AGH Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Praca magisterska

Łukasz Hanusiak, Mariusz Nowacki

kierunek studiów: **informatyka stosowana**

specjalność: **informatyka w nauce i technice**

Rozwój systemu robota mobilnego

Opiekun: **dr hab. inż. Marek Idzik**

Kraków, maj 2011

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....
(czytelny podpis)

.....
(czytelny podpis)

Kraków, maj 2011

Tematyka pracy magisterskiej i praktyki dyplomowej Łukasza Hanusiaka oraz Mariusza Nowackiego studentów V roku studiów kierunku informatyka stosowana, informatyka w nauce i technice

Temat pracy magisterskiej: **Rozwój systemu robota mobilnego**

Opiekun pracy:	dr hab. inż. Marek Idzik
Recenzenci pracy:	dr inż. Krzysztof Świentek
Miejsce praktyki dyplomowej:	WFiIS AGH, Kraków

Program pracy magisterskiej i praktyki dyplomowej

1. Omówienie realizacji pracy magisterskiej z opiekunem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Praktyka dyplomowa:
   - zapoznanie się z zasadą działania i budową robota Dark Explorer,
   - zapoznanie się z narzędziami i zasadami tworzenia oprogramowania sterującego pracą mikrokontrolera wbudowanego w robota,
   - dyskusja i analiza możliwych ścieżek rozwojowych zarówno warstwy sprzętowej jaki i programowej robota,
   - sporządzenie sprawozdania z praktyki.
4. Kontynuacja prac związanych z tematem pracy magisterskiej.
5. Zbieranie wyników z fazy testowania oprogramowania i sprzętu.
6. Analiza wyników testów, ich omówienie i zatwierdzenie przez opiekuna.
7. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: maj 2011

....

(Podpis kierownika katedry) (Podpis opiekuna)

Merytoryczna ocena pracy przez opiekuna:

Końcowa ocena pracy przez opiekuna: .....

Data: ..... Podpis: .....

Merytoryczna ocena pracy przez recenzenta:

Końcowa ocena pracy przez recenzenta: .....

Data: ..... Podpis: .....

Ze względu na obszerny zakres działań niniejsza praca magisterska była wykonywana przez zespół dwuosobowy. W celu ułatwienia zrozumienia treści zawartej w pracy nie rozdzielono jej na dwie części pisane przez poszczególnych autorów lecz została podzielona rozdziałami pod względem logicznym. Przypisanie opracowań redakcyjnych elementów pracy magisterskiej do autorów przedstawia poniższa lista:

Łukasz Hanusiak opracował:

— Historia i rozwój robotyki (rozdział 1).
— Akcelerometr trójosiowy (podrozdział 3.2).
— Czujnik odległości (podrozdział 3.5).
— Rozbudowa obudowy robota (podrozdział 3.8).
— Narzędzia do rozwoju systemu wbudowanego dla systemu Windows (podrozdział 4.1).
— Protokół komunikacji bluetooth (podrozdział 4.4).
— Lokalizacja twarzy na obrazie (podrozdział 4.7).
— Opis biblioteki do zarządzania robotem dla platformy .NET (podrozdział 5.1.1).
— Platforma mobilna - Windows Mobile (podrozdział 5.3).
— Specyfikacja poleceń protokołu komunikacji (dodatek A)

Mariusz Nowacki opracował:

— Analiza bazowej konfiguracji robota (rozdział 2).
— Inercjalny system nawigacyjny (podrozdział 3.1).
— Żyroskop (podrozdział 3.3).
— Magnetometr (podrozdział 3.4).
— Wyświetlacz LCD (podrozdział 3.6).
— Płyta rozszerzeń (podrozdział 3.7).
— Narzędzia do rozwoju systemu wbudowanego dla systemu Linux (podrozdział 4.2).
— Opis biblioteki AT91LIB (podrozdział 4.3).
— Modernizacja sposobu pobierania obrazu z kamery (podrozdział 4.6).
— Opis biblioteki do zarządzania robotem dla języka Java (podrozdział 5.1.2).
— Aplikacja do sterowania robotem dla komputerów PC (Java) (podrozdział 5.2).
— Platforma mobilna - Java ME (podrozdział 5.4).
— Wykonanie płytek drukowanych układów elektronicznych (dodatek B)

Wspólnie opracowano:

— Wprowadzenie
— Algorytm rekonstrukcji ścieżki powrotnej (podrozdział 4.5).
— Podsumowanie

Spis treści

Wprowadzenie

Celem pracy magisterskiej jest rozwój systemu robota mobilnego zrealizowanego pierwotnie w pracy magisterskiej pt.: „Rozwój systemu sterowania dla robota mobilnego" [1]. Temat pracy daje dużą swobodę w wyborze kierunku rozwoju części sprzętowej oraz programowej robota. Niniejsza praca zajmuje się proces rozbudową robota Dark Explorer, począwszy od analizy konfiguracji początkowej, poprzez poprawę wydajności istniejących elementów, aż po dodanie całkiem nowych funkcjonalności. Praca magisterska wymagała wiedzy z zakresu elektroniki, fizyki oraz szeroko pojętej informatyki.

W ramach pracy magisterskiej zostały podłączone oraz oprogramowane czujniki składające się na inercjalną jednostkę pomiarową. Jednostka ta umożliwia rejestrowanie toru ruchu, po którym robot jest przenoszony przez użytkownika. Podjęto również próbę stworzenia inercjalnego systemu nawigacyjnego w celu rozwiązania problemu rekonstrukcji ścieżki powrotnej na podstawie danych zarejestrowanych przy pomocy czujnika przyspieszenia oraz żyroskopu. W trakcie odtwarzania powrotnego toru ruchu robot, wykorzystując dane z dołączonych do niego dalmierzy, jest w stanie uniknąć zderzenia z napotkanymi przeszkodami jednocześnie starając się dotrzeć do punktu startowego. W pracy magisterskiej zawarty jest również przewodnik pozwalający programistom, nie posiadającym doświadczenia w tworzeniu oprogramowania dla systemów wbudowanych, zaznajomić się z narzędziami pomocnymi w rozwoju programów sterujących mikrokontrolerami opartymi o architekturę ARM. Poruszono także tematykę rozwoju aplikacji sterujących robotem dedykowanych na urządzenia mobilne oraz stacjonarne. Efektem tego jest stworzone oprogramowanie działające na komputerach oraz telefonach z obsługą technologii Java ME oraz .NET. Podczas tworzenia oprogramowania położono szczególny nacisk na przenośność wszystkich rozwiązań na różne środowiska uruchomieniowe oraz intuicyjną obsługę i prostą rozbudowę oprogramowania.

W rozdziale pierwszym przedstawiono dzieje robotyki od jej początków, aż po dzień dzisiejszy. W ramach tego rozdziału, zaprezentowane zostały dziedziny w jakich współcześnie roboty znajdują zastosowanie. Nie zabrakło również podstawowych informacji na temat sposobów klasyfikacji i podziału robotów. Rozdział ten ma na celu wprowadzenie użytkownika w zagadnienia z którymi robotyka zmaga się na co dzień.

Rozdział drugi przedstawia analizę sprzętu oraz oprogramowania robota Dark Explorer stworzonych w ramach poprzedniej pracy magisterskiej. Zwraca on uwagę na możliwości oraz potencjalne przeszkody które mogą się pojawić podczas rozwoju robota.

Trzeci rozdział opisuje narzędzia, oraz sposób ich konfiguracji, niezbędne podczas rozwoju oprogramowania dla systemów wbudowanych. Ze względu na znaczące różnice pomiędzy narzędziami przeznaczonymi dla systemu Windows i Linux, zostały one omówione w ramach oddzielnych podrozdziałów.

W rozdziale czwartym poruszana jest tematyka związana z tworzeniem oprogramowania sterującego robotem działającego na urządzeniach mobilnych oraz stacjonarnych.

Rozdział piąty przedstawia rozszerzenia warstwy sprzętowej wprowadzone w trakcie realizacji pracy magisterskiej. Czytelnik znajdzie tutaj szczegółowy opis zasady działania i możliwych zastosowań dla czujników dołączonych do nowej wersji robota Dark Explorer. W ramach rozdziału omówiono również sposób konstrukcji obudowy pozwalającej na wygodne podłączanie wszystkich elementów rozszerzających dotychczasowe możliwości robota.

Szósty rozdział prezentuje funkcje oprogramowania systemu wbudowanego rozwiązujące problemy takie jak: rekonstrukcja ścieżki powrotnej, omijanie przeszkód, pobieranie obrazu z kamery z maksymalną dostępną rozdzielczością oraz detekcja twarzy na obrazie statycznym. Opisany jest również protokół komunikacji bluetooth warstwy aplikacji modelu referencyjnego ISO.

Złożoność problemów realizowanych w ramach tej pracy magisterskiej oraz szeroki zakres wiedzy konieczny do jej realizacji wymagał, aby była ona realizowana w zespole dwuosobowym.

Rozdział 1

Historia i rozwój robotyki

Robotyka jest stosunkowo młodą dziedziną łączącą różne gałęzie nauk technicznych i nie tylko. Do pełnego zrozumienia zagadnień współczesnej robotyki oraz budowy i zastosowań robotów konieczna jest niejednokrotnie rozległa wiedza na temat elektroniki, mechaniki, inżynierii przemysłowej, matematyki oraz szeroko pojętej informatyki. Ponadto wiele nowopowstałych gałęzi wiedzy zajmujących się rozwojem sztucznej inteligencji, modelowaniem sztucznego życia czy rozwojem inżynierii wiedzy coraz częściej staje się nierozerwalnie związane z problemami współczesnej robotyki. Nie należy zapominać również o tym, że rozwój inżynierii wytwarzania oraz automatyki przemysłowej pozwala na nieustanny rozwój robotyki z którą mamy do czynienia w przemyśle w dniu dzisiejszym.

1.1. Historia robotyki

Pojęcie „robot" w literaturze pojawiło się po raz pierwszy w sztuce czeskiego pisarza Karel'a Capka w roku 1920. Termin „robot" oznacza w języku czeskim pracę lub służbę przymusową. Nieco ponad 20 lat później, amerykański uczony i pisarz Isaac Asimov w jednym ze swoich opowiadań po raz pierwszy używa słowa robotyka. W kolejnych latach Asimov w swojej twórczości niejednokrotnie wraca do problemu robotyki skutkiem czego jest wydanie w 1950 roku zbioru opowiadań pod tytułem „Ja, robot". Jako ciekawostkę można dodać, że w wydanym w 1942 roku opowiadaniu pod tytułem „Zabawa w berka", Asimov wprowadza trzy prawa robotyki według których, zdaniem autora, powinny być programowane roboty [2]. Zdefiniowane przez Asimov'a prawa robotyki w pełnym brzmieniu widoczne są na diagramie 1.1.

Rysunek 1.1: Prawa robotyki zdefiniowane przez Issaca Asimov'a

Nieco później, Isaac Asimov, jako uzupełnienie i prawo nadrzędne dodaje prawo zerowe o następującym brzmieniu „Robot nie może szkodzić ludzkości, ani przez zaniedbanie narazić ludzkości na szkodę" [3]. Oprócz wspomnianych powyżej podstawowych praw, zdefiniowano również inne prawa wynikające z prowadzonych w tej dziedzinie badań i rozwoju robotyki.

Takim sposobem urządzenie mechaniczne wykonujące zadania w sposób automatyczny otrzymało miano „robota" [4]. Operacje wykonywane przez robota mogą być bezpośrednio kontrolowane przez człowieka na podstawie przygotowanego wcześniej programu zawierającego zestaw reguł które umożliwiają robotowi wykonywanie określonych czynności. Możliwość wyręczenia człowieka przez maszynę w wykonywaniu monotonnych, złożonych i powtarzalnych czynności, niejednokrotnie z dużo większą wydajnością i precyzją, była jednym z podstawowych bodźców który sprzyjał rozwojowi robotyki już od samego jej początku. Pojęcie robot, używane jest również w stosunku do autonomicznie działających urządzeń pobierających informację z otocznia za pomocą różnego rodzaju czujników, nazywanych sensorami, oraz oddziałujących na swoje otoczenie i reagujące na jego zmiany.

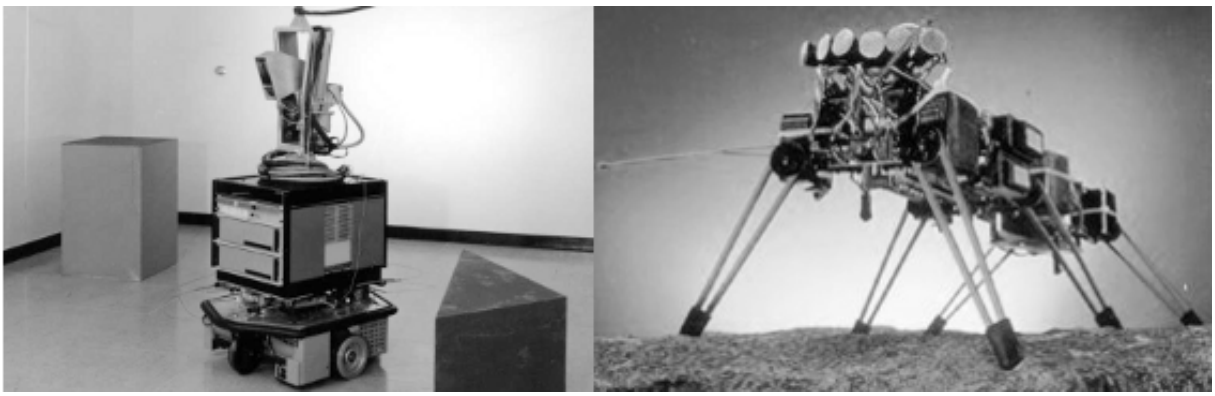
Dzięki gwałtownemu rozwojowi nauki i techniki w czasie II wojny światowej w roku 1956 GC. Devol i JF. Engelberger zainspirowani twórczością Asimov'a zaprojektowali i stworzyli pierwszy w dziejach ludzkości działający egzemplarz robota [3]. Engelberger założył firmę pod nazwą „Unimation" zajmującą się automatyzacją, pierwszym robotem stworzonym przez firmę Engelbergera był robot nazwany „Unimate". Został on zainstalowany w fabryce General Motors w Trenton gdzie został przystosowany do obsługi wysokociśnieniowej maszyny odlewniczej. W kolejnych latach roboty firmy „Unimation" znalazły swoje zastosowanie w innych gałęziach przemysłu, a sam Engelberger otrzymał miano ojca robotyki. [3]

Nieco później, bo w roku 1979 Robotics Industries Association zdefiniowało robota jako wielofunkcyjny, programowalny manipulator zaprojektowany do przenoszenia materiałów, narzędzi, części urządzeń poprzez serię programowalnych ruchów wykonywanych w celu realizacji różnych zadań. W myśl wspomnianej definicji jedną z podstawowych cech robota jest jego programowalność i możliwość dostosowywania się do zmiennych warunków środowiska pracy.

Pierwsze roboty projektowano z myślą o zastosowaniu ich do realizacji elementarnych zadań związanych z przenoszeniem elementów z jednego punktu do drugiego. Program pracy robota miał więc charakter sekwencji operacji prowadzących do realizacji określonego przez programistę zadania. W miarę rozwoju technicznego, stawiane przed robotami zadania wymusiły użycie przez konstruktorów czujników które pozwalały na zwiększenie poziomu interakcji robotów z otoczeniem, a co za tym idzie umożliwiły realizowanie zadań o wysokim stopniu złożoności. Od tego momentu kierunek rozwoju robotyki obrał sobie za cel stworzenie maszyny na tyle uniwersalnej i niezależnej aby mogła nosić miano androida. Jednak do realizacji tego zadania konieczne jest opracowanie sztucznej inteligencji na którą z pewnością przyjdzie nam jeszcze trochę poczekać.

Pierwszym krokiem na drodze do stworzenia androida było oderwanie robota od stałego miejsca instalacji i pozwolenie mu w miarę możliwości swobodne poruszanie się w dostępnej mu przestrzeni roboczej. Tak powstała klasa robotów które mogą przemieszczać się za pomocą kół, gąsienic czy nawet kończyn lub odnóży. Roboty tego typu potrafią pływać, latać i sprawnie poruszać się po lądzie dodatkowym ich atutem jest fakt iż większość z nich posiada niemal całkowitą autonomię i ograniczona jest jedynie poprzez wielkość otoczenia w jakim zostały umieszczone. Takim sposobem powstała grupa robotów nazywanych robotami mobilnymi. Cechą wspólną wszystkich urządzeń z tej rodziny była umiejętność swobodnego przemieszczania się oraz analiza najbliższego otoczenia. Na tej podstawie maszyna mogła przeprowadzić wnioskowanie pozwalające na podejmowanie dalszych akcji czy też przesłanie użytkownikowi odczytanych parametrów środowiska.

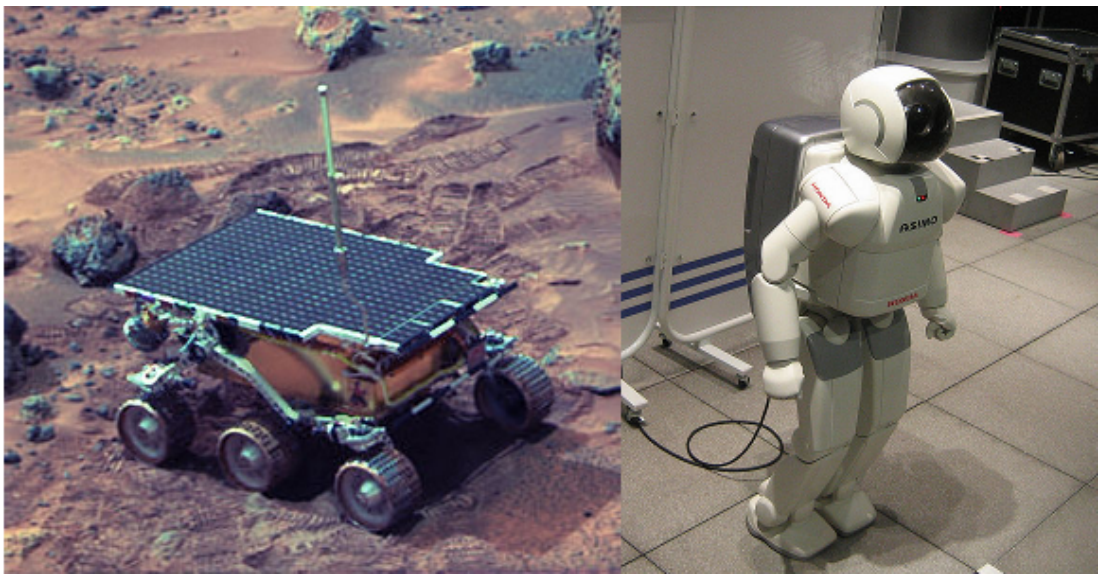
Historię robotów mobilnych zapoczątkował amerykański Uniwersytet w Stanford gdyż w roku 1968 jako pierwszy stworzył w pełni działający model robota mobilnego pod nazwą Shakey (rys. 1.2). Nazwa została zainspirowana szarpanymi ruchami z jakimi robot się poruszał. Głównym zadaniem robota było modelowanie otoczenia w którym się znajdował. W ślad za Uniwersytetem w Stanford ruszyło MIT. W roku 1983 posiadali już pierwszy działający model robota swobodnie skaczącego. Niecałe 6 lat później MIT stworzyło pierwszego robota kroczącego wzorowanego na owadach. Robot ten sterowany był za pomocą wielowarstwowych automatów o stanach skończonych, a ze światem zewnętrznym komunikował się za pomocą czułek, inklinometrów, czujników zbliżeniowych na podczerwień. Genghis, bo tak nazwany został robot, posiadał na pokładzie 4 ośmiobitowe jednostki obliczeniowe, ważył niecały kilogram i miał 35 cm długości.



Rysunek 1.2: Pierwsze roboty. Od lewej: Shakey (Stanford), Genghis (MIT)

Po sukcesie wspomnianych projektów, rozwój robotów mobilnych następował już bardzo dynamiczne. W latach 90 powstawało wiele różnych modeli robotów o bardzo różnorodnych rodzajach napędów oraz zestawach czujników umożliwiających interakcje ze światem zewnętrznym.

Swoistym ukoronowaniem prac było w 1997 roku stworzenie przez NASA robota o nazwie Pathfinder. Robot wyposażony był w czujniki laserowe, stereowizję, żyroskopy i inne rodzaje czujników o charakterze badawczym. Zasilany był on bateriami słonecznymi które pozwoliły mu na 83 dni nieprzerwanej pracy podczas której robot przebył około 100 metrów i wykonał 230 manewrów. W ostatnich latach do największych osiągnięć robotyki mobilnej można z pewnością zaliczyć powstanie robotów humanoidalnych takich jak japoński ASIMO (rys. 1.3).



Rysunek 1.3: Współczesne roboty. Kolejno: Pathfinder (NASA), Asimo (Honda)

Robot ten ważył 54 kg i mierzył 130 cm wysokości. Wersja z roku 2005 potrafiła biegnąc osiągnąć prędkość dochodzącą nawet do do 6 km/h. Ponadto robot potrafił wchodzić w interakcję z otaczającymi go ludźmi i przedmiotami. Urządzenie stworzone przez inżynierów z firmy Honda potrafiło rozpoznawać gesty takie jak podanie ręki, wskazanie kierunku czy machanie ręką na pożegnanie. Robot równie dobrze radził sobie z rozpoznawaniem twarzy, dźwięków i analizą otaczającego go środowiska. Potrafił on rozpoznać i omijać niebezpieczeństwa postawione na jego drodze jak na przykład schody czy osoby poruszające się w jego kierunku.

W roku 2006 podczas Robot World w Seulu grupa południowokoreańskich inżynierów prezentuje swojego robota EveR-2. EveR są robotami posiadającymi wygląd typowej dwudziestoletniej koreanki. Urządzenie potrafi rozmawiać i śpiewać dzięki wbudowanemu „silnikowi dialogu” (ang. embedded dialogue engine). Android EveR-2 w odróżnieniu od swojej poprzedniczki ma udoskonalony system wizyjny oraz możliwość wyrażania emocji takich jak znudzenie, zadowolenie, żal czy radość. Robot ma 170 cm wzrostu i waży około 60 kg. Twarz androida posiada wysoką elastyczność i poruszana jest przy pomocy 29 silniczków i licznych stawów które pozwalają na pełną swobodę w wyrażaniu emocji.

Wbudowany system rozpoznawania i syntezy mowy w połączeniu z możliwością wyrażania się za pomocą gestów pozwala na niemal całkowitą swobodę podczas rozmowy z robotem. Możliwości komunikacyjne robota są tak znakomite, że EveR-2 została pierwszą piosenkarką androidem. Jej pierwszy występ odbył się podczas jej prezentacji w Seulu gdzie na oczach publiczności odśpiewała koreańską balladę pt. „I will close my eyes for you".

Lata 2008 i 2009 zaowocowały powstaniem wielu robotów naśladujących w swoim zachowaniu niektóre zwierzęta. Bardzo imponującym przykładem takiego robota jest wyprodukowany przez firmę AeroVironment latający robot o nazwie Mercury. Urządzenie to potrafi „zawisnąć" w powietrzu poruszając jedynie skrzydłami dokładnie w taki sam sposób jak robią to prawdziwe kolibry. Lata te były również obfite w sukcesy w dziedzinie rozwoju sztucznego mózgu i sztucznej inteligencji. W roku 2008 grupa naukowców z University of Reading zbudowała robota sterowanego w całości za pomocą biologicznego mózgu z wyhodowanych neuronów. Biologiczny mózg w który wyposażony został zainstalowany w macierzy wieloelektrodowej (MEA). MEA jest to pewnego rodzaju naczynie które za pomocą około 60 elektrod przechwytuje sygnały elektryczne generowane przez komórki nerwowe i przekłada je na ruchy robota. W przypadku gdy robot napotka na swojej drodze przeszkody wspomniane wcześniej elektrody stymulują komórki nerwowe które tą samą drogą udzielają instrukcji na zachowania kół które pozwoliłoby na ominięcie wykrytej przeszkody. Robot jest całkowicie samodzielny i w całości sterowany przez własny mózg.

Obserwując postęp w dziedzinie robotyki można odnieść wrażenie iż w dzisiejszych czasach roboty znalazły dla siebie zastosowanie niemal w każdej dziedzinie życia. Od wielu lat sprawdzają się już w przemyśle, transporcie, budownictwie oraz są niezastąpione w środowiskach nieprzyjaznych człowiekowi, takich jak podmorskie głębiny czy otchłań kosmosu. Obszar zastosowań robotów jest tak szeroki iż wydawać się może, że jedynym czynnikiem ograniczającym rozwój współczesnej robotyki są względy czysto technologiczne. Nie staje to jednak na przeszkodzie do projektowania i tworzenia przez konstruktorów z całego świata rozwiązań coraz bardziej przybliżających ludzkość do stworzenia w pełni samodzielnego oraz inteligentnego androida.

