odometryczna dla nawigacji robotów mobilnych. System ten jest autonomiczny i może być użyty w dowolnej aplikacji robota, a integracja powinna nastąpić na poziomie algorytmów decyzyjnych. Klasy użyte w aplikacji wystawiają odpowiedni interfejs do pobierania wyznaczonej pozycji globalnej na podstawie użytych czujników. Mowa tu o procedurze *DajPozycjeGlobalna*, która jest przeciążona i w zależności od ilości parametrów pozycja wyznaczana jest w oparciu o różne algorytmy.

9. Literatura

[1] Agilent Technologies
„Agilent ADNS – 2610. Optical Mouse Sensor. Data Sheet”

[2] Allman S. (Cypress Semiconductor)
“*Using the HID class eases the job of writing USB device drivers*”
Electronics Design News 19.09.2002r

[3] Borenstein J., Everett H. R., Feng L.
“Where am I? Sensor and Method for Mobile Robot Positioning”
The University of Michigan. 1996r

[4] Condit R.
„TB055. PS/2 to USB Mouse Translator”
Microchip Technology Inc. 2004r.

[5] Hallmann I.
„Określanie położenia robota mobilnego na podstawie obrazu z kamery”
Pomiary Automatyka Robotyka 1/2002r

[6] Jamrógiewicz T.
„Magistrala USB”

[7] McCane B., Abbott J.H., King T.
“On calculating the finite centre of rotation for rigid planar motion”
Article in press, 2004r.

[8] Metzger P.
„Anatomia PC”
HELION Wydanie VI 2001r.

[9] Oney W.
“Programming the Microsoft Windows Driver Model”
Microsoft Press. Second Edition 2003 r.

[10] Pasquier S., Rousselie D.
“Method of Greville. The Moore-Penrose generalized matrix inverse”
2003 r.

[11] Siemiątkowska B., Chojecki R., Marcinkiewicz P.
„Oczy dookoła głowy. System nawigacji robota mobilnego NAVIGATOR 1”
Akademia Nr2 (2) 2005r.

[12] Silberschatz A., Galvin P.B.
„Podstawy systemów operacyjnych”
WNT,2000

[13] *http://www.activrobots.com/ROBOTS/p2dx.html* stan na 26.08.2007

[14] *http://en.wikipedia.org/wiki/Computer_mouse* stan na 23.08.2007