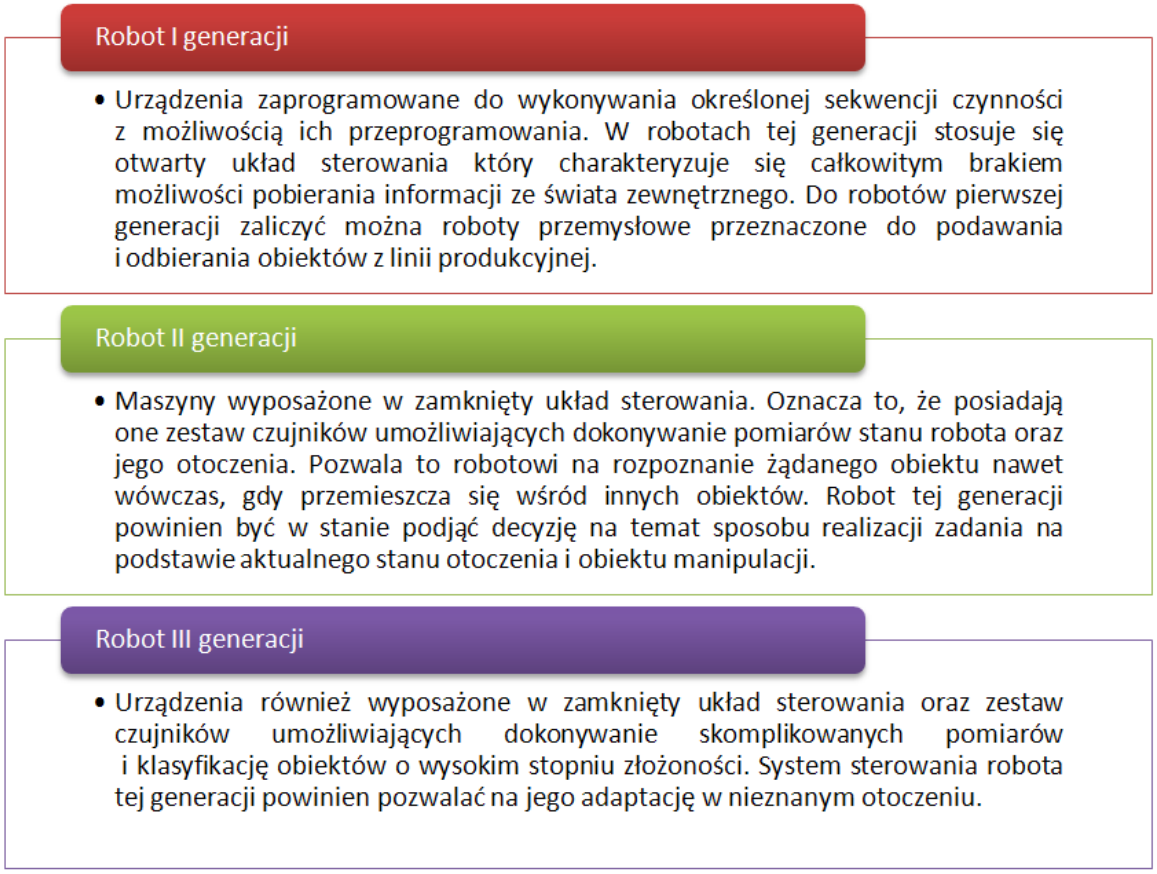
1.2. Obecny rozwój i zakres zastosowań robotyki

Robotyka zawładnęła wieloma dziedzinami życia człowieka od przemysłu poprzez zastosowania medyczne, wojskowe aż po urządzenia stosowane w gospodarstwach domowych. Roboty znalazły dla siebie zastosowanie w wykonywaniu zadań wymagających dużej szybkości, dokładności i wytrzymałości której nie może zapewnić praca wykonywana przez ludzi. W rezultacie wiele zadań wykonywanych w produkcyjnych zakładach pracy, dawniej wykonywane przez ludzi, zostało zastąpione przez roboty. Efektem tego jest zmniejszenie kosztów produkcji towarów masowych, szczególnie widocznych w przypadku części samochodowych i elektroniki. Zastosowanie robotów powoduje więc wzrost efektywności ekonomicznej i skraca czas uruchomienia produkcji, a co za tym idzie jest głównym czynnikiem ekonomicznym wspomagającym rozwój robotyki.

Istnieje szereg zadań które człowiek wykonuje lepiej niż maszyna, ale ze względu na męczący charakter pracy lub niebezpieczne środowisko jej wykonywania praca ludzka jest zastępowana przez maszyny. Stały rozwój technologii stosowanych w robotyce skutkuje w powstawaniu coraz bardziej zaawansowanych systemów co sprzyja powszechniejszemu ich stosowaniu. Zastosowanie robotów w życiu codziennym prowadzi do zwiększenia bezpieczeństwa pracy szczególnie na stanowiskach pracy zagrażających zdrowiu i życiu człowieka. Co więcej stały rozwój robotyki pozwala na prowadzenie badań w lokalizacjach fizycznie niedostępnych dla człowieka. Bardzo dobrym tego przykładem jest eksploracja odległych planet czy też wnętrz wulkanów. Jak można więc zauważyć rozwój robotyki stał się w chwili obecnej samonapędzającą się maszyną, w której powstawanie coraz doskonalszych robotów zwiększa na nie zapotrzebowanie, a to z kolei wymusza dalszy ich rozwój.

1.3. Klasyfikacja robotów

Współczesna różnorodność robotów doprowadziła do powstania wielu podziałów robotów ze względu na szereg różnych parametrów. Jednym z bardzo często spotykanych podziałów jest klasyfikacja ze względy na sposób programowania i możliwości komunikacyjne. Każda z grup tworzy tzw. generacje robotów. Można wyróżnić trzy generacje robotów (rys. 1.4) uwzględniając różnice w ich układzie sterowania oraz dostępne sensory [3].



Rysunek 1.4: Klasyfikacja robotów ze względu na ich generację

Istnieje szereg różnych klasyfikacji robotów bazujących na podziale ze względu na ich parametry techniczne. Jednym z bardziej znanych podziałów jest przedstawiona poniżej klasyfikacja ze względu na rodzaj środowiska w sposób w jaki się poruszają.

— roboty podwodne i poruszające się na wodzie,

— roboty lądowe oraz wodno-lądowe,

— roboty powietrzne,

Wśród robotów lądowych bardzo popularną grupą są roboty mobilne, podział których można przeprowadzić na podstawie sposobie poruszania się. Za pomocą takiego kryterium możemy wyróżnić roboty kołowe, kroczące, skaczące oraz pełzające. Innym popularnym sposobem klasyfikacji robotów jest podział ze względu na obszar ich zastosowania. Stosując kryterium takiego rodzaju istnieje możliwość wyróżnienia grup przedstawionych na rysunku 1.5

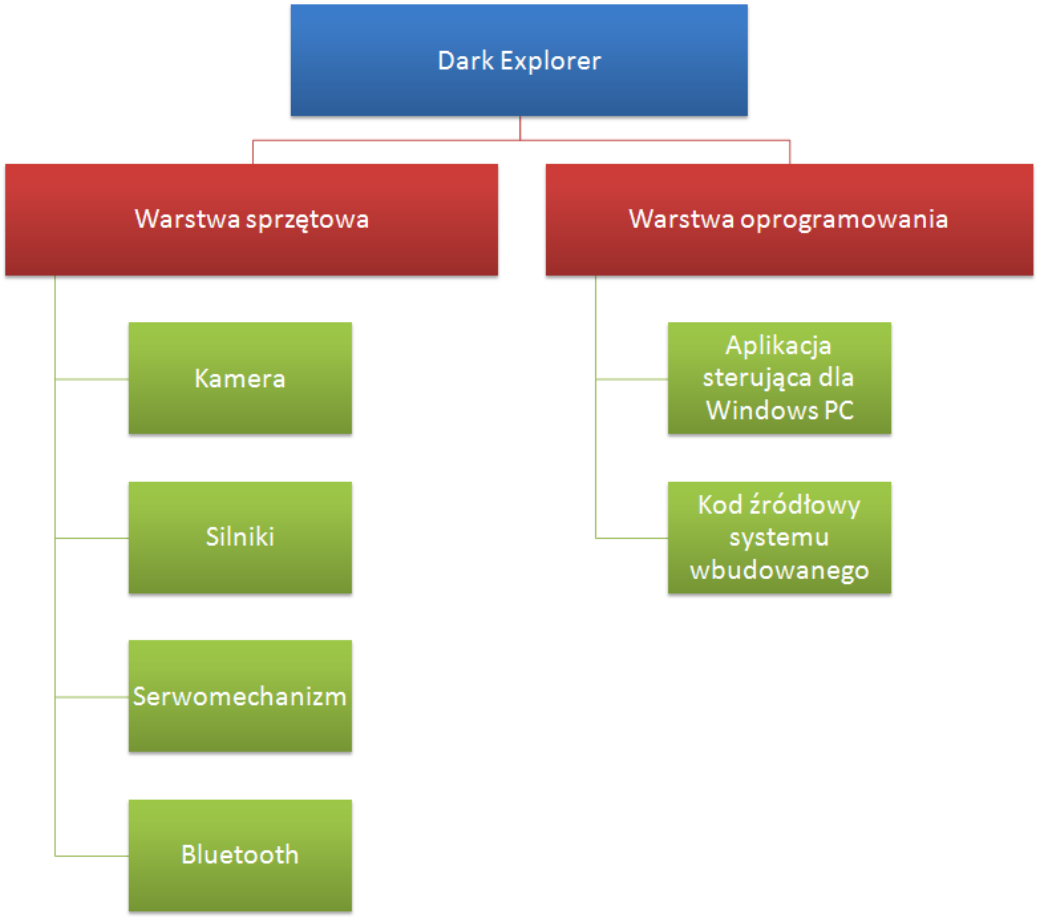
Rysunek 1.5: Podział robotów ze względu na obszar ich zastosowania

Omówione sposoby klasyfikacji nie wyczerpują w pełni problemu podziału robotów i ich zastosowań. Istnieje bowiem wiele charakterystyk opartych o takie parametry jak na przykład kształt czy rodzaj układu napędowego o których w tym rozdziale nie wspomniano. Celem przedstawionych informacji było zakreślenie obszaru zastosowań robotów oraz bogactwa ich różnorodności, a co za tym idzie uświadomienie czytelnikowi obszarów zastosowań robotyki we współczesnym świecie jak również potencjalnych problemów z jakim na codzień zmagają się ludzie projektujący i budujący roboty.

Rozdział 2

Analiza bazowej konfiguracji robota

W tym rozdziale zostanie opisana konfiguracja pierwotna robota z wyszczególnieniem elementów, które mogą być problemem przy dalszym rozwoju możliwości tego urządzenia. Struktura projektu zaprezentowana jest na rysunku 2.1.



Rysunek 2.1: Struktura platformy robota mobilnego zrealizowanej w ramach poprzedniej pracy magisterskiej [1]

2.1. Analiza sprzętu

Dark Explorer jest autonomicznym robotem mobilnym z wbudowaną kolorową kamerą cyfrową VGA. Jego podstawowe możliwości to:

— jazda ze zmienną prędkością w przód, tył, lewo oraz prawo

— komunikacja z urządzeniami zewnętrznymi przy pomocy technologii bluetooth

— wykonywanie zdjęć z maksymalną rozdzielczością 160 × 100 pikseli w kolorze oraz 320 × 200 pikseli w odcieniach szarości

— poruszanie wieżyczką na której zainstalowana jest kamera

— wykrywanie prostych wzorców przy pomocy analizy obrazu oraz podążanie za nimi

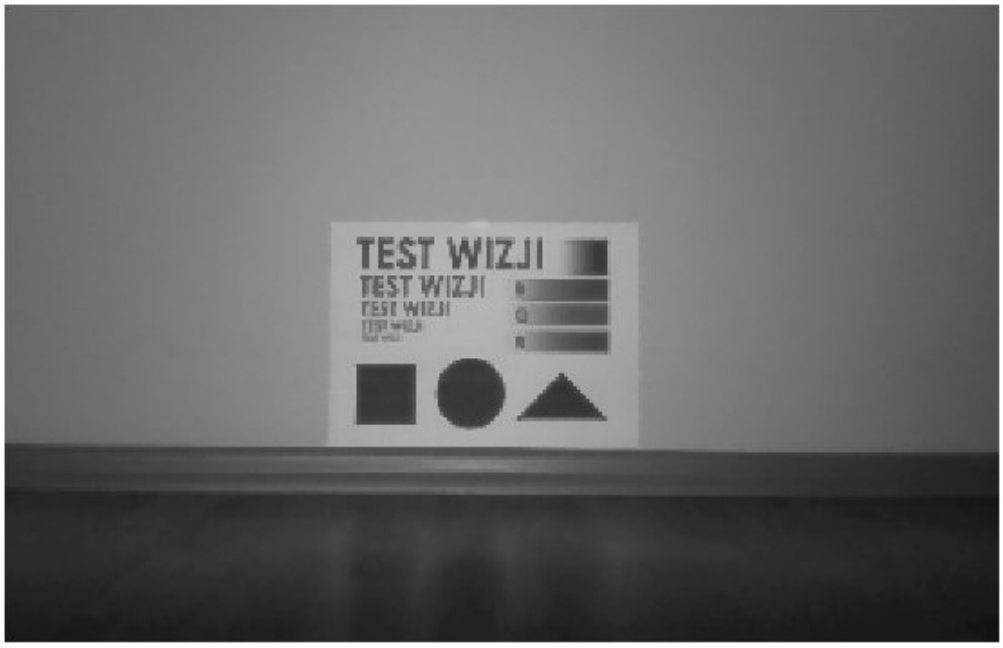
2.1.1. Elementy elektroniczne

Dark Explorer został wyposażony w mikrokontroler zarządzający AT91Sam7s256 o częstotliwości pracy zegara maksymalnie do 50 MHz oraz wbudowanej szybkiej pamięci SRAM 64 kB. Mikrokontroler ten jest bogaty w różnego rodzaju urządzenia peryferyjne takie jak na przykład: TWI[1], RTT[2], PDC[3], AIC[4], PWM[5], ADC[6]. To tylko część z nich. Dzięki tak wielkiemu wyborowi urządzeń peryferyjnych mikrokontroler ten daje nam duże możliwości rozwoju konfiguracji. Częstotliwość pracy mikrokontrolera ARM7 jest wystarczająca do zadań przez niego wykonywanych.

Większość elementów elektronicznych robota jest umieszczonych na płycie głównej zaprojektowanej przez autora projektu robota. Jest ona bardzo dobrze przemyślana ponieważ pozwala na wykorzystywanie urządzeń peryferyjnych mikrokontolera w praktycznie dowolny sposób. Większość wyjść oraz wejść mikrokontrolera nie jest połączona na stałe z konkretnymi podzespołami Dark Explorera lecz poprzez zworki, które pozwalają na podłączanie i odłączanie poszczególnych urządzeń.

Twórca Dark Explorera wyposażył go w kamerę cyfrową o maksymalnej rozdzielczości 640 × 480 pikseli. Jest to urządzenie $PO6040$ firmy Pixelplus. Można go konfigurować przy pomocy interfejsu I^2C. W konfiguracji pierwotnej robot potrafi odbierać

[1] TWI – Two Wire Interface znany również jako I^2C, interfejs szeregowej komunikacji danych
[2] RTT – Real Time Timer, służy do odmierzania dłuższych odcinków czasu
[3] PDC – Peripheral DMA Controller, kontroler DMA
[4] AIC – Advanced Interrupt Controller, kontroler przerwań
[5] PWM – Pulse Width Modulation Controller
[6] ADC – Analog to Digital Converter, konwerter analogowo cyfrowy



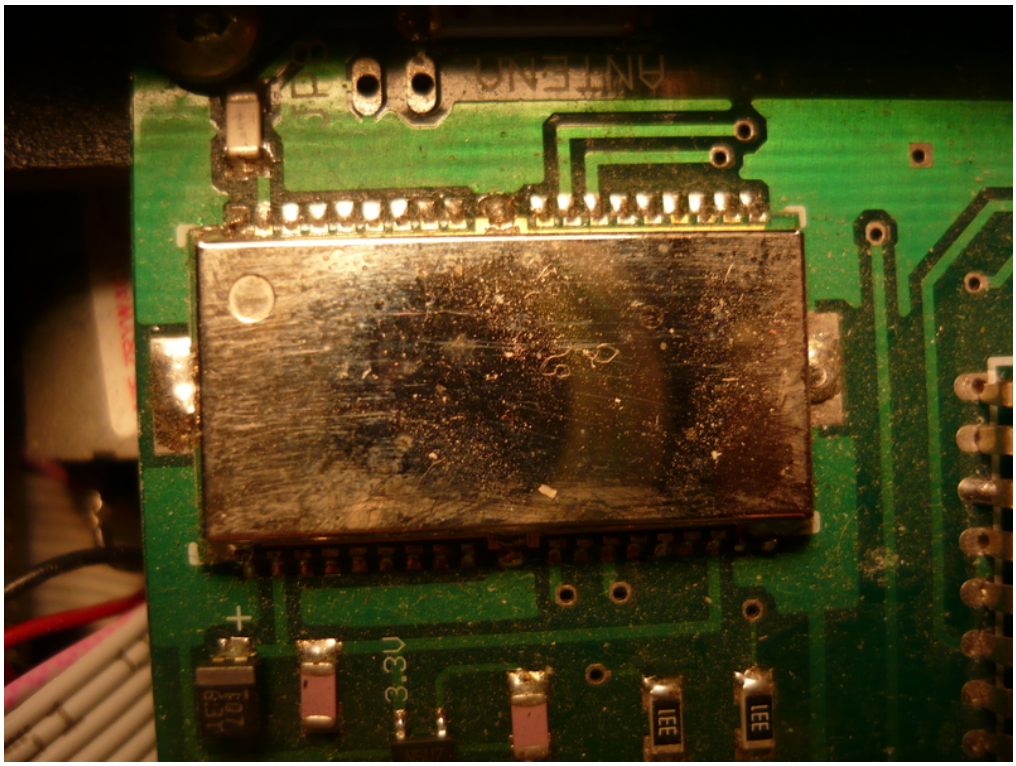
Rysunek 2.2: Obraz wykonany przy pomocy kamery zamontowanej w robocie. 320 × 200 pikseli w odcieniach szarości [1]

zdjęcia o maksymalnej rozdzielczości 320 × 200 pikseli w odcieniach szarości (rys. 2.2) oraz 160 × 100 pikseli w kolorze (rys. 2.3). Tak niskie rozdzielczości nie są wystarczające do wykrywania i tym bardziej rozpoznawania twarzy na obrazie dlatego też konieczna jest poprawa tych parametrów.

Rysunek 2.3: Obraz wykonany przy pomocy kamery zamontowanej w robocie. 160 × 100 pikseli w kolorze. [1]

W skład robota wchodzą także silniki napędowe, serwomechanizm, dioda oświetleniowa (LED[7]) oraz moduł bluetooth. Wszystkie te elementy muszą być podłączone do mikrokontrolera przez konkretne wejścia/wyjścia. Z tego powodu do dyspozycji osób przeprowadzających dalszy rozwój robota pozostało jedynie pięć wejść/wyjść cyfrowych oraz trzy wejścia analogowe. Jest to kolejna przeszkoda którą trzeba było pokonać.

Robot mobilny komunikuje się z urządzeniami zewnętrznymi przy pomocy modułu bluetooth BTM-222 firmy Rayson. Maksymalna przepustowość danych z jaką potrafi działać wynosi 3Mb/s. Obsługiwane przez niego interfejsy to: USB, UART[8] oraz PCM[9]. Mikrokontroler komunikuje się z modułem bluetooth przy pomocy interfejsu UART o maksymalnej przepustowości 460,8 kb/s. Jak widać takie rozwiązanie nie wykorzystuje pełnych możliwości modułu BTM-222. Bardziej efektywne byłoby skorzystanie z interfejsu USB który potrafi przesyłać dane z prędkością maksymalną od 1,5Mb/s w standardzie USB 1.0 do 5Gb/s w standardzie USB 3.0. Aby dokonać zmiany interfejsu komunikacyjnego pomiędzy mikrokontrolerem a modułem bluetooth konieczna jest ingerencja w konstrukcję płyty głównej robota, gdyż BTM-222 jest przylutowany bezpośrednio do niej (rys. 2.4).

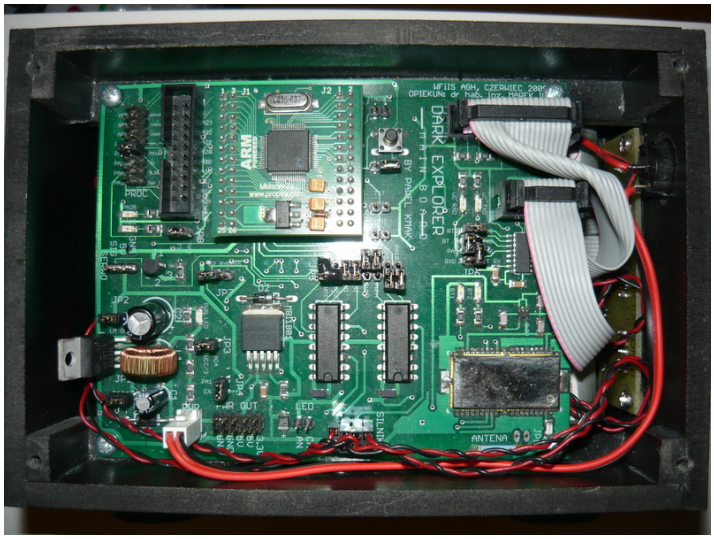


Rysunek 2.4: Zdjęcie modułu bluetooth zamontowanego na płycie głównej robota.

[7] LED - Light Emitting Diode
[8] UART - Universal Asynchronous Receiver and Transmitter
[9] PCM - Pulse Code Modulation

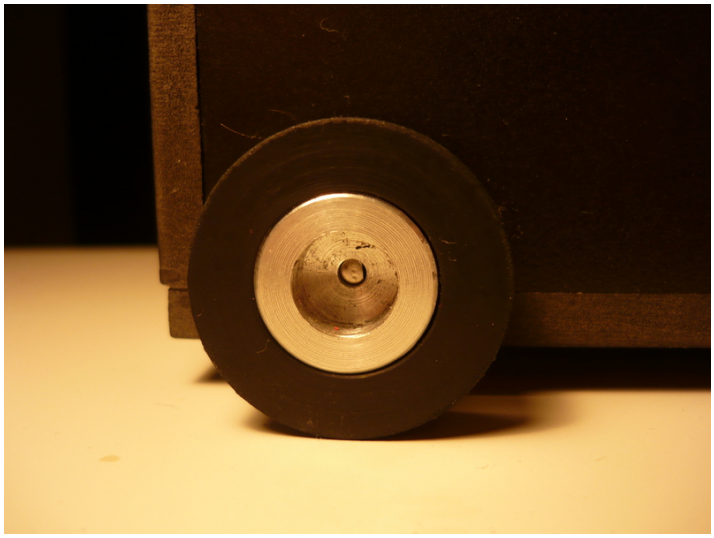
2.1.2. Elementy mechaniczne

Konstrukcja obudowy Dark Explorera została wykonana z tworzywa sztucznego i jest dopasowana do elementów które zostały zaprojektowane przez autora. Wewnątrz obudowy nie ma miejsca na jakiekolwiek nowe podzespoły, dlatego też konieczna będzie jej modyfikacja (rys. 2.5).



Rysunek 2.5: Widok na płytę główną Dark Explorer'a

Podczas testów konfiguracji pierwotnej zauważono lekkie kłopoty robota z poruszaniem się po gładkich powierzchniach. Najprawdopodobniej jest to spowodowane kółkami (rys. 2.6) zamontowanymi przy robocie, które tracą przyczepność na nieco bardziej śliskim podłożu. Możliwe, że podczas rozwoju robota, konieczna będzie ich wymiana w celu zapewnienia dobrej przyczepności i poprawnego toru jazdy urządzenia.



Rysunek 2.6: Koło napędowe Dark Explorer'a

2.2. Analiza oprogramowania

2.2.1. Firmware

Jako firmware określane jest oprogramowanie którego komendy są wykonywane przez mikrokontroler. W tym podrozdziale zajmiemy się kodem źródłowym opisującym sposób działania robota.

Pliki składające się na kod źródłowy robota możemy podzielić na dwa rodzaje:

— napisane przez autora

— dostarczone przez producenta mikrokontrolera

Pliki dostarczone przez producenta mikrokontrolera są po prostu biblioteką, dzięki której programista może kontrolować działanie różnych urządzeń peryferyjnych mikrokontrolera. Biblioteka wykorzystana przez twórcę Dark Explorera bazuje na prostych funkcjach wpisujących podawane wartości do odpowiednich rejestrów mikrokontrolera. Można powiedzieć, że jest to biblioteka niskopoziomowa. Wymaga ona dużej wiedzy na temat urządzeń peryferyjnych z których mamy zamiar korzystać oraz rejestrów które nimi sterują. Praktycznie nie jest możliwe oprogramowanie robota przy pomocy tej biblioteki bez wcześniejszego dokładnego zaznajomienia się z notą katalogową mikrokontrolera AT91Sam7S256. Podczas rozwoju robota przydatnym byłoby wykorzystanie innej biblioteki, która jest bardziej przyjazna dla programisty.

Bazowa wersja firmware'u napisanego przez autora Dark Explorera została podzielona na sześć plików:

board.h	Plik nagłówkowy z informacjami dotyczącymi mikromodułu mikrokontrolera
pio.h	Definicje określające wejścia i wyjścia ogólnego przeznaczenia zamontowane na płycie głównej robota
main.c	Inicjalizacje początkowe, obsługa przerwań systemowych, interpretacja komend bluetooth, pętla główna programu
peripherals.c	Funkcje do obsługi urządzeń peryferyjnych
rozpoznawanie.c	Analiza i rozpoznawanie obrazu
utils.c	Procedury sterujące i obliczeniowe wyższego poziomu

Biorąc pod uwagę ilość kodu znajdującą się w plikach, taki podział wydaje się być wystarczający. W przyszłości trzeba zadbać o gęstsze partycjonowanie kodu źródłowego na logiczne bloki w celu zapewnienia przejrzystości kodu źródłowego.

2.2.2. Aplikacja zarządzająca

Do kontrolowania robota została stworzona aplikacja graficzna (rys. 2.7) działająca pod systemem Windows. Interfejs graficzny aplikacji jest atrakcyjny i przejrzysty. Pozwala ona na kontrolę następujących funkcji robota:

— kontrola kierunku i szybkości jazdy robota za pomocą wektora wodzącego oraz klawiatury

— kontrola wieży obserwacyjnej

— włączanie oraz wyłączanie diody oświetlającej

— informacja o stanie akumulatorów

— zarządzanie trybem autonomicznym robota

— konfiguracja oraz status połączenia bluetooth

Rysunek 2.7: Okno aplikacji zarządzającej Dark Explorera

Aplikacja zarządzająca została napisana tylko na jeden rodzaj systemu operacyjnego. Nie ma możliwości kontrolowania robota przy pomocy komputera z zainstalowanym systemem operacyjnym innym niż Microsoft Windows. Dlatego też, konieczne będzie stworzenie aplikacji zarządzającej, którą będzie można modyfikować oraz uruchomić na dowolnym systemie.

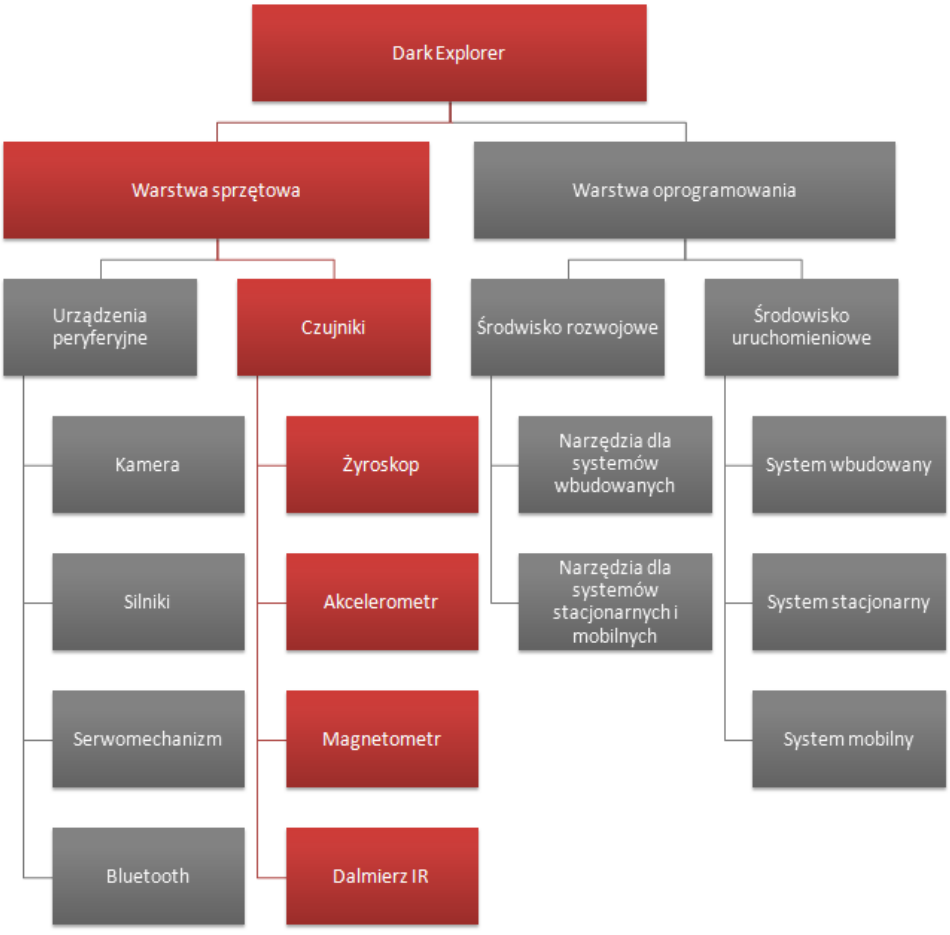
W celu komunikacji z robotem, aplikacja zarządzająca wysyła do niego odpowiednie komunikaty. Komunikaty te składają się z pojedynczego znaku i liczby. Możliwe, że w przypadku obsługi większej ilości urządzeń na Dark Explorerze, będzie konieczne stworzenie bardziej zaawansowanego protokołu komunikacyjnego. Zapewni to możliwość sprawdzenia czy dana komenda jest poprawnie skonstruowana, czy może zawiera jakieś błędy które pojawiły się podczas transmisji.

Zarówno w aplikacji zarządzającej jak i na robocie został zastosowany mechanizm retransmisji pakietów z obrazem w przypadku błędów w trakcie połączenia. Jest to bardzo dobry pomysł w szczególności w przypadku przesyłania takiej ilości danych jaką generuje kamera.

Rozdział 3

Rozwój sprzętowej warstwy robota mobilnego

W ramach pracy magisterskiej zostały opisane elektroniczne i mechaniczne elementy sprzętowe robota Dark Explorer. W tym rozdziale został opisany sposób tworzenia tych elementów oraz zasadę ich działania. Patrz rysunek 3.1



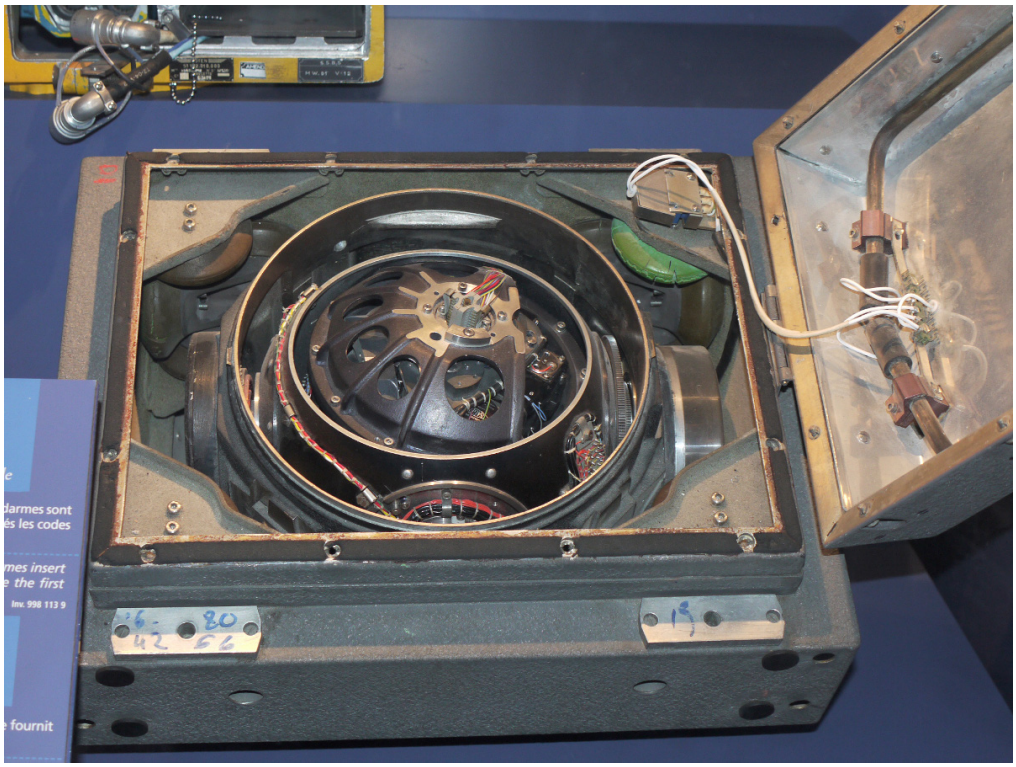
Rysunek 3.1: Struktura platformy robota mobilnego po zakończeniu prac. Kolorem czerwonym oznaczono zakres prac opisanych w bieżącym rozdziale.

3.1. Inercjalny system nawigacyjny

W celu rozwinięcia możliwości robota, zamontowano w nim elementy, przy pomocy których została podjęta próba stworzenia inercjalnego systemu nawigacyjnego (INS[1]). Założeniem było, aby robot mobilny zapamiętał tor ruchu po jakim się porusza, gdy jest niesiony na ręce osoby operującej nim. Następnie na podstawie wykonanych pomiarów robot miał powrócić po zapamiętanym torze w miejsce początkowe. Poniższy rozdział omawia pokrótce czym jest INS, opisuje zasadę działania jego elementów oraz sposób wykorzystania tych podzespołów do osiągnięcia wystarczająco dobrych efektów.

3.1.1. Wprowadzenie do INS

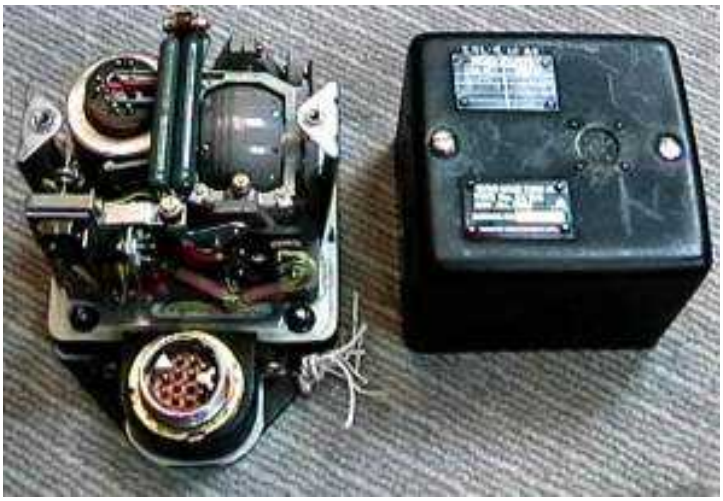
Inercjalny system nawigacyjny (rys. 3.2) jest to narzędzie służące do określenia położenia, prędkości oraz orientacji obiektu w przestrzeni, bez korzystania z żadnych zewnętrznych elementów naprowadzających, które byłyby dla niego punktem odniesienia. Wykorzystuje on jedynie elementy wbudowane, składające się na inercjalną jednostkę pomiarową (IMU[2]).



Rysunek 3.2: INS francuskiego IRBM S3[3]

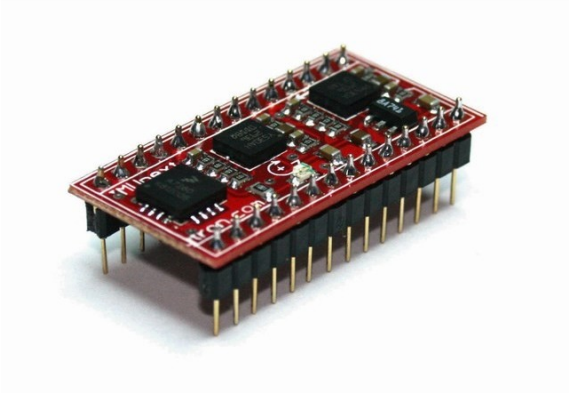
[1] Inertial Navigation System
[2] Inertial Measurement Unit
[3] źródło: `http://en.wikipedia.org/wiki/Inertial_navigation_system`

Inercjalne systemy nawigacyjne mają zastosowanie tam, gdzie jest potrzebna informacja o aktualnym położeniu obiektów, natomiast nie ma możliwości odbioru sygnału zewnętrznego, wymaganego do działania takich urządzeń jak na przykład urządzeń GPS. Systemy tego typu są stosowane w: samolotach, statkach, łodziach podwodnych, pojazdach bezzałogowych czy statkach kosmicznych. Pierwsze rozwiązania tego typu były drogie i bazowały na bardzo dużych gabarytowo elementach mechanicznych (rys. 3.3).



Rysunek 3.3: Żyrokompas samolotowy[4]

Rozwój miniaturowych układów elektromechanicznych MEMS[5] (rys. 3.4) otworzył przed nami cały wachlarz potencjalnych nowych zastosowań INS, np. do śledzenia ruchów ludzi bądź zwierząt.



Rysunek 3.4: IMU stworzone przy pomocy układów MEMS, cena ok.: 80$[6]

[4] źródło: `http://www.gyroscopes.org/uses.asp`
[5] Microelectromechanical Systems
[6] źródło: http://www.flytron.com

3.1.2. Elementy IMU robota

System nawigacyjny o którym mowa w tym rozdziale oblicza swoje położenie na podstawie ciągłego badania przyspieszenia liniowego oraz prędkości kątowej. INS musi otrzymać na starcie wartości początkowe położenia oraz prędkości z jaką się porusza, aby móc zacząć wyznaczać dalsze przemieszczenie i zmiany w orientacji.

Robot mobilny został wyposażony w IMU składające się z następujących elementów MEMS:

— cyfrowy żyroskop trójosiowy L3G4200D firmy STMicroelectronics (pomiar kąta obrotu robota)

— analogowy akcelerometr trójosiowy MMA7260 firmy Freescale (wykrywanie kroków – przybliżanie wartości pokonanej drogi)

— cyfrowy magnetometr dwuosiowy firmy MMC2120MG firmy Memsic (określanie zwrotu i kierunku robota)

Elementy użyte do wykonania IMU zostały wybrane pod kątem walorów ekonomicznych. Nie były przeprowadzane testy porównawcze pomiędzy podzespołami danego typu. Zasada działania poszczególnych elementów oraz opis ich wykorzystania można znaleźć w kolejnych podrozdziałach.

3.2. Akcelerometr trójosiowy

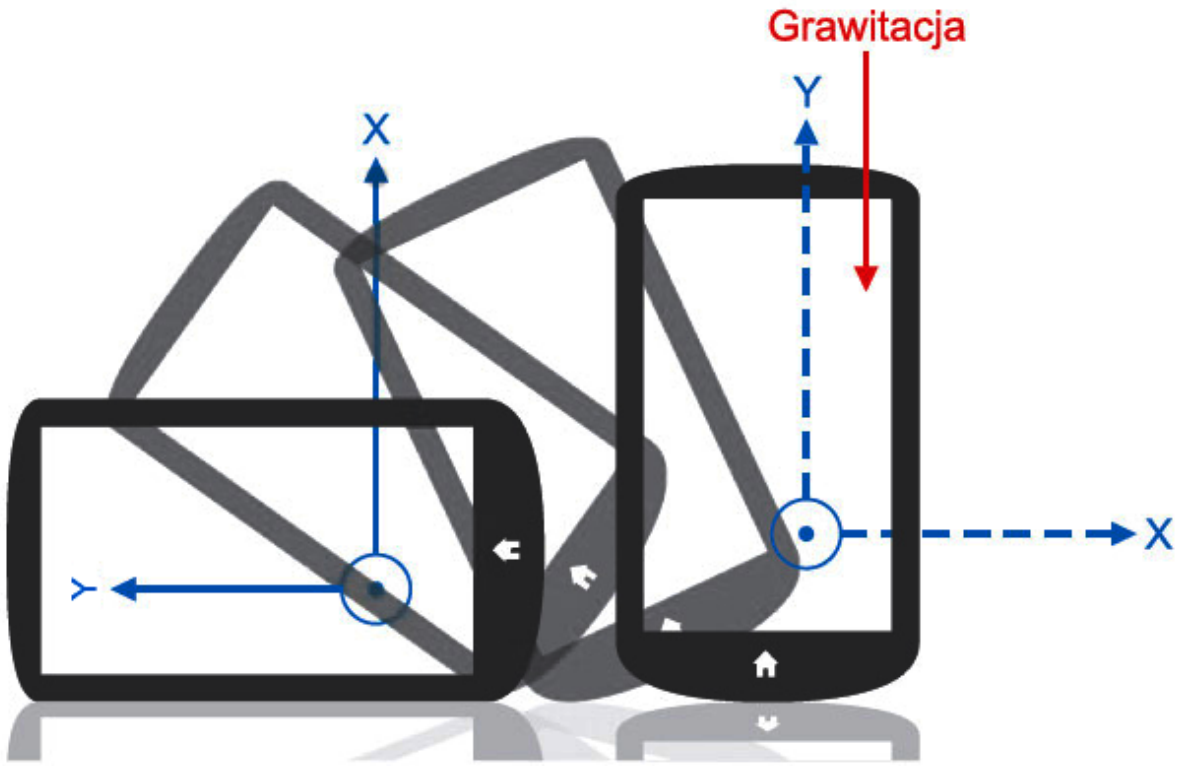
W latach 90 akcelerometry instalowano jedynie w roli mechanizmów służących do uruchamiania poduszek powietrznych w samochodzie w przypadku wystąpienia zderzenia. W chwili obecnej akcelerometry wykonane w technologii MEMS można znaleźć w większości urządzeń codziennego użytku tj. telefony komórkowe, konsole do gier, komputery przenośne czy cyfrowe aparaty fotograficzne. Powodem tak spektakularnego wzrostu popularności było pełne rozwinięcie się technologii MEMS. W dobie miniaturyzacji fakt, iż w miejsce do niedawna używanych urządzeń można wstawić pojedynczy układ scalony jest jednym z podstawowych powodów dla których, między innymi, akcelerometry cieszą się tak szerokim spektrum zastosowań. Jednakże, ceną tak daleko idącej miniaturyzacji jest duży spadek dokładności pomiarowej w porównaniu do akcelerometrów opartych o tensometry. Jak się jednak okazuje, do codziennego użytku precyzja oferowana przez tego typu czujniki jest w zupełności wystarczająca.

Bardzo dobrym przykładem zastosowania akcelerometru jest dostępna w większości aparatów cyfrowych funkcja optycznej stabilizacji obrazu (Optical Image Stabilization). Funkcja ta polega na redukcji zniekształceń spowodowanych drżeniem rąk fotografa w trakcie akwizycji obrazu. Problem jest tym poważniejszy iż nieustanna miniaturyzacja tego typu urządzeń dodatkowo wzmacnia zakłócenia związane z mimowolnym[7] drżeniem rąk użytkownika. Dlatego też większość aparatów wyposażonych jest w czujniki które mierzą wspomniane drgania, a następnie uzyskane informacje przekazują do modułu sterującego położeniem soczewek i przetwornika obrazu. Następnie moduł ten dostosowuje parametry pracy aparatu tak aby zminimalizować wpływ niepożądanych przesunięć na efekt końcowy pracy jakim jest zdjęcie.

Innym popularnym przykładem zastosowania akcelerometrów są pedometry, potocznie nazywane krokomierzami. Dzięki zastosowanej technologii czujniki w krokomierzach mogą dokonywać równocześnie pomiaru względem trzech osi. Co więcej tego typu urządzenia umożliwiają dokonywanie pomiarów niemal całkowicie niezależne od swojej orientacji, a ich niewielkie rozmiary pozwalają na ich integrację z dowolnymi urządzeniami mobilnymi. Akcelerometry wykonane w technologi MEMS najlepiej sprawdzają się podczas pomiaru przyspieszenia statycznego pozwalającego jednoznacznie wyznaczyć kąt odchylenia urządzenia względem pionu. Można nimi dokonywać również pomiaru przyspieszenia dynamicznego pojawiającego się na skutek wibracji, uderzenia czy innego rodzaju ruchu.

[7] Ręka przeciętnego człowieka drży mimowolnie z częstotliwością od 10 do 20 Hz

Niestety, pomimo swoich licznych zalet akcelerometry MEMS posiadają pewne ograniczenia które mogą być bardzo istotne podczas projektowania bardziej skomplikowanych systemów. Jedynym z podstawowych problemów jest brak możliwości uzyskania jednoznacznej informacji na temat orientacji urządzenia zarówno w pionie jak i poziomie. W ramach przykładu spróbujmy przeanalizować wykorzystanie akcelerometru do przełączania widoku na ekranie w zależności od jego orientacji. W przypadku gdy oś X lub Y są równoległe z wektorem grawitacji na podstawie informacji z akcelerometru można w bardzo prosty sposób ustalić w jakim położeniu znajduje się aktualnie urządzenie (rys. 3.5).



Rysunek 3.5: Pokrywanie się osi ekranu z kierunkiem wektora grawitacji pozwala na precyzyjne wyznaczenie orientacji urządzenia [5]

Natomiast w przypadku gdy osie ekranu ustawią się prostopadle do wektora grawitacji dane pozyskane z akcelerometru nie pozwalają na wyznaczenie jednoznacznej pozycji w jakiej znalazło się urządzenie [5]. Co więcej w przypadku aplikacji w których akcelerometr jest źródłem informacji o odchyleniu od pionu, konieczne jest rozdzielenie zmian związanych z wpływem grawitacji od szumów związanych z przyspieszeniem dynamicznym. Tak pozyskane informacje będą służyć jako punkt odniesienia pozwalającego na wyznaczenie kąta wychylenia. Głównym problemem jest fakt, że sygnał taki można wyróżnić jedynie w chwili gdy akcelerometr jest w stanie spoczynku, co w przypadku niektórych rodzajów zastosowań może dawać nieoczekiwane rezultaty i zachwiać stabilność pracy całego systemu.

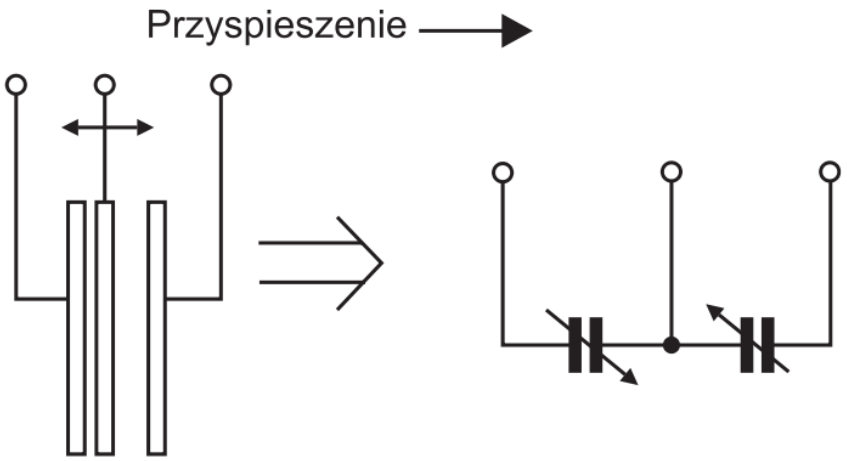
Jeżeli założymy, że zmiany odpowiadające przyspieszeniu dynamicznemu mają dużą częstotliwość to możemy przy użyciu filtru dolnoprzepustowego usunąć zakłócenia związane z drganiami wywołanymi przez użytkownika. Oczywistą implikacją powyższego założenia jest założenie, że zmiany kąta nachylenia względem pionu są wolne gdyż w przeciwnym wypadku informacje o zmianie odchylenia zostaną usunięte przez wspomniany filtr. Aby pominięcie wspomnianych ograniczeń stało się możliwe konieczne jest zastosowanie dodatkowych czujników takich jak np. żyroskop które dostarczą uzupełniających informacji pozwalających na stabilną implementację funkcjonalności.

3.2.1. Rodzaje akcelerometrów

Istnieje wiele różnych rodzajów akcelerometrów. Akcelerometry mechaniczne przypominają po części zachowanie pasażera w samochodzie który gwałtowanie przyspiesza i zwalnia. Posiadają one element masy przyczepiony do elementu sprężynowego umieszczonego w całości w zewnętrznej obudowie. Kiedy taki akcelerometr zacznie przyspieszać obudowa zewnętrzna przemieści się podczas gdy punkt masy wychyli się w kierunku przeciwnym do przyspieszenia co spowoduje rozciągnięcie się sprężyny wprost proporcjonalne do siły która spowodowała przyspieszenie. Mierząc więc odległość na jaką nastąpiło wychylenie jesteśmy w stanie wyznaczyć wartość działającej siły, a co za tym idzie również wartość przyspieszenia. Opisana powyżej zasada pomiaru leży u podstaw działania sejsmografów które za pomocą masy dołączonej do elementu piśmiennego rejestrują siły występujące w chwili trzęsienia ziemi.

Alternatywnym podejściem jest wykorzystanie sygnałów elektrycznych i magnetycznych do pomiaru przyspieszenia. Wśród tego rodzaju akcelerometrów możemy wyróżnić następujące trzy typy: akcelerometr piezorezystywny, akcelerometr pojemnościowy oraz akcelerometr piezoelektryczny. Istnieją również akcelerometry które dokonują pomiaru przyspieszenia w oparciu o efekt Halla.

Przyspieszeniomierze piezorezystywne mają punkt masy dołączony do potencjometru który zwiększa i zmniejsza przepływ prądu w zależności od siły jaka oddziałuje na czujnik. Bardzo podobną zasadą działania charakteryzują się akcelerometry pojemnościowe, te jednak wykorzystują efekt zmiany pojemności kondensatorów zastosowanych w miejscu w którym poprzednio użyty został potencjometr. Schemat ideowy jednoosiowego akcelerometru pojemnościowego widoczny jest na rysunku 3.6.



Rysunek 3.6: Schemat ideowy akcelerometru pojemnościowego dokonującego pomiaru wzdłuż jednej osi [6]

Ostatnim rodzajem akcelerometrów są urządzeniami bazujące na piezoelektrycznych kryształach takich jak na przykład kwarc. W urządzeniach tych punkt masy, pod wpływem przyspieszenia, naciska na kryształ co powoduje, powstawanie napięcia które jest wykorzystywane do wykonywania pomiaru. Cechą charakterystyczną tego rodzaju czujników jest brak wskazań wartości przyspieszenia statycznego [7].

3.2.2. Algorytm rozpoznawania kroków

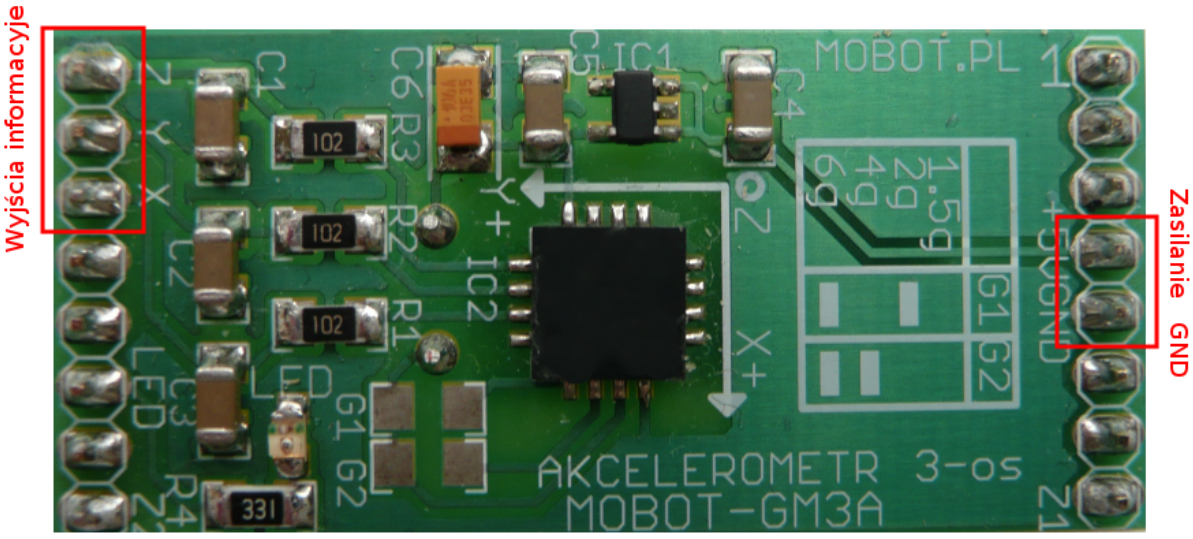
Do poprawnego działania algorytmu zaprezentowanego w ramach noty aplikacyjnej [8] wymagane jest wykorzystanie akcelerometru trójosiowego. Jak już wspomniano wcześniej, dzięki zastosowaniu urządzenia tego typu jesteśmy w stanie wyeliminować problemy wynikające z konieczności uwzględnienia aktualnego wychylenia akcelerometru względem wektora grawitacji. W przypadku gdyby możliwe byłoby korzystanie tylko z pojedynczych osi poprawne działanie algorytmu byłoby zapewnione jedynie w ściśle określonej pozycji co jest znaczącym utrudnieniem biorąc pod uwagę niehomogeniczny charakter ruchów wykonywanych podczas chodzenia. Aby wyeliminować wspomniany problem do ostatecznego rozwiązania brana jest jedynie wartość stanowiąca złożenie wartości przyspieszenia dla poszczególnych osi. Do wyznaczenia bezwzględnego przyspieszenia na podstawie danych z osi X, Y, Z użyte zostało równanie 3.1.

$$a_{xyz} = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2} \tag{3.1}$$

Zaimplementowany w ramach pracy magisterskiej algorytm wykrywania i zliczania kroków bazuje na założeniu iż sumaryczna wartość przyspieszenia wykrywanego przez akcelerometr w trakcie chodzenia oscyluje wokół wartości 1G[8]. Podstawową zasadą działania algorytmu jest więc wykrywanie cykli w jakich następuje kolejno spadek wartości odczytywanego przyspieszenia, a następnie jego wzrost ponad wartość spoczynkową. Do poprawnego działania algorytmu konieczne jest dobranie nie tylko odpowiedniej stałej czasowej w której wykryty cykl będzie zliczany jako krok, ale również ustalenie progów przyspieszenia poniżej i powyżej wartości spoczynkowej które będą traktowane jako odpowiednio początek i koniec cyklu.

3.2.3. Implementacja algorytmu

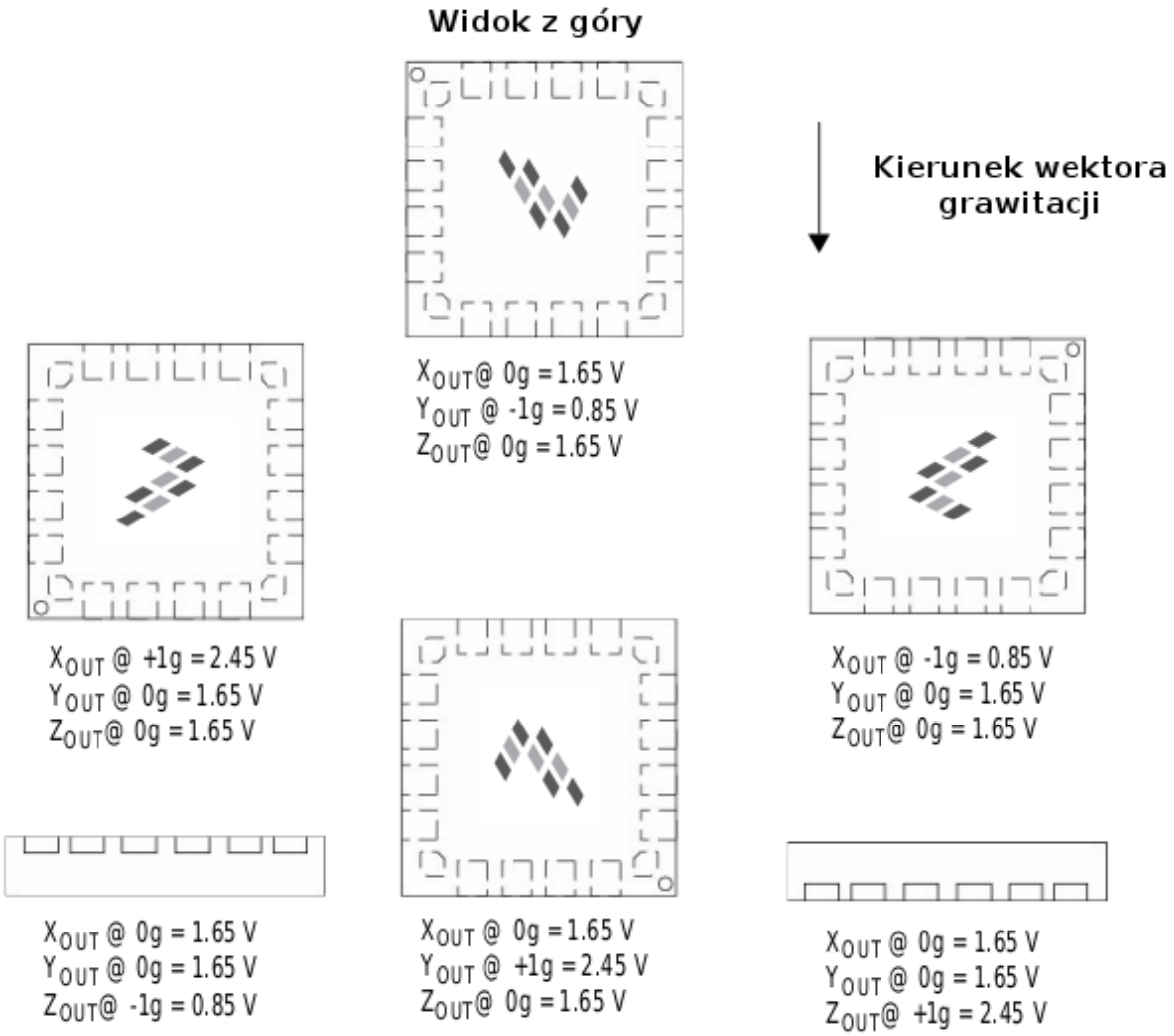
Do praktycznej implementacji algorytmu wykrywania kroków wykorzystany został moduł akcelerometru trójosiowego firmy Freescale Semiconductor. Czujnik MMA7260 [9] został zainstalowany w robocie jako element modułu MOBOT-GM3A (rys. 3.7). Moduł akcelerometru jest to jedyny moduł w całości dostarczony przez zewnętrznego dostawcę, a wybór takiej strategii podyktowany był jedynie względami ekonomicznymi i dostępnością tego typu urządzeń na polskim rynku elektronicznym. Na ilustracji zamieszczone zostało zdjęcie gotowego układu wraz z zaznaczonymi wyprowadzeniami które zostały wykorzystane w ramach pracy magisterskiej.



Rysunek 3.7: Moduł akcelerometru trójosiowego z czujnikiem przyspieszenia MMA7260

[8] 1G - Jednokrotność przyspieszenia ziemskiego wynoszącego w przybliżeniu 9.81 $\frac{m}{s^2}$

Wykorzystany czujnik przyspieszenia oferuje cztery zakresy czułości których zmiana odbywa się poprzez konfigurację odpowiednich zworek w module MOBOT - GM3A. W ramach implementacji wybrany został zakres czułości od $-1.5g$ do $1.5g$ gdyż udziela on najdokładniejszej informacji na temat przyspieszenia w przedziale w jakim mieszczą się przyspieszenia związane z chodzeniem. W wybranym trybie czułości akcelerometr wykazuje czułość rzędu 800mV/g, co przy wartości przyspieszenia równej zero daje środek przedziału czułości na poziomie 1.65V. Ze względu na analogową charakterystykę sygnału wyjściowego z modułu akcelerometru konieczne jest wykorzystanie konwertera analogowo-cyfrowego w celu odczytania danych pomiarowych otrzymywanych na wyjściach urządzenia. Szczegółowy opis działania wbudowanego w robot ADC[9] można znaleźć w rozdziale poświęconym dalmierzom IR.



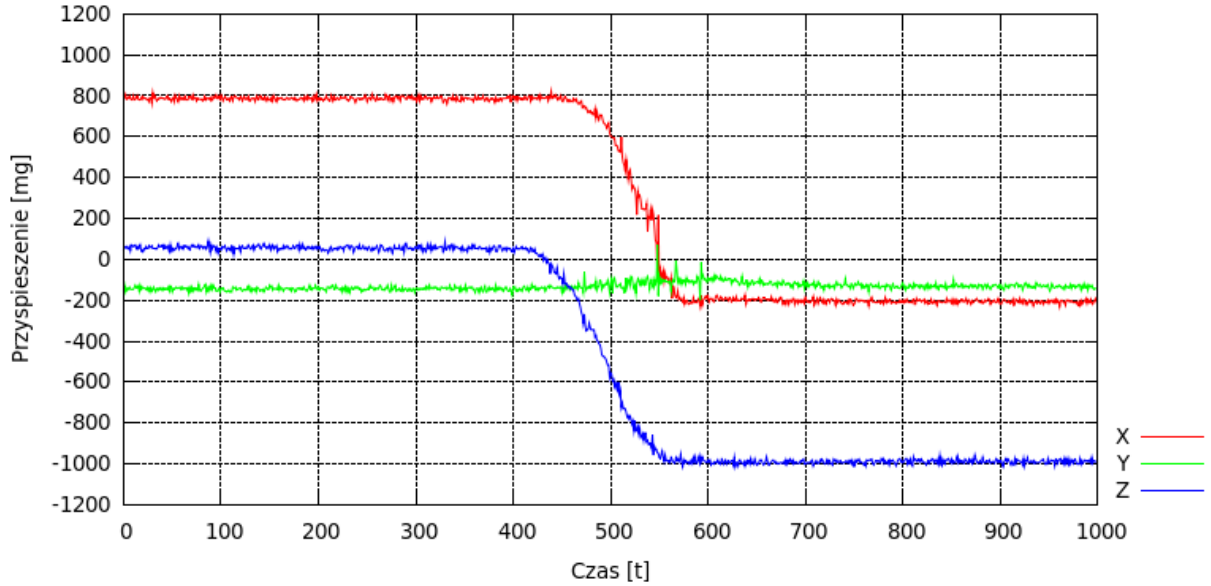
Rysunek 3.8: Wartości przyspieszenia statycznego dla układu MMA7260 dostarczone w ramach specyfikacji technicznej urządzenia [9]

[9] Analog to Digital Converter

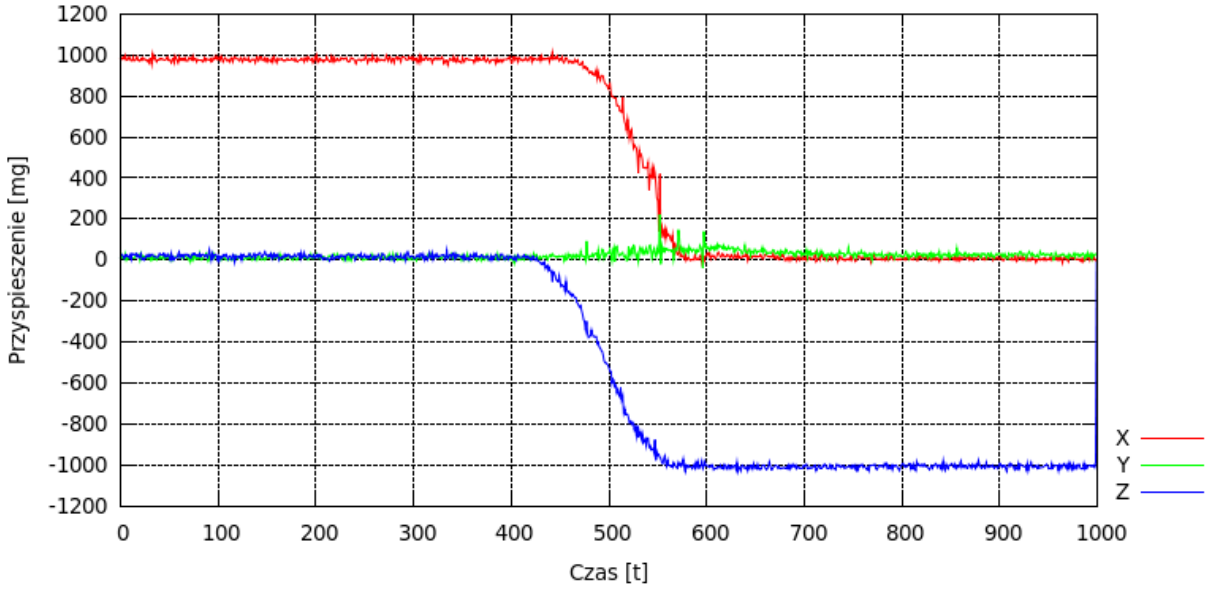
Przed rozpoczęciem praktycznej implementacji algorytmu wykrywania kroków konieczne okazało się przeprowadzenie kalibracji modułu, gdyż już wstępne testy czujnika zwróciły spore różnice pomiędzy wartościami podanymi w nocie katalogowej, a stanem zmierzonym na wyjściach akcelerometru. Zgodnie z rysunkiem 3.8 znalezionym w dokumentacji udostępnionej przez producenta czułość oraz zakres napięć na poszczególnych osiach powinien być w stałych, jednakowych przedziałach. Niestety, jak się okazało, poszczególne osie miały bardzo znaczące różnice w zakresie swojej czułości co mogłoby bardzo niekorzystnie wpłynąć na stabilność działania zaproponowanego algorytmu zliczania kroków.

Wspomniany proces kalibracji polegał na ustawieniu czujnika na płaskiej, poziomej powierzchni oraz odczytaniu wartości przyspieszenia statycznego dla każdej z osi z osobna. Procedurę taką powtarzano wielokrotnie zmieniając orientację modułu względem wektora grawitacji. Po zebraniu wszystkich danych pomiarowych dla każdej osi z osobna, została wyznaczona rzeczywista czułość, a następnie w module odpowiedzialnym za akwizycję danych o przyspieszeniu zostały zaprogramowane funkcje korygujące otrzymane dane wejściowe o wartości wyliczone podczas kalibracji urządzenia.

Na wykresach 3.9 oraz 3.10 znajdują się informacje o stanie wyjść przed i po przeprowadzeniu kalibracji. Wykresy przedstawiają przyspieszenie statyczne jakie ustaliło się na poszczególnych osiach akcelerometru w stanie spoczynku jak również zmianę tego przyspieszenia w chwili ustalania nowej orientacji czujnika. Wartość przyspieszenia statycznego została wyznaczona na podstawie napięcia odczytanego za pomocą konwertera ADC z odpowiednich wyjść akcelerometru. Jak można wnioskować na podstawie rysunku 3.8 oraz wartości przyspieszeń na osiach X, Y, Z prezentowanych na wykresach 3.9, 3.10 w chwili czasowej 0 akcelerometr był ustawiony bokiem z a zwrot i kierunek osi X był zgody z kierunkiem wektora grawitacji. W chwili czasowej 400 rozpoczyna się obrót akcelerometru o 90 ° wokół osi Y, aż do ustalenia się nowej orientacji czujnika w chwili czasowej 600. Oczekiwaną wartością wypadkową przyspieszeń na poszczególnych osiach w chwili gdy akcelerometr znajduje się w stanie spoczynku jest wartość przyspiesznia w przybliżeniu równą 1000 mg, podczas gdy wartości przed kalibracją, widoczne na wykresie 3.9, w pierwszej pozycji ustalonej dają przyspieszenie wypadkowe rzędu 700 mg natomiast w drugiej pozycji ustalonej -1300 mg. Jak można zaobserwować rozbieżności pomiędzy odczytami bez i z kalibracją są dosyć znaczące, a dla rozważanego sposobu zastosowania wręcz krytyczne. Dlatego też istotne jest aby przed uruchomieniem algorytmu rozpoznawania kroków mieć pewność, że czujnik jest w prawidłowy sposób skalibrowany.



Rysunek 3.9: Wartości przyspieszenia odczytywane na wyjściach akcelerometru przed przeprowadzeniem kalibracji



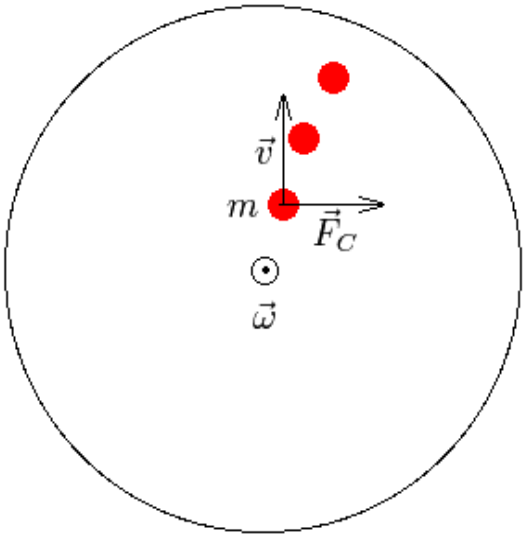
Rysunek 3.10: Wartości odczytywane na wyjściach akcelerometru po przeprowadzeniu kalibracji

3.3. Żyroskop

Do budowy inercjalnej jednostki pomiarowej został użyty żyroskop trójosiowy wykonany w technologii MEMS. Żyroskopy wyprodukowane w tej technologii są wykorzystywane np.: w stabilizatorach obrazu do kamer lub kontrolerach do gier wideo. Ich główną zaletą jest mały rozmiar. W poniższym podrozdziale zostanie przedstawiona zasada działania takiego żyroskopu oraz sposób wykorzystania go w robocie mobilnym.

3.3.1. Zasada działania

Żyroskopy MEMS korzystają z pozornej siły Coriolis'a (rys. 3.11) do pomiaru prędkości kątowej z jaką obraca się ciało.

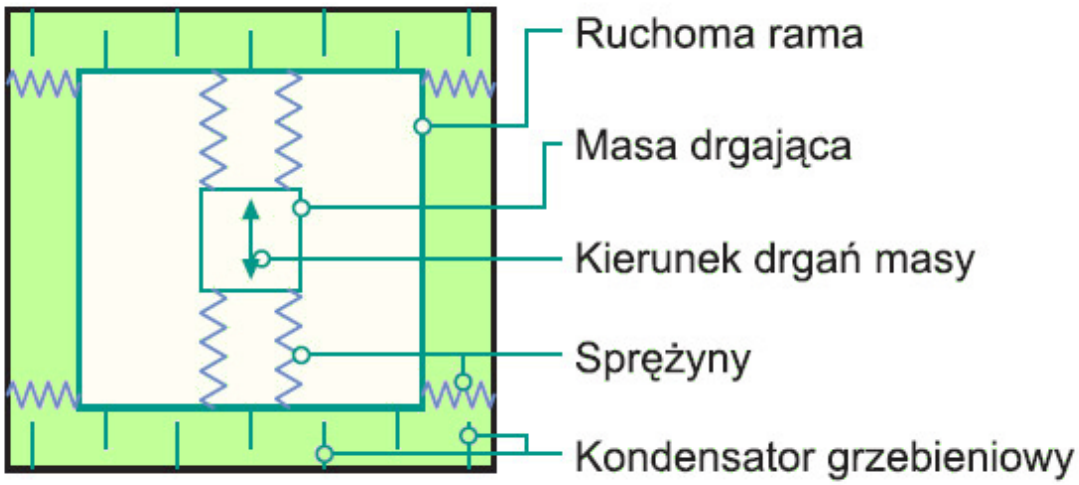


Rysunek 3.11: Siła Coriolis'a

Załóżmy, że ciało o masie m porusza się z prędkością $\vec{v}$ oraz układ w którym przemieszcza się to ciało obraca się z prędkością kątową $\vec{\omega}$. W takich warunkach ciało o którym mowa zostanie odchylone od kierunku przemieszczania się wyznaczonego przez wektor prędkości $\vec{v}$. Odchylenie to będzie spowodowane siłą Coriolis'a $\vec{F_c}$ którą można opisać przy pomocy wzoru 3.2. Żyroskopy MEMS wykorzystują to zjawisko określając przemieszczenie drgającego ciała, które jest tak naprawdę jedną z okładek kondensatora, jako zmianę pojemności.

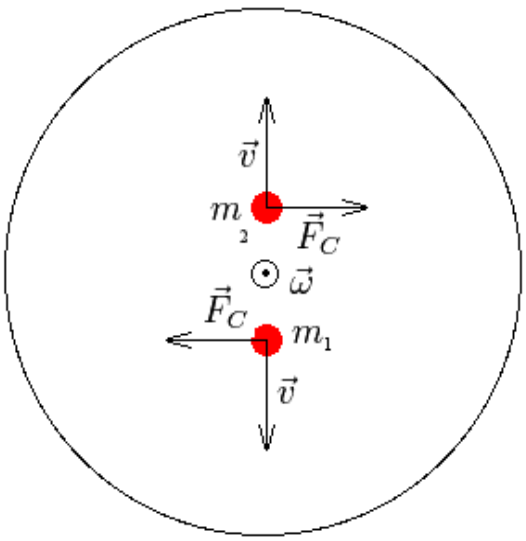
$$\vec{F_c} = -2m\,(\vec{\omega} \times \vec{v}) \tag{3.2}$$

Żyroskopy tego typu posiadają dwa oscylujące miniaturowe elementy które poruszają się w przeciwnych względem siebie kierunkach (rys. 3.12).



Rysunek 3.12: Budowa elementu pomiarowego żyroskopu [10]

Jeżeli urządzenie do którego przymocowany jest żyroskop zacznie się obracać, spowoduje to wychylenie się oscylujących elementów w przeciwnych kierunkach (rys. 3.13). Wychylenie to z kolei powoduje zmianę pojemności, a różnica pomiędzy pojemnościami zmierzonymi przy pomocy obydwu ruchomych elementów jest proporcjonalna do prędkości kątowej. W ten sposób zmierzona szybkość obrotu jest następnie reprezentowana jako wynik analogowy (napięcie) lub cyfrowy (wartość liczbowa).



Rysunek 3.13: Dwie oscylujące masy odchylane przez siłę Coriolis'a

Wynik działania żyroskopu MEMS jest obojętny na zmianę przyspieszenia liniowego, w szczególności na przyspieszenie ziemskie. Przyspieszenie liniowe wywoła wychylenie się jednocześnie obydwu elementów oscylujących w tym samym kierunku. W ten sposób nie będzie wykryta żadna różnica pojemności. Prędkość kątowa wskazywana przez żyroskop nie ulegnie zmianie. Dzięki temu żyroskopy MEMS są odporne na wstrząsy, uderzenia oraz wibracje.

Podstawowe wielkości charakteryzujące żyroskop:

Zakres $[dps^{10}]$ Definiuje wartość maksymalną i minimalną prędkości kątowej jaką żyroskop jest w stanie zmierzyć. Często żyroskop ma kilka zakresów z których możemy wybrać jeden odpowiadający naszym potrzebom.

Czułość $[\frac{mdps}{digit}]$ Wielkość określająca wartość minimalną prędkości kątowej jaką żyroskop może zmierzyć.

Zmiana czułości względem temperatury $[\%]$
Określa jak bardzo pomiary urządzenia są podatne na zmianę temperatury.

Zkres poziomu zero $[dps]$
Definiuje zakres w jakim pomiary mogą się wahać w chwili gdy żyroskop pozostaje w spoczynku.

Zmiana zakresu poziomu zero względem temperatury $[\frac{dps}{^\circ C}]$
Określa zmiany pomiarów żyroskopu pozostającego w spoczynku względem temperatury.

Nieliniowość $[\%FS]$ Parametr określa maksymalne procentowe odchylenie wartości na wyjściu żyroskopu od dopasowanej do nich linii prostej.

Przepustowość $[Hz]$ Definiuje częstotliwość z jaką żyroskop może wykonywać kolejne pomiary.

3.3.2. Kalibracja i odczytywanie wyników

Żyroskopy są zazwyczaj fabrycznie testowane i kalibrowane pod kątem zakresów pomiarów poziomu zerowego oraz czułości. Jednak po umieszczeniu elementu na płytce PCB[11] zostaje on poddany naprężeniom przez co może być potrzebna dodatkowa kalibracja układu. Wartości wyjściowe otrzymywane w wyniku pomiaru przeprowadzonego za pomocą żyroskopu można przedstawić za pomocą równania 3.3:

$$\omega_t = \omega_m - \omega_0 \tag{3.3}$$

gdzie:

ω_t – rzeczywista wartość prędkości kątowej,
ω_m – pomiar z żyroskopu,
ω_0 – wartość zerowa reprezentująca brak ruchu,

W celu kompensacji niestabilności żyroskopu należy pozostawić urządzenie nieruchome i wykonać około tysiąc pomiarów. Następnie konieczne jest obliczenie wartości średniej z wykonanych pomiarów. W ten sposób określimy średnie wychylenie od wartości zerowej żyroskopu, czyli nasze ω_0.

Podczas używania żyroskopu do pomiaru bardzo małych prędkości kątowych należy jeszcze wziąć pod uwagę minimalne odchylenia wartości w zależności od zmiany temperatury. Prędkość kątową opisujemy wzorem 3.4:

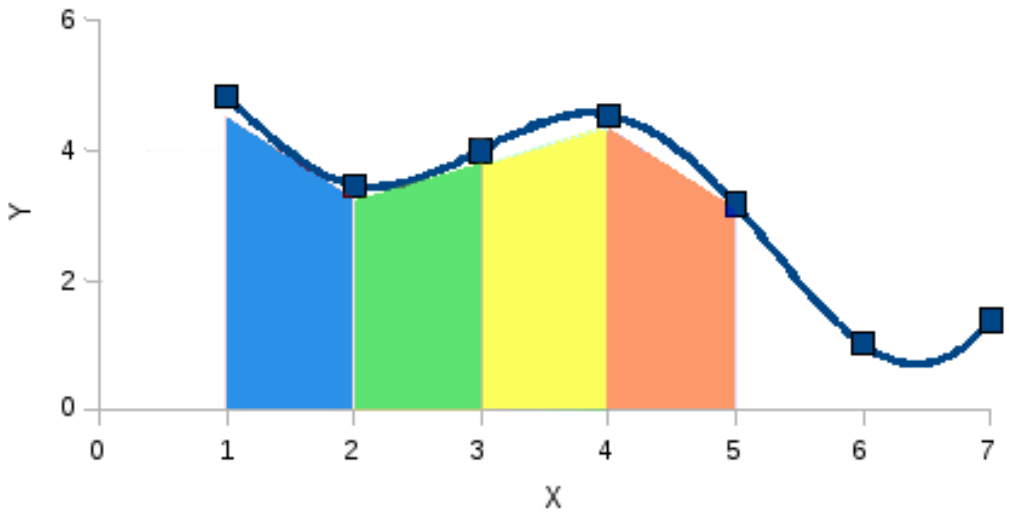
$$\omega = \frac{d\varphi}{dt} \tag{3.4}$$

Wzór na kąt o jaki zostało obrócone ciało poruszające się z prędkością kątową ω (równanie 3.5) wyprowadzamy całkując obustronnie równanie 3.4.

$$\varphi = \int \omega dt \tag{3.5}$$

Do obliczenia kąta obrotu można wykorzystać jedną z numerycznych metod całkowania, np. metodę trapezów (rys. 3.14). Metoda ta polega na podzieleniu pola pod wykresem całkowanej funkcji na wąskie trapezy, a następnie zsumowanie pól tych trapezów.

[11] PCB (z ang. Printed Circuit Board) - płyta obwodu drukowanego



Rysunek 3.14: Całkowanie przy pomocy metody trapezów[12]

Po wykorzystaniu metody trapezów, wzór na kąt o jaki żyroskop został obrócony w czasie pomiędzy dwoma pomiarami, można zapisać w postaci równania 3.6:

$$\varphi = C_f \cdot \frac{\omega_{i-1} + \omega_i}{2} \cdot \Delta t \tag{3.6}$$

gdzie:

C_f – współczynnik korygujący,

Δt – czas jaki upłynął pomiędzy dwoma pomiarami,

ω_{i-1} – szybkość kątowa odczytana z poprzedniego pomiaru,

ω_i – szybkość kątowa odczytana z obecnego pomiaru,

W taki sposób otrzymujemy wzór 3.7, na kąt o jaki ciało zostało obrócone w czasie Δt. Kąt całkowity pomiędzy orientacją początkowa a orientacją końcową ciała jest sumą wszystkich wykonanych pomiarów kątów.

$$\varphi_c = \varphi_0 + \sum_{i=1}^{n} \varphi_i \tag{3.7}$$

gdzie:

φ_0 – kąt początkowy,

φ_i – wartość kąta z pojedynczego pomiaru

[12] źródło: http://mateusz-lach.blogspot.com/2010/09/cakowanie-numeryczne.html

Do otrzymania jak najlepszych wyników konieczne jest określenie współczynnika korygującego pomiar żyroskopu. W tym celu porównujemy wyniki pomiarów żyroskopu z pomiarami innych przyrządów np. magnetometru wykorzystując wzór:

$$C_f = \frac{\varphi_o}{\varphi_g} \tag{3.8}$$

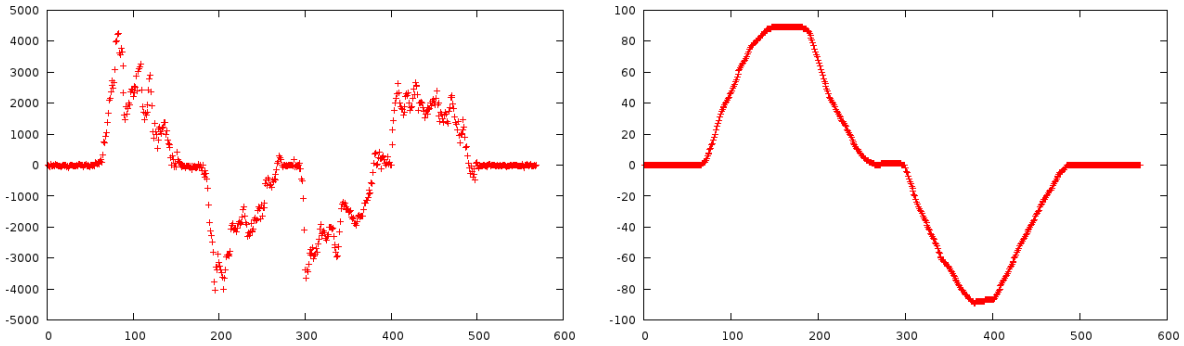
gdzie:

C_f – współczynnik korygujący,

φ_o – kąt obliczony przy pomocy urządzenia zewnętrznego,

φ_g – kąt wyznaczony przez żyroskop,

Niestety z powodu ciągle nakładających się błędów całkowania oraz niedoskonałości samego żyroskopu, wskazywany kąt odchylenia od ułożenia początkowego będzie się oddalał od wartości rzeczywistej wraz z upływem czasu. W przypadku tej pracy dyplomowej nie jest jednak potrzebna bardzo wysoka stabilność i precyzja wartości mierzonych przez żyroskop dla długich okresów pomiarowych.



(a) Zmiany wartości rejestrów podczas obrotu (b) Zmiany kąta wychylenia z pozycji startowej

Rysunek 3.15: Dane o obrocie zarejestrowane za pomocą modułu żyroskopu

Rysunek 3.15 przedstawiają informacje o obrocie robota uzyskane za pomocą żyroskopu. Robot został najpierw obrócony o 90° w lewo po czym przekręcono go do położenia pierwotnego, następnie obrócono go o 90° w prawo i ponownie przywrócono jego pierwotne ustawienie. Na rysunku 3.15a zostały przedstawione kolejne wartości rejestrów żyroskopu odzwierciedlających prędkość kątową z jaką poruszał się czujnik wokół osi Z podczas wykonywania pomiarów. Wykres przedstawiony na rysunku 3.15b, pokazuje efekt działania algorytmu odpowiedzialnego za całkowanie pomiarów z żyroskopu i przeliczanie ich na wartość kąta. Na osi Y wykresu widzimy wartość kąta obrotu, natomiast oś X określa numer kolejnego pomiaru.

3.3.3. Opis zastosowanego elementu i budowy modułu

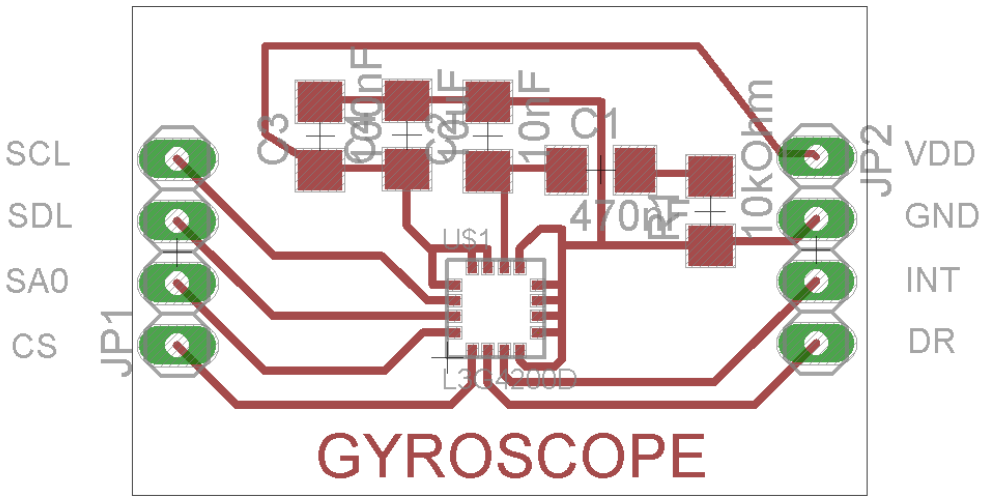
W robocie został użyty ultra-stabilny trójosiowy żyroskop cyfrowy L3G4200D firmy STMicroelectronics. Jego główną zaletą jest bardzo dobra jakość pomiarów osiągnięta dzięki zastosowaniu innowacyjnej struktury mechanicznej. Zazwyczaj urządzenia tego typu stosują dwie lub trzy struktury odpowiedzialne za wykonywanie pomiarów. Żyroskop firmy STMicroelectronics natomiast posiada jeden taki element odpowiedzialny za pomiar na wszystkich osiach. Takie rozwiązanie eliminuje zakłócenia pomiędzy osiami, powodowane przez same elementy pomiarowe.

Zgodnie z notą katalogową [11] dostępną na stronie internetowej producenta żyroskop L3G4200D zamontowany w robocie posiada charakterystykę mechaniczną taką jak zaprezentowano w tabeli 3.1:

Tabela 3.1: Charakterystyka mechaniczna żyroskopu L3G4200D dla napięcia zasilania $3.0V$ i temperatury pracy $25°C$

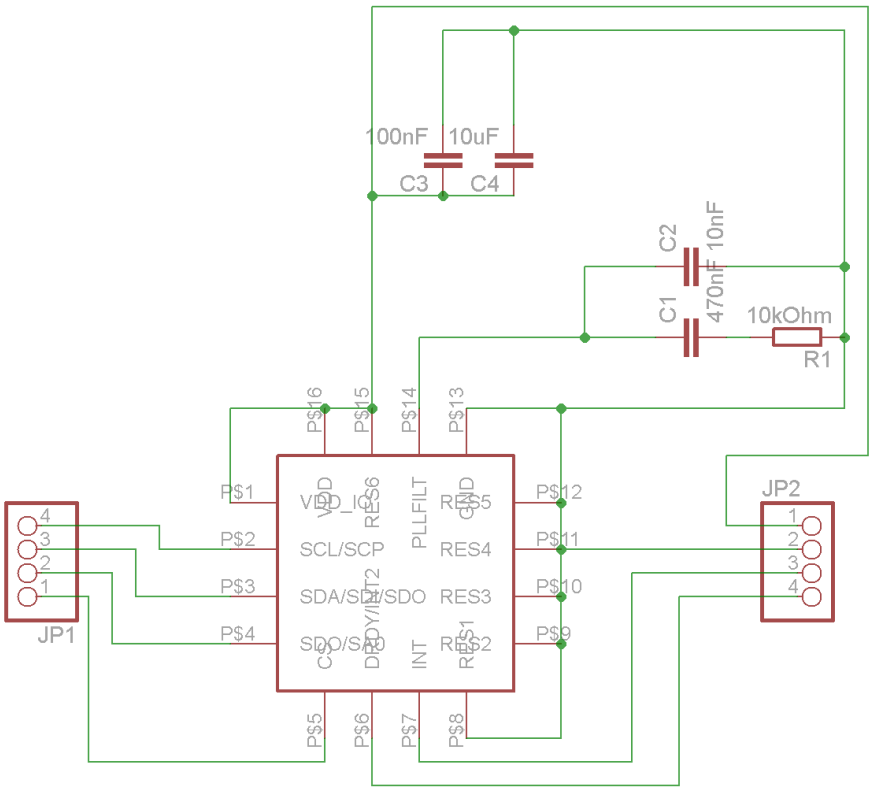
Symbol	Parametr	Warunki testowe	Wartość	Jednostka
			±250	
FS	Skala		±500	dps
			±2000	
		$FS = 250dps$	8.75	
So	Czułość	$FS = 500dps$	17.50	$\frac{mdps}{digit}$
		$FS = 2000dps$	70	
SoDr	Zmiana czułości	$-40°C$ do $+85°C$	±2	%
	względem temperatury			
		$FS = 250dps$	±10	
DVoff	Zakres poziomu zero	$FS = 500dps$	±15	dps
		$FS = 2000dps$	±75	
OffDr	Zmiana zakresu poziomu	$FS = 250dps$	±0.03	dps
	zero względem temperatury	$FS = 2000dps$	±0.04	
NL	Nieliniowość		±0.2	% FS
Rn	Współczynnik szumów	$BW = 50Hz$	±0.03	$\frac{dps}{\sqrt{Hz}}$
ODR	Częstotliwość odświeżania		100/200	Hz
	danych na wyjściu		400/800	
Top	Zakres temperatur		od $-40°C$	$°C$
	operacyjnych		do $+85°C$	

Aby możliwe stało się podłączenie żyroskopu do robota, została zaprojektowana w programie Eagle specjalna płytka podłączeniowa widoczna na rysunku 3.16.



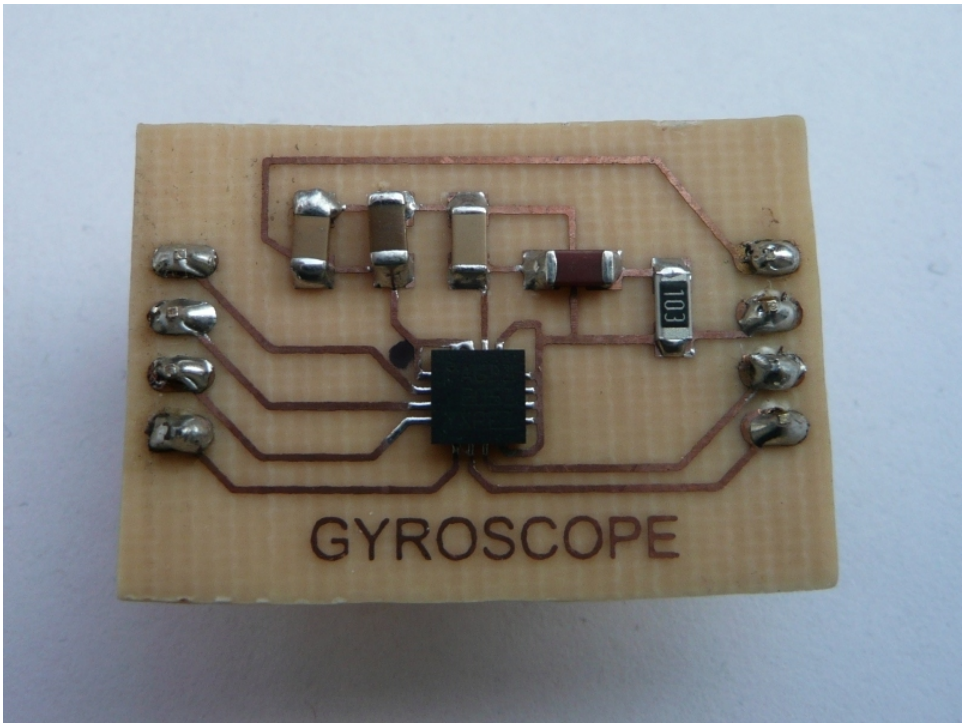
Rysunek 3.16: Płytka PCB modułu żyroskopu trójosiowego

Schemat układu umożliwiającego podłączenie czujnika został przedstawiony na rysunku 3.17. Jego projekt został stworzony na podstawie wspomnianej wcześniej noty katalogowej.



Rysunek 3.17: Schemat płytki PCB modułu żyroskopu

Płytka modułu została zaprojektowana z wykorzystaniem elementów SMD[13] w obudowie 1206. Rozmiar obudowy tych elementów został wymuszony przez brak kondensatorów ceramicznych o pojemności $10\mu F$ w obudowie mniejszej. Cały proces tworzenia płytki modułu żyroskopu był przeprowadzony metodami domowymi ze względu na niski koszt oraz brak konieczności czekania kilku tygodni na wykonanie takiej płytki przez specjalistyczną firmę. Szczegółowy opis procesu tworzenia płytki metodami domowymi można znaleźć w dodatku B. Poniżej zamieszczone zostało zdjęcie 3.18 przedstawiające gotowy moduł żyroskopu oraz tabela 3.2 omawiająca szczegółowo wyprowadzenia modułu umożliwiające jego podłączenie do robota.



Rysunek 3.18: Gotowy moduł żyroskopu

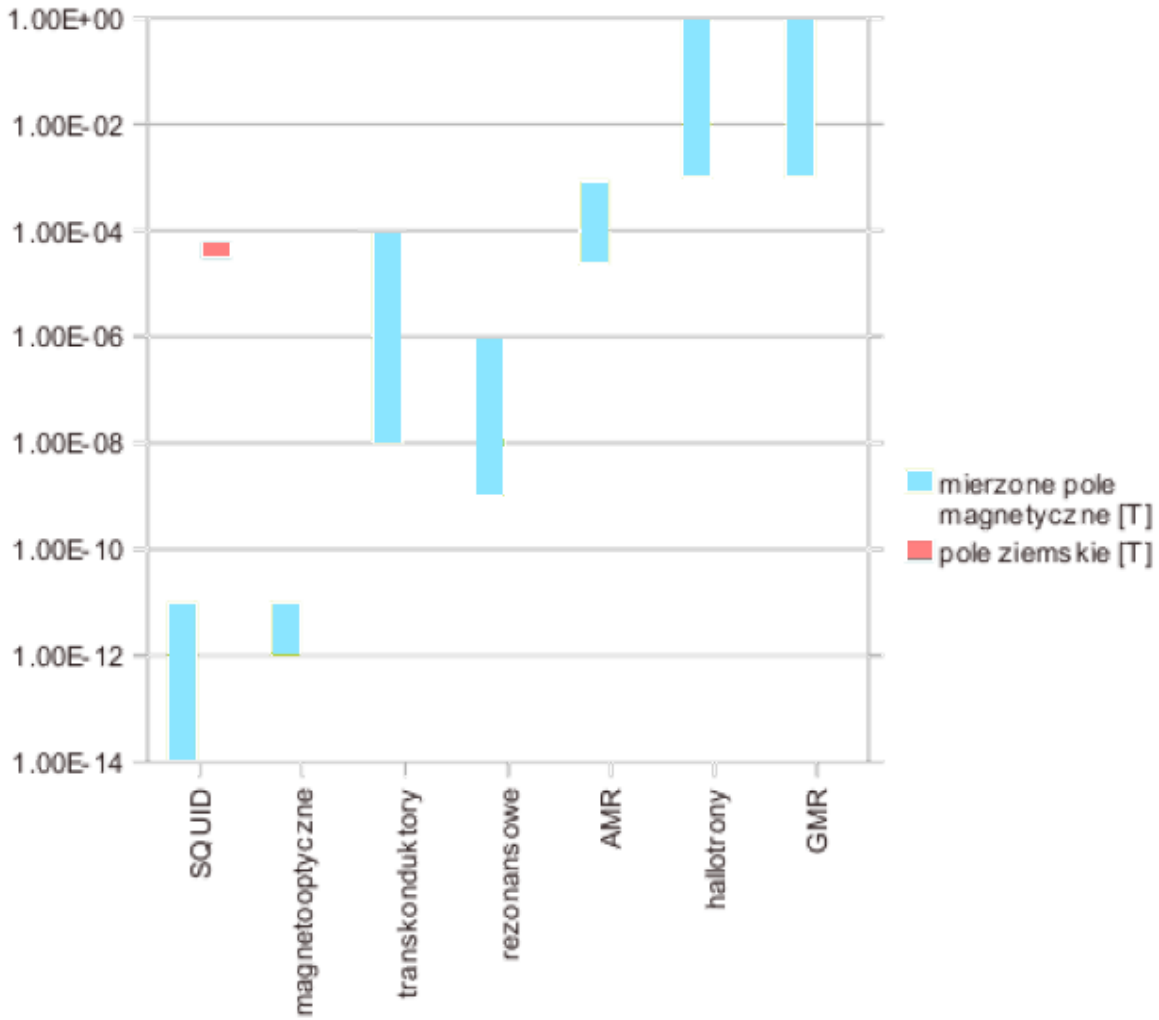
Tabela 3.2: Opis wyprowadzeń płytki modułowej żyroskopu

Nazwa	Typ	Opis
VDD	Zasilanie	Napięcie zasilania $3.3V$
GND	Zasilanie	Masa
INT	Wyjście	Konfigurowalny pin przerywania
DR	Wyjście	Pin informujący o nowych danych do pobrania
SCL	Wejście	Zegar sterujący I^2C
SDL	We / Wy	Szyna do przesyłania danych I^2C
SA0	Wejście	Najmniej znaczący bit adresu urządzenia
CS	Wejście	Przełącznik trybu działania I^2C lub SPI

[13] SMD – Surface Mount Device

3.4. Magnetometr

Jednym z elementów większości Inercjalnych Systemów Nawigacyjnych jest magnetometr. Wykorzystuje się go do określenia kierunku w którym porusza się dane ciało. Typów czujników mierzących pole magnetyczne jest wiele. Dla INS istotny jest jedynie czujnik, będący w stanie wykonać pomiary w zakresie indukcji pola magnetycznego ziemi (rys. 3.19), która wynosi od $30\mu T$ do $60\mu T$.



Rysunek 3.19: Zakresy czujników pola magnetycznego różnego typu [12]

Na rysunku 3.19 przedstawione są zakresy pomiarowe czujników pola magnetycznego różnego typu. Jak widać tylko dwa rodzaje magnetometrów z wyżej wymienionych działają w przedziale obejmującym zakres indukcji pola magnetycznego ziemi. Są to magnetometry transduktorowe oraz AMR[14]. Z powodu małych rozmiarów, przystępnej ceny i dostępności, w tej pracy wykorzystany został magnetometr AMR.

[14] AMR - Anisotropic Magneto Resistance, anizotropowy magnetoopór

3.4.1. Zasada działania

Czujnik AMR wykorzystuje zjawisko anizotropowego magnetooporu w celu pomiaru indukcji pola magnetycznego. Zjawisko to objawia się zmianą rezystancji materiału pod wpływem zmiany orientacji działającego pola magnetycznego względem kierunku prądu płynącego przez ten materiał.

Czujniki tego typu charakteryzują się wysoką dokładnością i szybkością działania jednocześnie pobierając mniejszy prąd w stosunku do alternatywnych technologii.

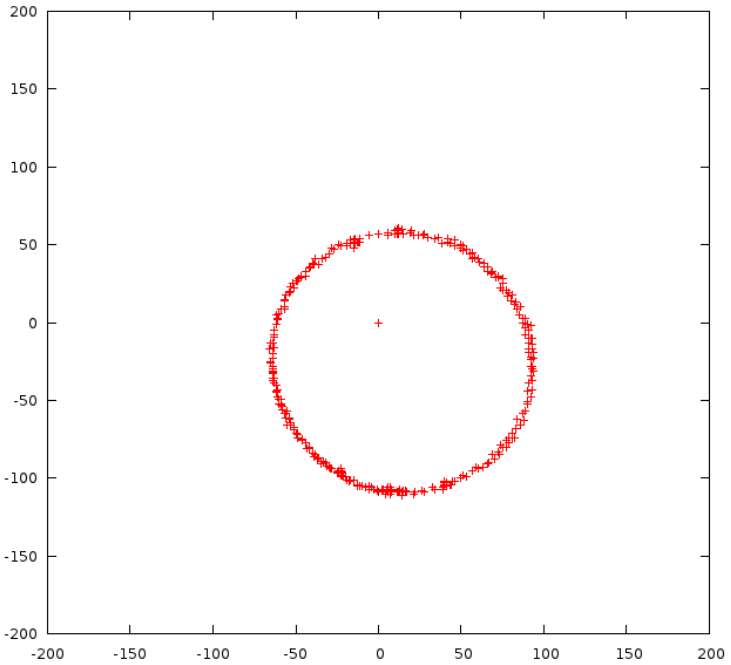
3.4.2. Opis elementu

W celu stworzenia modułu kompasu został wykorzystany magnetometr dwuosiowy MMC2120 firmy Memsic. Parametry techniczne zastosowanego magnetometru uzyskane na podstawie noty katalogowej producenta [13] zostały podsumowane w ramach tabeli 3.3. Dane z magnetometru są przesyłane przy pomocy interfejsu I^2C w postaci dwóch 12-bitowych liczb. Jedna z nich określa wartość indukcji pola magnetycznego zmierzonego na osi X druga zaś na osi Y.

Tabela 3.3: Parametry magnetometru MMC2120

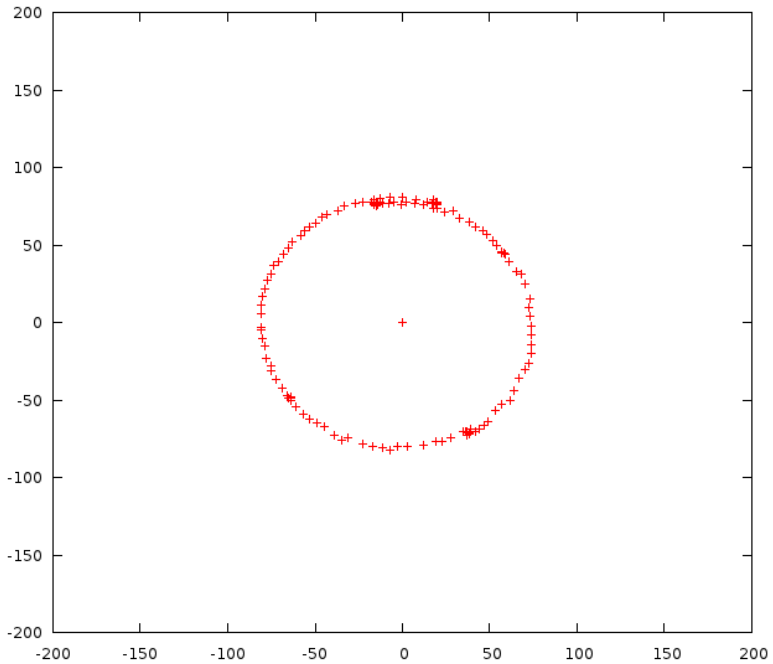
Parametr	Warunki testowe	Wartość	Jednostka
Zakres pomiarów	Wypadkowe pole magnetyczne	od -2 do 2	gauss
Nieliniowość	± 1 gauss	0.1	$\%FS$
	± 2 gauss	0.5	$\%FS$
Dokładność		od ± 2 do ± 5	deg
Czułość		od $\frac{1}{461}$ do $\frac{1}{563}$	gauss
Wartość na wyjściu przy braku pola magnetycznego		od -0.2 do 0.2	gauss
		2048	wartość zerowa

Dla poprawnego działania kompasu niezbędna jest kalibracja magnetometru. Przeprowadzamy ją poprzez obracanie magnetometru we wszystkich możliwych stopniach swobody jednocześnie zapisując wartość maksymalną i minimalną wskazywaną zarówno na osi X jak i Y. Dzięki tym wartością będziemy w stanie wyznaczyć, niezależnie dla obydwu osi, przesunięcie wartości zerowej oraz ich czułość.



Rysunek 3.20: Wykres przedstawiający odczyty z rejestrów x oraz y układu MMC2120 przed kalibracją

Kalibrację należy wykonać po umieszczeniu modułu w robocie w celu wyeliminowania zakłóceń spowodowanych metalowymi elementami robota. MMC2120 jako wartość zerową przyjęta została liczba 2048. Jest to środek przedziału liczb składającego się z 12 bitów. Wartości poniżej tej liczby są uznawane za ujemne, a wartości powyżej za dodatnie.



Rysunek 3.21: Wykres przedstawiający odczyty z rejestrów x oraz y układu MMC2120 z uwzględnieniem przesunięcia wartości zerowej

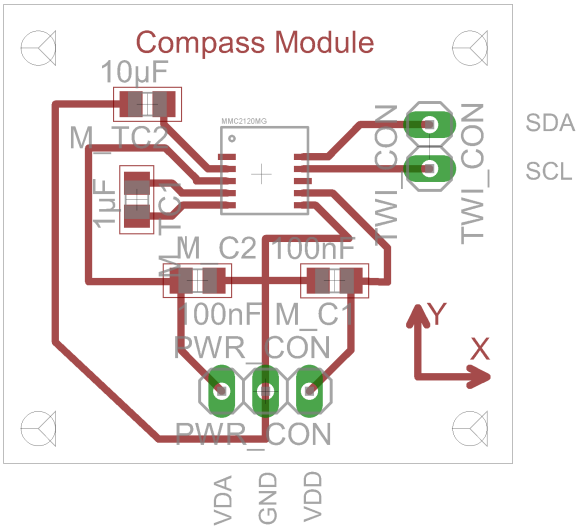
Niestety z powodu niedokładności samego układu, a także negatywnego wpływu metalowych elementów konstrukcji robota i modułu, wartość zerowa magnetometru może ulec przemieszczeniu (rys. 3.20). Dlatego też musimy obliczyć wartość odchylenia wartości zerowej obydwu osi w celu otrzymania prawidłowego wyniku. Wspomniane odchylenie wyznaczamy dla każdej osi jako wartość środkową pomiędzy maksymalnym a minimalnym odczytem podczas kalibracji. Dopiero po właściwym skalibrowaniu magnetometru (rys. 3.21) jesteśmy w stanie wykorzystać go jako kompas. W celu wyznaczenia azymutu, czyli kąta pomiędzy kierunkiem wskazywanym przez oś Y, a kierunkiem północnym, należy skorzystać z prostego wyrażenia trygonometrycznego przedstawionego na wzorze 3.9.

$$\alpha = arctg\left(\frac{x}{y}\right) \tag{3.9}$$

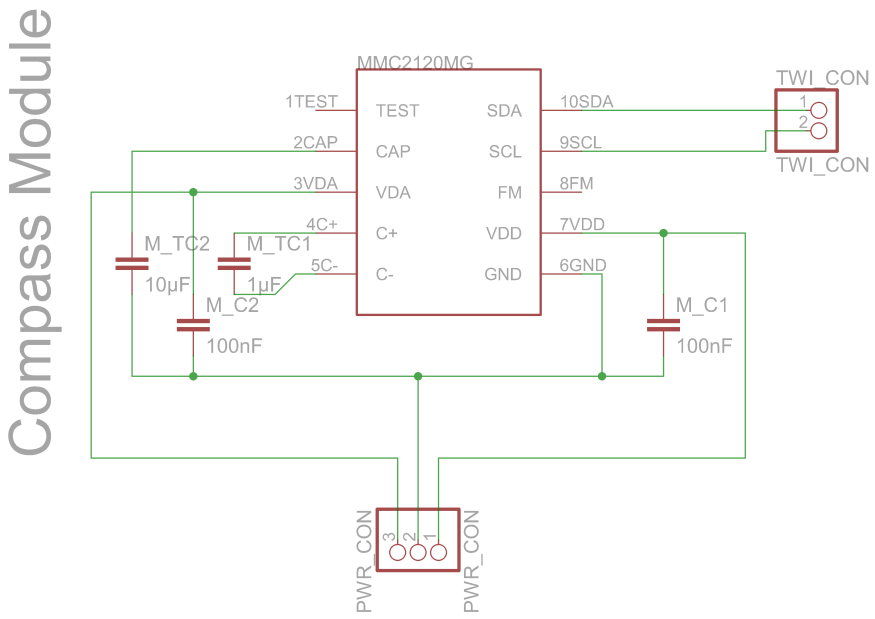
Podczas odczytu wskazań kompasu istotne jest także wzięcie pod uwagę takich czynników jak odchylenie osi X i Y czujnika od kierunku równoległego do kierunku pola magnetycznego ziemi. W przypadku gdy płaszczyzna pomiarów magnetometru nie będzie równoległa do płaszczyzny linii pola magnetycznego, pojawią się błędy związane z brakiem pomiarów magnetometru względem osi Z.

3.4.3. Budowa modułu

Płytka modułu kompasu została zaprojektowana przy pomocy darmowej wersji programu Eagle na podstawie danych zawartych w nocie katalogowej czujnika MMC2120. Na rysunkach 3.22 oraz 3.23 przedstawiono schemat oraz layout płytki modułu kompasu.



Rysunek 3.22: Projekt płytki kompasu elektronicznego

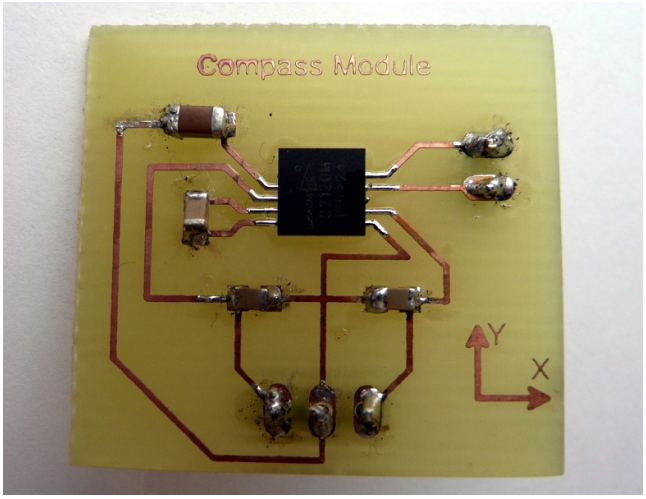


Rysunek 3.23: Schemat układu podłączeniowego dla magnetometru MMC2120

Zdjęcie 3.24 prezentuje gotowy do pracy moduł kompasu. Natomiast w tabeli 3.4 opisane są poszczególne wyjścia wyprowadzone z przygotowanego modułu.

Tabela 3.4: Opis wyprowadzeń modułu kompasu

Nazwa	Typ	Opis
VDA	Zasilanie	Napięcie zasilania $3.3V$
GND	Zasilanie	Masa
SCL	Wejście	Zegar sterujący I^2C
SDA	We / Wy	Szyna do przesyłania danych I^2C
VDD	Zasilanie	Napięcie zasilania szyny danych $3.3V$



Rysunek 3.24: Gotowy moduł kompasu

3.4.4. Nieudana próba zastosowania

Docelowo moduł kompasu miał być wykorzystywany jako element Inercjalnego Systemu Nawigacji w celu określania kierunku w którym robot się przemieszcza. Zasada działania miała być prosta. Robot miał zapamiętywać aktualny azymut z jakim się poruszał będąc na rekach operatora. Następnie po postawieniu na podłożu miał odtwarzać tor ruchu obracając wszystkie zapamiętane azymuty o 180°.

Niestety wykonane testy pokazały, iż nie jest możliwe wykorzystanie kompasu w celu określenia azymutu robota poruszającego się po podłodze budynku. Jest to spowodowane stosowaniem metalowych prętów zbrojeniowych w stropach budynków, które zakłócają pracę kompasu. Testy zostały wykonane zarówno przy pomocy modułu elektronicznego kompasu jak i tradycyjnej wojskowej busoli. Rysunki 3.25 oraz 3.26 przedstawiają odchylenie azymutów wyznaczonych na podłodze budynku.

Rysunek 3.25: Busola w położeniu początkowym

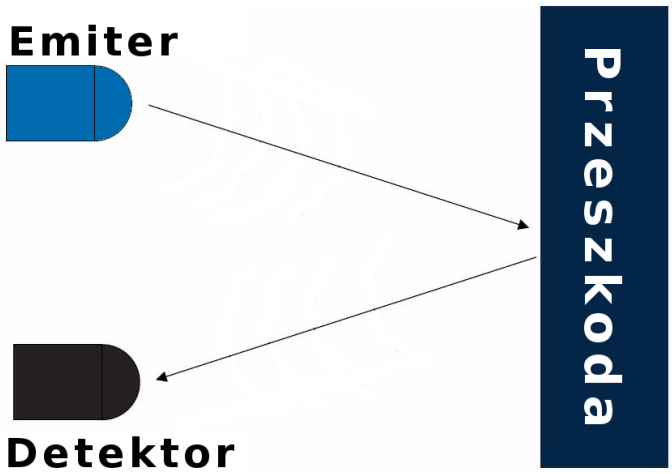
Rysunek 3.26: Busola przesunięta równolegle o 25 cm

Busola przedstawiona na zdjęciach została przesunięta równolegle o 25 cm w prawo. Wskazywany kierunek północy zmienił się o ponad 90°. Jak widać przy tak dużych wahaniach nie jest możliwe wykorzystanie magnetometru jako elementu jednoznacznie określającego kierunek w którym robot ma się poruszać. Po wielu nieudanych próbach wyeliminowania zakłóceń moduł kompasu został zastąpiony modułem żyroskopu.

3.5. Czujnik odległości

Bardzo popularnym rozwiązaniem problemu omijania przeszkód jest wykorzystanie czujników ultradźwiękowych. Dane pomiarowe pobrane w pierwszym kroku z sonaru wykorzystywane są do stworzenia lokalnej reprezentacji otoczenia pozwalającej na późniejsze sterowanie robotem [14]. Dla tak zdefiniowanego problemu możemy wyróżnić dwa rodzaje reprezentacji środowiska. Pierwszy rodzaj reprezentacji opiera się o siatkę dzięki której środowisko podzielone jest na skończoną liczbę komórek które mogą posiadać stan świadczący o pustej przestrzeni lub obecności przeszkody w danej komórce. Drugim sposobem reprezentacji jest przedstawienie otoczenia za pomocą zbioru właściwości takich jak punkty, linie oraz płaszczyzny. Wybór sposobu reprezentacji jest z reguły podyktowany rodzajem problemu jaki stawiany jest przed robotem mobilnym. W ramach pracy zrezygnowano z użycia sonaru ze względu na dużą liczbę wejść potrzebnych do jego podłączenia.

Innym rodzajem czujników bardzo dobrze sprawdzających się w rozwiązywaniu problemu omijania przeszkód są dalmierze oparte o czujnik podczerwieni. Często zdarza się, iż dalmierze IR[15] są wybierane ze względu na ich bardzo krótki czas odpowiedzi w porównaniu do czujników ultradźwiękowych. Niestety jakość pomiaru w przypadku czujników podczerwieni jest w dużym stopniu zależna od współczynnika odbicia powierzchni przeszkody, jak również odległości od przeszkody oraz położenia nadajnika i odbiornika w stosunku do powierzchni przeszkody. Zdarza się więc, że brak precyzyjnych danych o lokalizacji przeszkody i jej właściwościach wykluczają dalmierze IR z niektórych zastosowań. Jednakże szeroka dostępność tego typu urządzeń oraz łatwość ich wykorzystania spowodowała, że są one najczęściej stosowane w robotach zajmujących się podążaniem za ścianą czy omijaniem i liczeniem przeszkód.



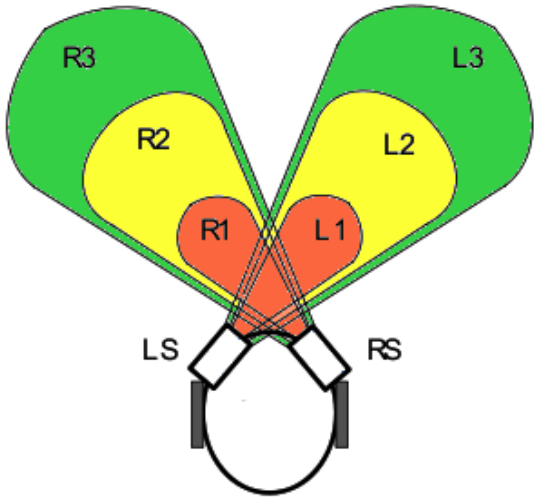
Rysunek 3.27: Zasada działania dalmierza wykorzystującego podczerwień

[15] IR (ang. infrared) - skrótowe oznaczenie urządzeń wykorzystujących podczerwień

Typowy dalmierz IR składa się z dwóch elementów. Pierwszym z nich jest dioda emitująca promieniowanie podczerwone, natomiast drugim elementem jest detektor, najczęściej fotodioda lub fototranzystor (rys. 3.27). Zasada działania tak zbudowanego czujnika odległości opiera się o zjawisko odbicia. Światło wyemitowane przez emiter odbija się od przeszkody i trafia do detektora. Niestety pomiar tego rodzaju jest niezwykle wrażliwy na zmiany w oświetleniu oraz rodzaj i kształt napotkanej przeszkody.

3.5.1. Algorytm omijania przeszkód

Algorytm sterujący ruchami robota bazuje w całości na danych pomiarowych otrzymanych z czujników robota. Opisane podejście jest adaptacją metodologii opisanej w ramach pracy „Obstacles Avoidance Method for an Autonomous Mobile Robot" [14]. Zaproponowane rozwiązanie bazuje na prostej maszynie stanów w której przejścia pomiędzy kolejnymi stanami odbywają się poprzez zmianę poziomów na czujnikach odległości. Autorzy proponują podłączenie do robota dwóch dalmierzy, jednego po lewej (LS), a drugiego po prawej stronie (RS) robota w taki sposób aby przed robotem nie było martwego punktu. Sposób montażu czujników został przedstawiony na rysunku 3.28.



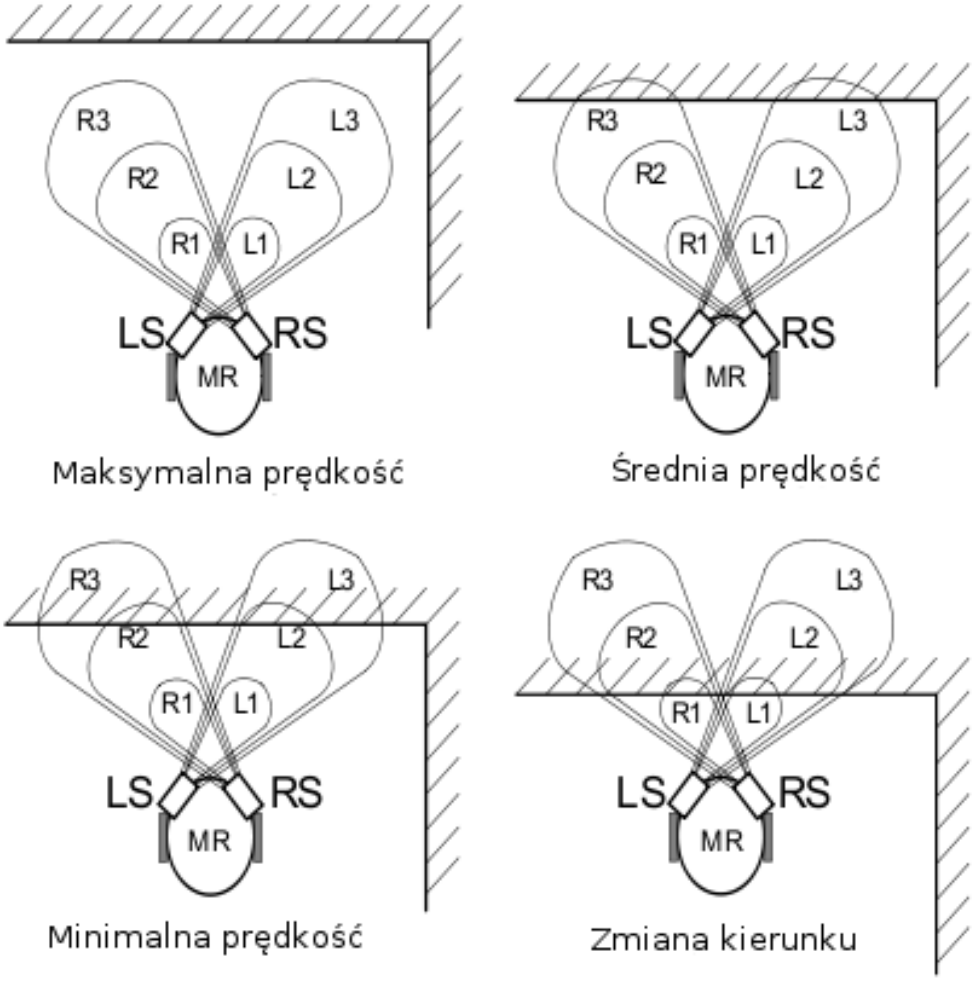
Rysunek 3.28: Sposób montażu czujników podczerwieni na robocie [14]

Każdy z czujników składa się z nadajnika i odbiornika podczerwieni, a ich przedział czułości, zgodnie z zaleceniami autorów, podzielony został na trzy poziomy oznaczone symbolami L1, L2, L3 dla czujnika lewego i odpowiednio R1, R2, R3 dla czujnika prawego. Istotne jest aby kolejne poziomy ułożone były w porządku rosnącym, a przerwa pomiędzy kolejnymi była na tyle szeroka, aby zminimalizować wpływ niedokładności pomiaru odległości. W chwili gdy robot znajduje się w ruchu na każdym z czujników jesteśmy w stanie

odczytać wartość odległości od przeszkody. Dzięki takiej informacji możemy jednoznacznie kontrolować zachowanie robota w zależności od poziomu jaki w danej chwili ustalił się na poszczególnych dalmierzach.

Opierając się na takich założeniach, jeżeli oba czujniki nie wykrywają na swojej drodze przeszkody to robot porusza się z maksymalną możliwą prędkością. Gdy jeden z czujników wykryje obecność przeszkody na poziomie L3 lub/i R3 prędkość poruszania się robota zostanie zmniejszona o połowę. Gdy przeszkoda znajdzie się w odległości poziomu L2 lub/i R2 prędkość z jaką porusza się robot zostanie ustalana na 1/4 prędkości maksymalnej. Jeżeli natomiast przeszkoda zostanie wykryta na poziomie L1 lub/i R1 robot skręci w prawo lub lewo w zależności od położenia przeszkody i wyznaczonego celu.

Na rysunku 3.29 zaprezentowane zostały cztery najbardziej typowe ze wszystkich z możliwych sytuacji oraz definicja manewru jaki robot podejmie, aby skutecznie ominąć napotkaną przeszkodę.



Rysunek 3.29: Przykładowe sytuacje i definicja reakcji robota

Uogólniając zasadę działania algorytmu, robot dostosowuje swoją prędkość na podstawie aktualnej odległości od przeszkody odczytanej z czujników odległości. W chwili gdy robot znajdzie się bezpośrednio przed przeszkodą skręci on w prawo lub lewo aby ją ominąć. Kąt o jaki wykonany zostanie zakręt zależy od tego na którym czujniku przeszkoda została wykryta oraz aktualnego celu do którego robot zmierza. Szczegółowy opis poszczególnych możliwych sytuacji oraz rodzaj akcji podejmowanej przez robota jest przedstawiony w tabeli 3.5.

Podane w tabeli wartości mają charakter orientacyjny i gdy robot ma wyznaczony cel do którego zmierza kierunki skrętu mogą w niektórych przypadkach ulec zmianie. Mogą również pojawić się dodatkowe zmiany kierunku jazdy umożliwiające powrócenie na ścieżkę do wyznaczonego punktu docelowego.

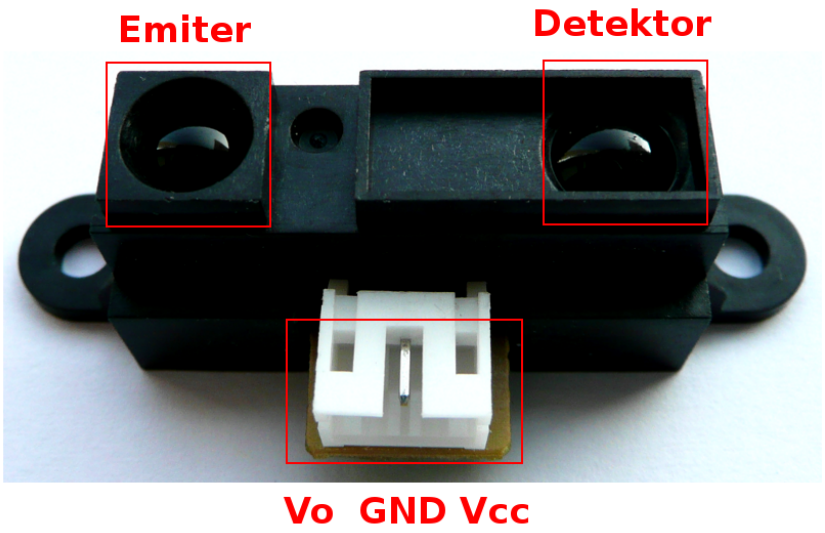
W pierwszych sześciu kolumnach tabeli prezentowany jest stan ustalony na poszczególnych poziomach odległości z podziałem na czujnik prawy i lewy. Jedynka w odpowiedniej kolumnie oznacza obecność przeszkody w danym przedziale odległości, natomiast zerem oznaczany jest brak przeszkody. W ostatniej kolumnie prezentowana jest akcja jaką podejmie robot po wykryciu danej konfiguracji stanów.

Tabela 3.5: Zachowanie robota w zależności od stanu wykrywanego na czujnikach odległości

R3	L3	R2	L2	R1	L1	Akcja robota
0	0	0	0	0	0	Maksymalna prędkość
0	1	0	0	0	0	Średnia prędkość
0	1	0	1	0	0	Minimalna prędkość
0	1	0	1	0	1	Skręt w lewo o 45°
1	0	0	0	0	0	Średnia prędkość
1	0	1	0	0	0	Minimalna prędkość
1	0	1	0	1	0	Skręt w prawo o 45°
1	1	0	0	0	0	Średnia prędkość
1	1	1	0	0	0	Minimalna prędkość
1	1	1	0	1	0	Skręt w prawo o 45°
1	1	0	1	0	0	Minimalna prędkość
1	1	0	1	0	1	Skręt w lewo o 45°
1	1	1	1	0	0	Minimalna prędkość
1	1	1	1	0	1	Skręt w lewo o 45°
1	1	1	1	1	0	Skręt w prawo o 45°
1	1	1	1	1	1	Skręt w lewo o 90°

3.5.2. Implementacja

W zrealizowanej w ramach pracy magisterskiej implementacji robot, Dark Explorer, wyposażony został w dwa dalmierze Sharp GP2D12 o efektywnym zasięgu od 10 do 80 cm. Każdy z dalmierzy wyposażony jest w nadajnik i odbiornik IR. Czujnik GP2D12 (rys. 3.30) samodzielnie dokonuje pomiaru odległości i zwraca go w postaci sygnału analogowego. Dlatego też, robot wykorzystuje konwerter analogowo-cyfrowy do przetwarzania wyniku pomiaru otrzymanego z czujnika. Wartość napięcia otrzymana na wyjściu z konwertera jest za pomocą równania, otrzymanego na podstawie charakterystyki wyjściowej dalmierza, zamieniana na odległość czujnika od przeszkody. Na zdjęciu poniżej zaznaczone zostały poszczególne elementy składowe czujnika oraz kolejne wyprowadzenia.

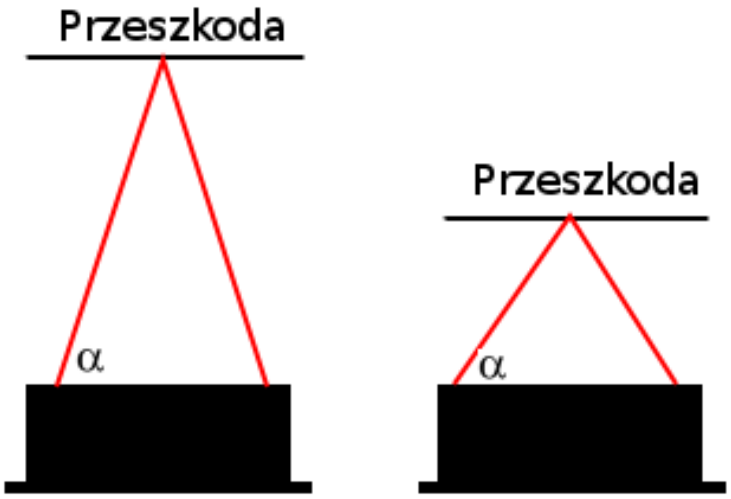


Rysunek 3.30: Budowa czujnika GP2D12 wraz z wyporowadzeniami

Zastosowany czujnik odległości został wyposażony w unowocześniony sposób dokonywania pomiarów odległości. Użyty dalmierz wykorzystuje triangulację oraz małą liniową matrycę CCD aby obliczyć odległość od przeszkody w polu widzenia sensora [15]. Podstawową zasadą działania nowego systemu pomiaru odległości jest wysłanie impulsu światła podczerwonego który po odbiciu od przeszkody wraca do detektora i tworzy trójkąt pomiędzy emiterem, detektorem a przeszkodą. Kąty w tym trójkącie zmieniają się razem z odległością od przeszkody, co zostało zaprezentowane na rysunku 3.31.

Kluczem do poprawnego działania takiego rozwiązania jest soczewka która dba o to, aby odbite światło trafiało na odpowiednie pole matrycy CCD w zależności od wartości kątów w opisanym powyżej trójkącie. Dzięki takiej budowie soczewki, czujnik na podstawie danych z matrycy CCD może określić kąt pod jakim światło odbite wróciło, a dzięki temu jest w stanie obliczyć aktualną odległość od przeszkody.

Zastosowana w dalmierzu metoda pomiaru odległości jest niemal całkowicie odporna na zakłócenia spowodowane światłem zewnętrznym. Dodatkowo eliminuje błędy pomiarowe związane z kolorem powierzchni przeszkody. Dzięki zastosowaniu takiego podejścia możliwe jest wykrycie całkowicie czarnej ściany w całości oświetlonej przez intensywne światło słoneczne, co w przypadku starszych dalmierzy praktycznie nie jest możliwe.



Rysunek 3.31: Zasada przeprowadzania pomiaru przez czujnik GP2D12

Zgodnie z założeniami algorytmu obszar zasięgu czujników został podzielony na trzy poziomy. Poziom trzeci zostały wyznaczony w odległości 50 cm, drugi poziom wyznaczony został w odległości 25 cm, a poziom pierwszy wyznaczony został w odległości 12 cm od przeszkody. Program zarządzający działaniem robota odczytuje kolejno dane o odległości otrzymane z czujnika prawego i lewego i kontroluje zachowanie robota zgodnie z zasadami przedstawionymi w tabeli 3.5. Z praktycznego punktu widzenia implementacja tego rodzaju algorytmu sprowadza się do stworzenia tablicy w której przechowywane są prędkości poruszania się każdego z czterech kół z uwzględnieniem lokalizacji czujnika oraz progu odległości w jakim znajduje się przeszkoda.

Aby móc praktycznie wykorzystać dane analogowe uzyskane z dalmierza wykorzystany został wbudowany w robota przetwornik analogowo-cyfrowy (ADC). Sposób obsługi przetwornika analogowo-cyfrowego, na potrzeby omijania przeszkód, został zrealizowany w oparciu o dokumentację i przykłady zawarte w książce J. Augustyna [16].

Zgodnie ze specyfikacją, przetwornik, wbudowany w procesor SAM7S pozwala na przetwarzanie napięcia w granicach od 0V do napięcia referencyjnego które w przypadku robota Dark Explorer jest równe napięciu zasilania procesora tj. 3.3V. Aby uzyskać jak najdokładniejszy wynik pomiaru przetwornik został skonfigurowany do pracy z rozdzielczością 10-bitową. Wydłuża to czas potrzebny na konwersję danych jednakże przy

prędkościach z jakimi robot się porusza nie ma to większego wpływu na jakość działania aplikacji gdyż wykorzystany przetwornik ADC mierzy i zwraca wartość chwilową napięcia [16].

W trakcie działania aplikacji uruchamiany jest proces konwersji na kanałach do których podłączone są czujniki odległości, a następnie aplikacja oczekuje na wywołanie przerwania związanego z wpisaniem wartości na temat napięcia do odpowiednich rejestrów. Po otrzymaniu wartości napięcia widocznego na wyjściu z dalmierzy, obliczana jest odległość w jakiej znajduje się przeszkoda. Ze względu na zastosowaną metodę pomiarową napięcie wyjściowe z czujników odległości nie ma charakterystyki liniowej. Jest to związane z obliczeniami trygonometrycznymi wykonywanymi w celu wyznaczenia odległości od przeszkody. W tym celu, udostępniona w ramach dokumentacji czujnika [17] charakterystyka wyjściowa, posłużyła do wyznaczenia progów napięcia dla zdefiniowanych przez algorytm progów odległości.

W ramach testów przeprowadzono kilka symulacji działania implementacji dla różnych pozycji startowych przy zbliżaniu się robota to brzegu przeszkody. Już pierwsze testy wykazały, że gdy robot porusza się prostopadle do płaszczyzny ściany algorytm działa bardzo dobrze, jednakże w przypadku gdy ruch ten nie jest prostopadły w niektórych przypadkach pojawiają się problemy z działaniem algorytmu. Wspomniane problemy związane są w niektórych przypadkach z odbieraniem przez dalmierz dodatkowych sygnałów wysłanych z sąsiedniego czujnika. Rozwiązaniem takiego problemu mogłoby być zastosowanie czujników z kodowaniem sygnału. Co więcej zasada działania dalmierzy IR nie pozwala na wykrycie małych i wąskich obiektów stawianych na drodze robota. Dodatkowym problemem jest nieliniowość charakterystyki czujnika która przy odległości mniejszej od 10cm wskazuje napięcia które mogą być mylnie zinterpretowane.

3.6. Wyświetlacz LCD

Do sygnalizacji wykonywanych czynności oraz polepszenia komunikacji pomiędzy człowiekiem a robotem został wykorzystany wyświetlacz LCD. Użyty model to wyświetlacz LCD 2x16, ze złączem pasującym do płyt systemu EVBXXX, z podświetleniem biało-niebieskim. Element ten został wybrany ze względu na stosunkowo niską cenę oraz możliwość podłączenia do płyty ewaluacyjnej EVBsam7s wykorzystywanej podczas rozwoju robota.

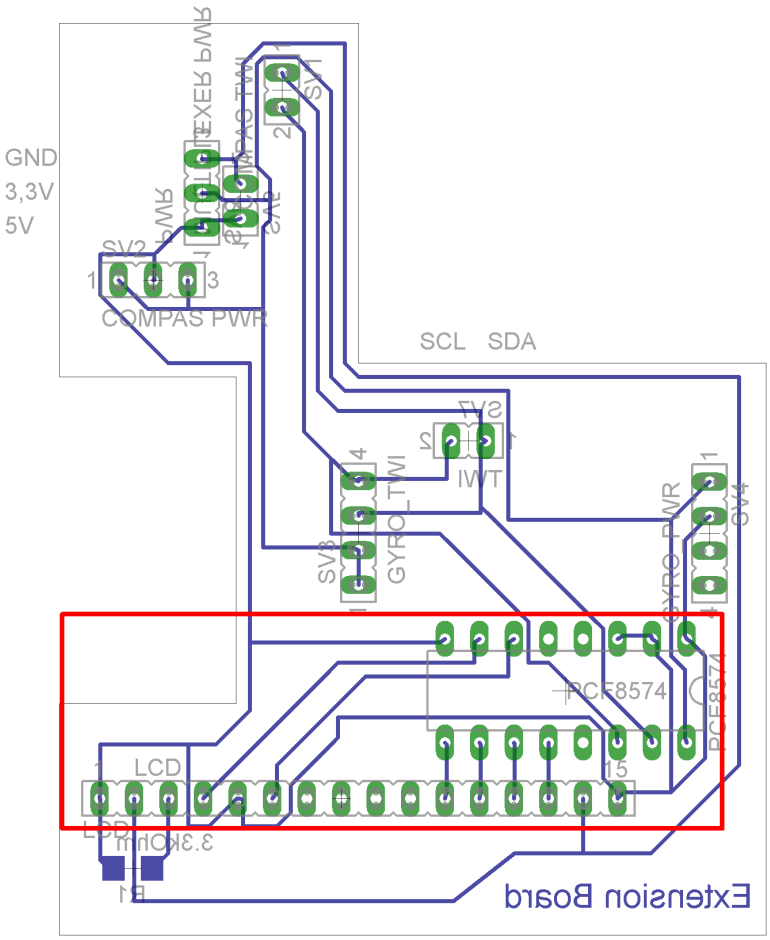
Na wyświetlaczu pokazywane są czynności aktualnie wykonywane przez robota oraz informacje pobierane z czujników zamontowanych na nim. Przykładowe komunikaty pokazywane przez Dark Explorera mówią o: aktualnej temperaturze, ilości kroków które wykonał użytkownik, kierunku w jakim podąża. Podłączony do robota wyświetlacz pokazany jest na zdjęciu 3.32.

Rysunek 3.32: Wyświetlacz LCD z komunikatem powitalnym

W celu wykorzystania wyświetlacza LCD konieczne było zastosowanie jakiegoś sposobu na redukcję wejść oraz wyjść wykorzystywanych przez to urządzenie. W konfiguracji podstawowej Dark Explorer udostępniał jedynie pięć złącz GPIO[16] natomiast sam wyświetlacz potrzebuje takich złącz sześć.

[16] GPIO – General Purpose Input/Output, złącza wejścia wyjścia ogólnego przeznaczenia

Z pomocą przyszedł tutaj 8-bitowy ekspander GPIO z interfejsem I^2C[17]. Wykorzystany element to PCF8574 firmy Philips. Układ ten posiada ośmio bitowy rejestr w którym przechowuje słowo odebrane poprzez interfejs I^2C. Każdy bit tego rejestru jest odzwierciedlany jako stan wysoki lub niski na odpowiednich wyjściach GPIO. W taki sposób możliwe jest sterowanie wyświetlaczem LCD. Wykorzystując ten pomysł można znacząco rozszerzyć ilość portów GPIO na robocie i sterować dowolnymi urządzeniami. Schemat płyty rozszerzeń z zaznaczonym modułem wyświetlacza LCD widoczny jest na rysunku 3.33.



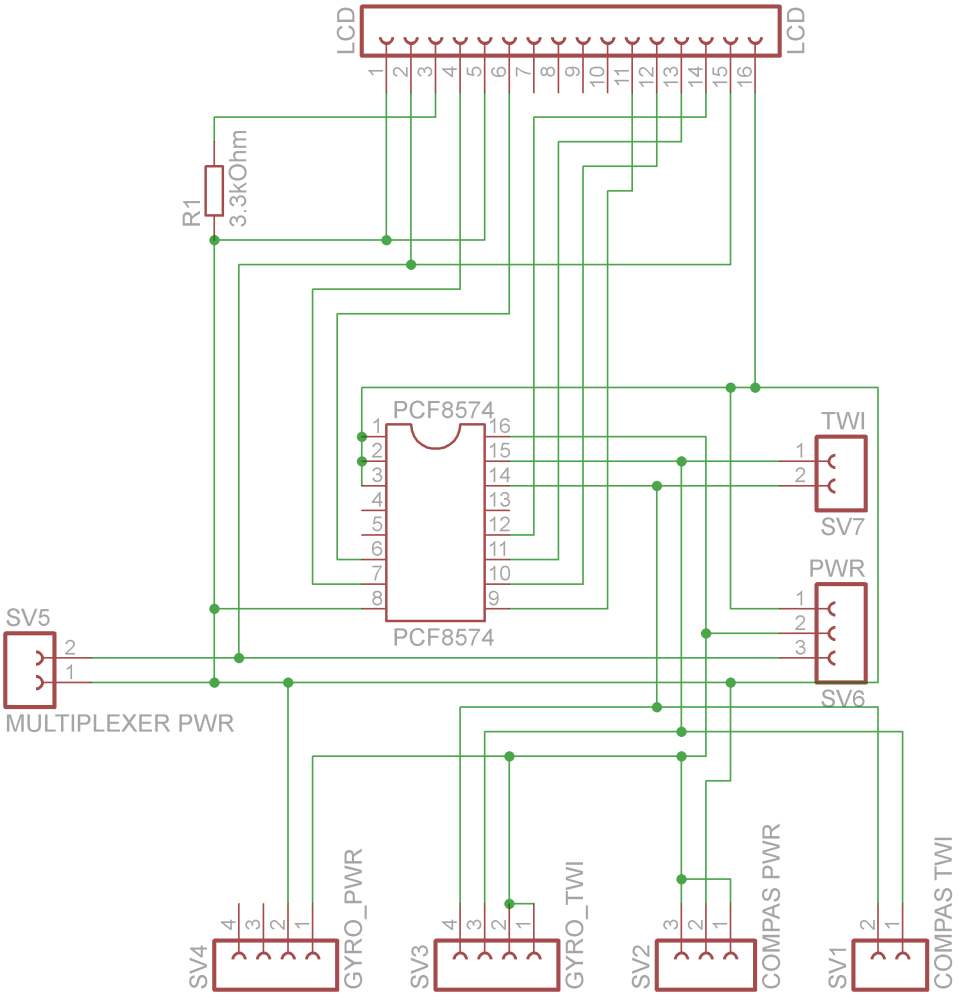
Rysunek 3.33: Layout cyfrowej części płyty rozszerzeń z zaznaczonymi elementami odpowiedzialnymi za obsługę wyświetlacza LCD

Wyświetlacz został zakupiony jako moduł z wtykiem 16 pinowym. Można go podłączyć do gniazda LCD na płycie rozszerzeń Dark Explorera.

[17] I^2C – szeregowa dwukierunkowa magistrala służąca do przesyłania danych w urządzeniach elektronicznych

3.7. Płyta rozszerzeń

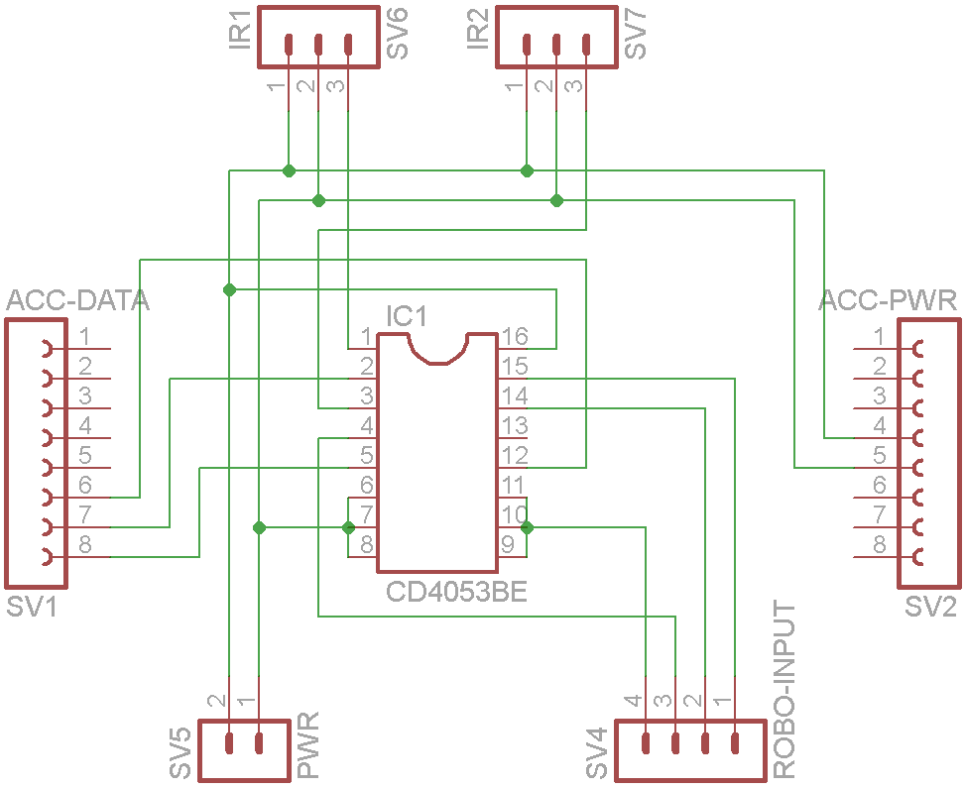
Wszystkie elementy elektroniczne stworzone na rzecz tej pracy magisterskiej zostały połączone przy pomocy płyty rozszerzeń. Doprowadza ona zasilanie oraz odpowiedni interfejs komunikacyjny do poszczególnych urządzeń. Płyta ta została przygotowana przy pomocy oprogramowania Eagle[18] w wersji edukacyjnej. Niestety z powodu ograniczeń na maksymalny rozmiar płytki stworzonej za pomocą tego oprogramowania z licencją edukacyjną, konieczne było podzielenie płyty rozszerzeń na dwie części.



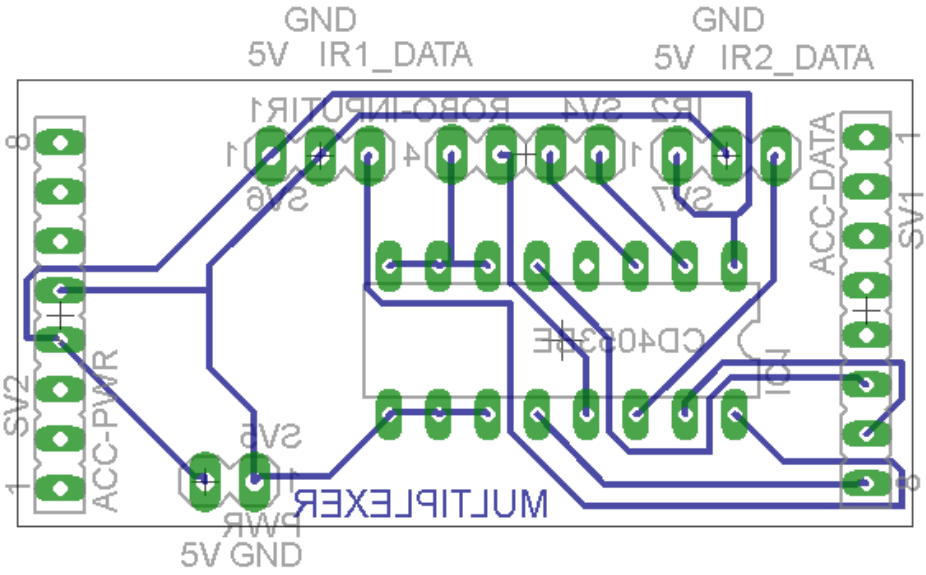
Rysunek 3.34: Schemat części cyfrowej płyty rozszerzeń

Jedna część płyty rozszerzeń (rys. 3.34) jest odpowiedzialna za wszystkie elementy cyfrowe które komunikują się z robotem przy pomocy interfejsu I^2C. Druga natomiast (rys. 3.35, 3.36) pozwala na podłączanie elementów analogowych których sygnały są interpretowane przez przetwornik analogowo cyfrowy będący jednym z urządzeń peryferyjnych mikrokontrolera ARM.

[18] Eagle – Easily Applicable Graphical Layout Editor



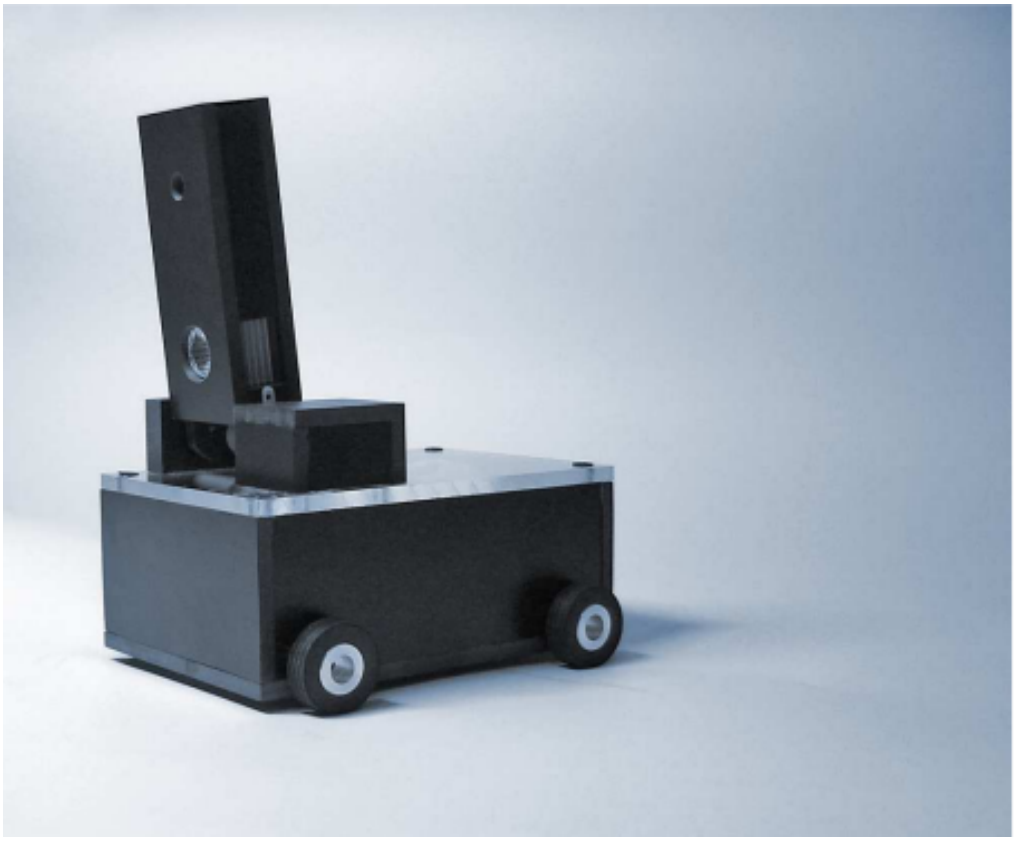
Rysunek 3.35: Schemat części analogowej płyty rozszerzeń



Rysunek 3.36: Layout analogowej części płyty rozszerzeń

3.8. Rozbudowa obudowy robota

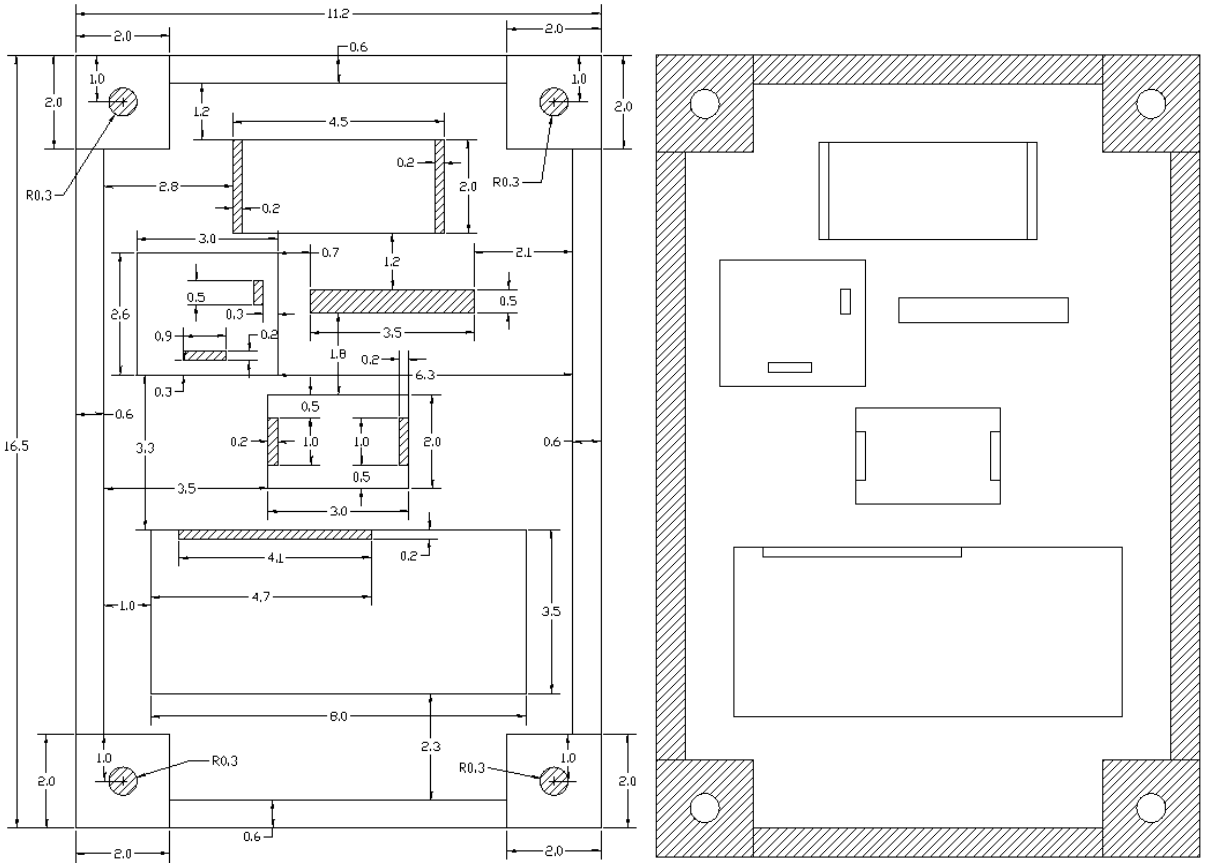
Obudowa robota Dark Explorer stworzona w ramach poprzedniej pracy magisterskiej [1] składała się z dwóch elementów bazowych. Pierwszym elementem jest podwozie wykonane ze spienionego PCW o grubości 6mm. Głównym jego zadaniem jest zapewnienie ochrony wszystkim podzespołom elektronicznym zainstalowanym w robocie, jak również dostarczenie punktów mocowania pozwalających na integrację poszczególnych grup funkcjonalnych. Drugim ważnym elementem jest pokrywa wierzchnia wykonana z przeźroczystej płyty pleksi na której zamontowany został serwomechanizm z wieżą na której zamontowana została kamera. Całość po zmontowaniu prezentuje się w sposób pokazany na rysunku 3.37.



Rysunek 3.37: Wygląd robota w pierwotnej konfiguracji po zmontowaniu wszystkich elementów

Ze względu na dużą ilość czujników dodatkowych których obsługa została dodana w obecnej wersji robota, wszelkie próby wykorzystania istniejącej obudowy zakończyły się niepowodzeniem. Dlatego też zaprojektowany został ekspander pozwalający w łatwy sposób zintegrować poprzednią konstrukcję z zestawem nowych czujników i urządzeń peryferyjnych. Wspomniany ekspander ma postać prostopadłościanu wykonanego w całości z płyt bezbarwnej pleksi. Wszystkie elementy elektroniczne zostały rozmieszczone

na spodniej płycie ekspandera, a potrzebne połączenia zostały wyprowadzone poprzez otwory z gniazdami zainstalowanymi w miejscu podłączenia czujnika. Schemat projektowy dolnej płyty ekspandera widoczny jest na rysunku 3.38.



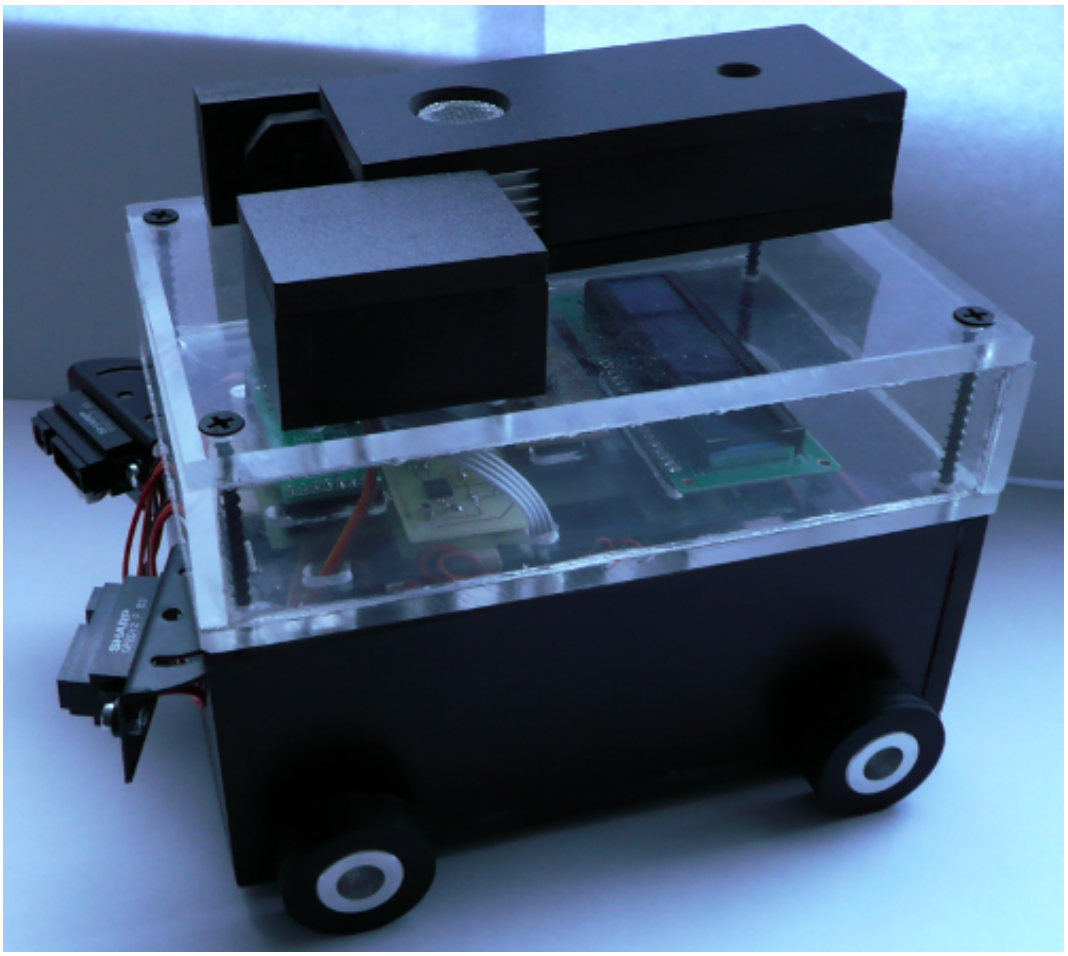
(a) Wymiarowanie projektu obudowy

(b) Rozmieszczenie poszczególnych elementów wewnątrz obudowy

Rysunek 3.38: Projekt modułu rozszerzeń obudowy robota

W centrum ekspandera umiejscowiony został żyroskop, celem takiego zabiegu było wyeliminowanie potencjalnych problemów związanych z pomiarem kąta o jaki robot się obrócił w przypadku gdy czujnik pomiarowy nie znajduje się na osi wzdłuż której obrót jest dokonywany. W południowej części umieszczony został wyświetlacz LCD za pomocą którego użytkownik będzie mógł na bieżąco monitorować aktualne działania robota. W północnej części zamontowany został akcelerometr oraz magnetometr. Taka konfiguracja pozwoliła na ograniczenie ilości i długości przewodów potrzebnych do podłączenia wszystkich elementów do płyty głównej robota. Do tak przygotowanej płyty dołączona została elektronika umożliwiająca podłączenie wszystkich dodatkowych modułów. Szczegółowa prezentacja elektronicznej części płyty głównej ekspandera zamieszczona została w rozdziale 3.7.

Całość obudowy została zaprojektowana w taki sposób aby umożliwić bezproblemowe podłączanie i odłączenie nie tylko poszczególnych czujników ale również całego ekspandera. Takie podejście do problemu nie tylko nie ogranicza możliwości dalszego rozwoju, ale umożliwia również swobodne modyfikowanie zestawu podłączonych czujników, co w znaczącym stopniu ułatwia dodawanie nowych modułów pomocniczych. Wygląd działającego ekspandera dołączonego do pierwotnej budowy robota można zobaczyć na zdjęciu 3.39.

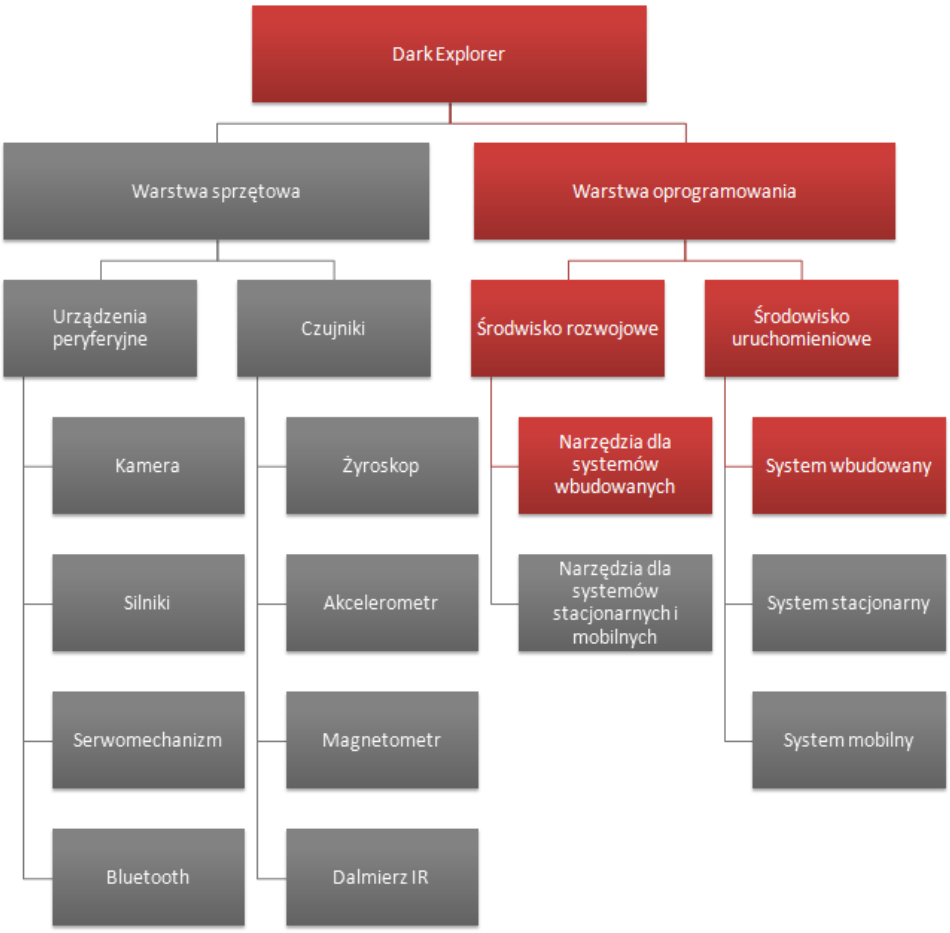


Rysunek 3.39: Wygląd robota z zamontowanym modułem rozszerzeń

Rozdział 4

Rozwój oprogramowania systemu wbudowanego

Podczas przygotowywania się do rozpoczęcia pracy z nową technologią, każdy programista powinien być świadomy tego jakie są istniejące narzędzia, które mogą mu pomóc w pracy. W tym rozdziale opisany jest sposób instalacji oraz używania narzędzi wykorzystywanych przez autorów tej pracy (rys. 4.1).



Rysunek 4.1: Struktura platformy robota mobilnego po zakończeniu prac. Kolorem czerwonym oznaczono zakres prac opisanych w bierzącym rozdziale.

4.1. Narzędzia dla systemu Windows

Pierwszym krokiem do przygotowania środowiska rozwojowego umożliwiającego rozwijanie oprogramowania sterującego robotem jest instalacja sterowników wymaganych przez system Windows do obsługi interfejsu JTAG[1] za pomocą którego odbywa się proces wgrywania przygotowanego oprogramowania do pamięci robota. Kolejnym wymaganym krokiem jest instalacja i konfiguracja narzędzi umożliwiających tworzenie programów które będą mogły być uruchomiene w ramach platformy sprzętowej robota. Ostatnim etapem przygotowań jest instalacja oprogramowania umożliwiającego programowanie układu za pomocą wspomnianego interfejsu JTAG oraz debugowanie aplikacji w trakcie jej działania na robocie.

4.1.1. Instalacja WinARM

Do kompilacji kodu źródłowego oprogramowania zajmującego się sterowaniem podzespołami robota wykorzystany został zestaw narzędzi znany pod nazwą WinARM. WinARM jest zestawem narzędzi umożliwiających tworzenie oprogramowania dla kontrolerów opartych na platformie ARM. W odróżnieniu od innych dostępnych obecnie rozwiązań, środowisko to, nie wymaga dodatkowej instalacji narzędzi udostępnianych w ramach MinGW[2] czy też Cygwina[3]. Wszystkie potrzebne narzędzia dostarczane są w ramach SDK[4]. Narzędzia WinARM pomyślnie przeszły testy z kontrolerami Atmel AT91SAM7S64, AT91SAM7S256, AT91RM9200 ARM7TDMI oraz Philips LPC2106, Philips LPC2129, Philips LPC2138, Philips LPC2148. Dodatkowo dostarczane w ramach środowiska kompilatory i narzędzia powinny prawidłowo współpracować ze wszystkimi mikrokontrolerami opartymi o architekturę ARM(-TDMI/Thumb itp.).

Instalację środowiska WinARM należy rozpocząć od pobrania archiwum z najnowszą wersją narzędzi ze strony `http://gandalf.arubi.uni-kl.de/avr_projects/arm_projects/`. W chwili pisania pracy dostępna była wersja środowiska WinARM w wersji 20060606. Po zakończeniu procesu pobierania, archiwum należy rozpakować w taki sposób aby wszystkie podstawowe narzędzia dostępne były w katalogu `C:\WinARM\bin`.

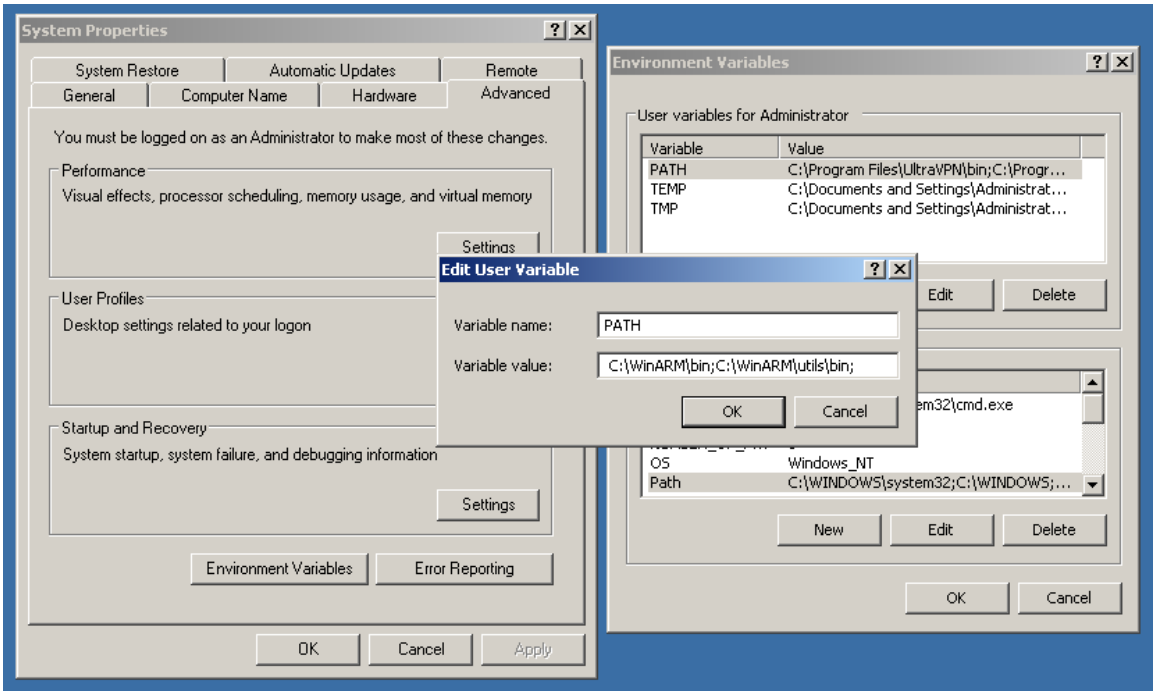
[1] Joint Test Action Group - jest to nazwa standardu który definuje protokół wykorzystywany do testowania połączeń na płytkach drukowanych oraz uruchamiania i programowania układów i systemów mikroprocesorowych.

[2] Minimalist GNU for Windows - port GCC dostarczający zestaw darmowych narzędzi do komiplacji natywnych plików wykonywalnych dla platformy Windows

[3] Cygwin - implementacja standardu POSIX przeznaczona dla systemów z rodziny Windows

[4] SDK (z ang. Software Development Kit) - Zestaw narzędzi do rozwoju oprogramowania

Umieszczenie katalogu z narzędziami WinARM w innej lokalizacji jest również możliwe, ale może wymagać wykonania dodatkowych operacji konfiguracyjnych w celu zapewnienia poprawności działania wszystkich narzędzi. Aby udostępnić narzędzia WinARM z linii poleceń systemu Windows konieczne jest dodanie do zmiennej systemowej `PATH` ścieżki do katalogów z plikami wykonywalnymi biblioteki. W przypadku instalacji w podanym powyżej katalogu wartości powinny być następujące `C:\WinARM\bin;C:\WinARM\utils\bin;`. Jeżeli jednak katalog z pakietem został umieszczony w innej lokalizacji konieczne jest odpowiednie zmodyfikowanie wspomnianych wpisów. Szczegóły okna konfiguracji widoczne są na rysunku 4.2

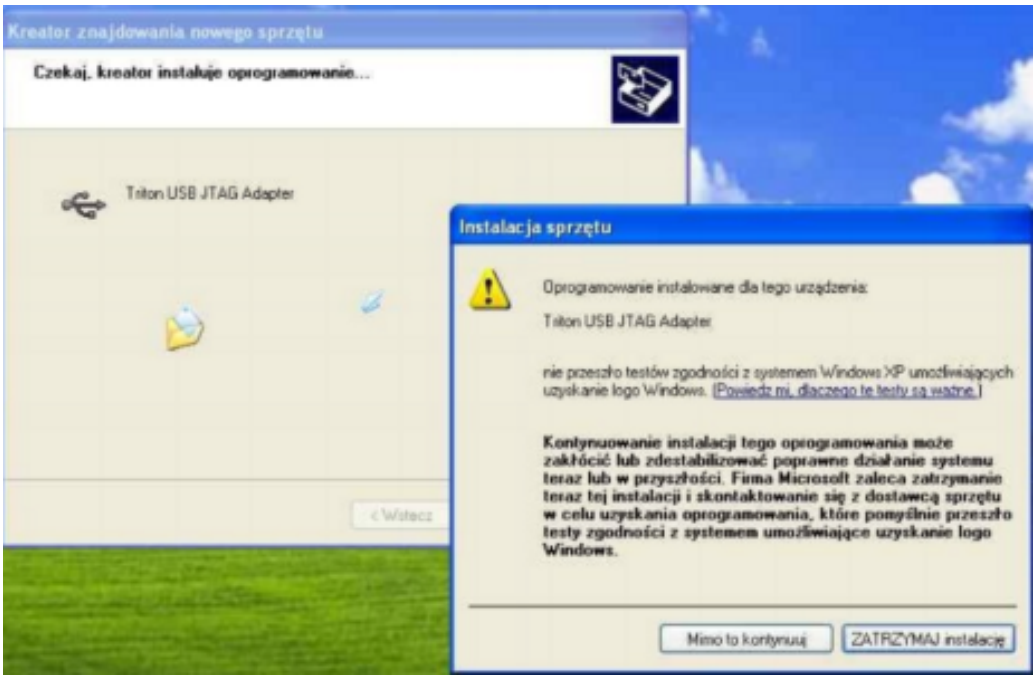


Rysunek 4.2: Konfiguracja narzędzi pakietu WinARM

4.1.2. Instalacja sterowników programatora

Programowanie oraz debugowanie aplikacji robota może zostać zrealizowane za pomocą dowolnego programatora kompatybilnego z interfejsem JTAG. Programatory oparte o interfejs LPT nie wymagają od użytkownika żadnej dodatkowej konfiguracji. Nieco inaczej wygląda sytuacja z programatorami opartymi o interfejs USB, które to wymagają przed pierwszym użyciem zainstalowania sterowników umożliwiających prawidłowe rozpoznanie programatora przez system Windows. Jednym z bardziej popularnych programatorów USB jest TriTon JTAG. TriTon JTAG to programator przeznaczony dla procesorów zbudowanych w oparciu o rdzeń ARM podłączany do komputera za pomocą portu USB.

TriTon JTAG posiada standardowe 20-pinowe złącze JTAG wraz z wyprowadzeniami sygnałów RxD i TxD interfejsu UART. TriTon JTAG współpracuje z OpenOCD[5], pozwalając na programowanie oraz debugowanie działającej na urządzeniu aplikacji. Urządzenie oparte jest o układ FT2232 który umożliwia jego współprace także z innymi środowiskami rozwoju oprogramowania dla platformy ARM, kompatybilnymi z FT2232.



Rysunek 4.3: Instalacja sterowników do programatora TriTon JTAG

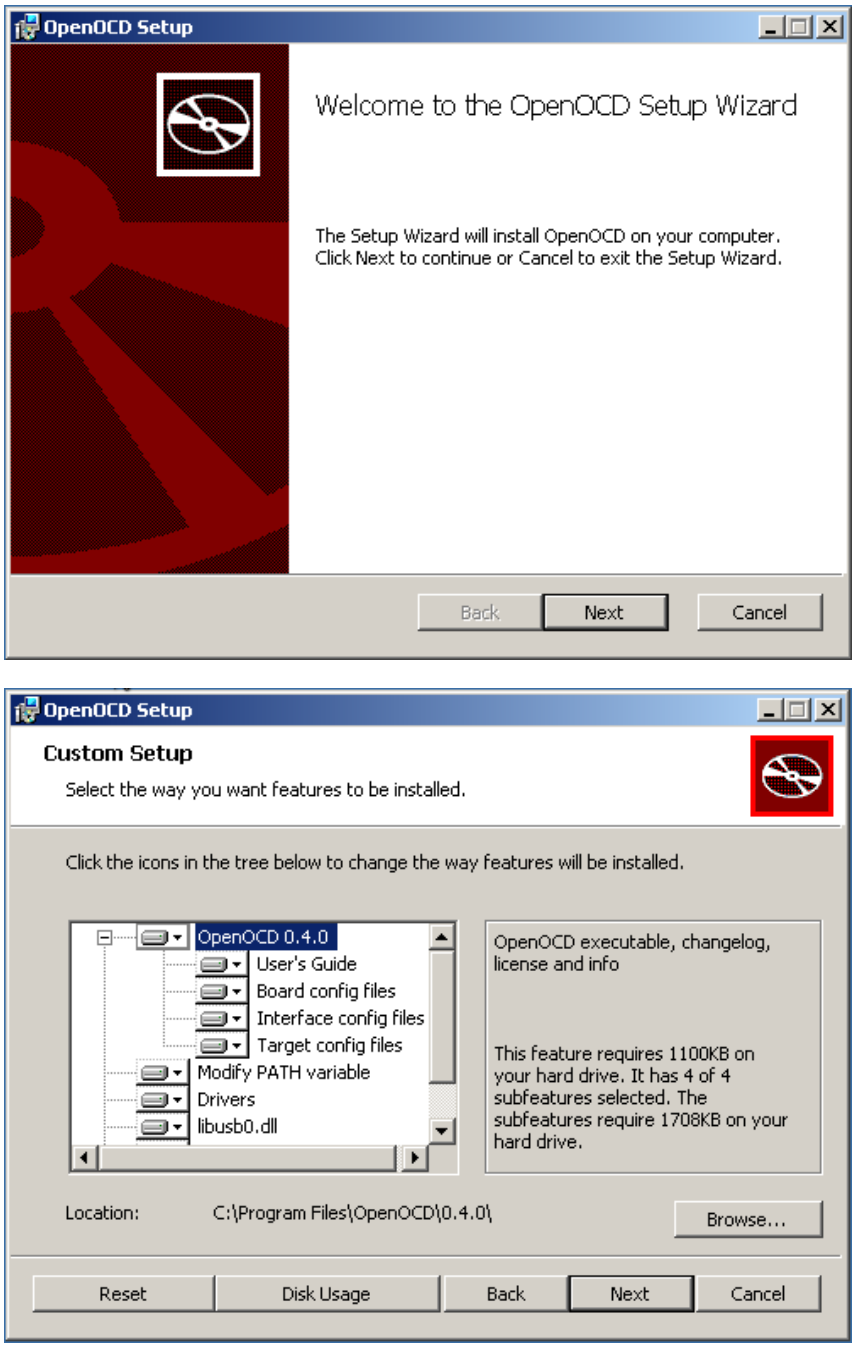
Instalację programatora należy rozpocząć od pobrania sterowników do układu FT2232 ze strony `http://www.ethernut.de/en/download/`. Na stronie dostępne są sterowniki przeznaczone dla systemów Windows 2000, XP, Server 2003, Vista oraz Server 2008. Po rozpakowaniu archiwum ze sterownikami należy za pomocą interfejsu USB podłączyć programator do komputera. Po wykryciu system Windows trzykrotnie poprosi o podanie ścieżki do sterowników do urządzeń Triton JTAG, Triton USB RS232 Adapter oraz USB Serial Port. Należy wtedy skazać ścieżkę do katalogu w którym rozpakowane zostały sterowniki pobrane ze strony wspomnianej wcześniej. W przypadku pojawienia się ostrzeżenia o braku testów zgodności, należy kontynuować instalację klikając na przycisk „Mimo to kontyuuj" (rys. 4.3). Po poprawnym zakończeniu instalacji w Menadżerze urządzeń systemu Windows widoczne będą następujące elementy:

— Triton USB JTAG Adapter,
— Triton USB RS232 Adapter,
— Triton JTAG

[5] OpenOCD - Open On-Chip Debbuger

4.1.3. Instalacja Open On-Chip Debugger

OpenOCD zostało zapoczątkowane przez Dominika Rath w ramach pracy dyplomowej realizowanej na uniwersytecie w Augsburg. Od tamtego czasu OpenOCD bardzo się rozwinęło i urosło do rozmiarów aktywnego projektu open-sourcowego wspieranego przez programistów z całego świata. Celem OpenOCD jest dostarczenie uniwersalnego narzędzia umożliwiającego debugowanie i programowanie systemów wbudowanych.



Rysunek 4.4: Instalator Open On-Chip Debugger'a (OpenOCD)

Strona projektu OpenOCD dostępna jest pod adresem `http://openocd.berlios.de/web/`. Dostępne są tam zarówno źródła jak i dokumentacja do projektu. Niestety w chwili pisania pracy magisterskiej autorzy nie udostępniali wersji skompilowanej dla systemu Windows. Dlatego też użyta została niezależna wersja OpenOCD z przygotowanym instalatorem dla systemu Windows. Instalator OpenOCD dla Windows jest do pobrania ze strony `http://www.ethernut.de/en/download/`. Po uruchomieniu instalatora wybieramy lokalizację docelową w której OpenOCD ma zostać zainstalowane. Po zakończeniu instalacji konieczne jest uzupełnienie wartości zmiennej systemowej PATH ścieżką do miejsca instalacji OpenOCD, domyślnie `C:\ethernut\nut\tools\win32`.

4.1.4. Konfiguracja zintegrowanego środowiska programistycznego

Poprawne zainstalowanie pakietów WinARM oraz OpenOCD dostarcza wszystkich niezbędnych narzędzi potrzebnych do rozwijania aplikacji dla platform wbudowanych. Niemniej jednak korzystanie z nich wymaga bezpośredniej interakcji z linią poleceń systemu Windows, co dla niektórych programistów może być uciążliwe. Możliwe jest napisanie skryptów systemowych pozwalających na uruchamianie sekwencji procedur wymaganych np. do zaprogramowania robota, ale jest to rozwiązanie nieprzenośne i słabo konfigurowalne. Dlatego też zaleca się instalację zintegrowanego środowiska programistycznego które oprócz edytora kodu umożliwi automatyzację najczęściej wykonywanych zadań. Wybór rodzaju środowiska programistycznego zależy niemal w całości od preferencji programisty gdyż większość potrzebnych narzędzi można bez większych trudności zintegrować z ulubionym edytorem. Na potrzeby tej pracy omówiona zostanie konfiguracja dla środowiska Eclipse. Wybór podyktowany został faktem iż jest to obecnie jedna z najpopularniejszych platform do rozwoju oprogramowania, a co więcej jej konfiguracja przebiega identycznie dla systemu Windows jak i Linux. Z tego względu szczegółowy opis procedury konfiguracji zamieszczony został w rozdziale poświęconym narzędziom dla systemu Linux.

4.2. Narzędzia dla systemu Linux

Jednym z wymagań pracy dyplomowej było przygotowanie zestawu narzędzi dla systemu Linux, dzięki którym będzie możliwy dalszy rozwój robota. Zbiór programów potrzebnych do rozwoju projektu to: zintegrowane środowisko programistyczne (IDE), kompilator oraz oprogramowanie pozwalające zaprogramować mikrokontroler. Bieżący rozdział opisuje sposób instalacji i wykorzystania poszczególnych narzędzi, a także daje światło na inne tego typu oprogramowanie, które było brane pod uwagę podczas pracy nad projektem, jednak nie zostało użyte.

4.2.1. Wybór zintegrowanego środowiska programistycznego

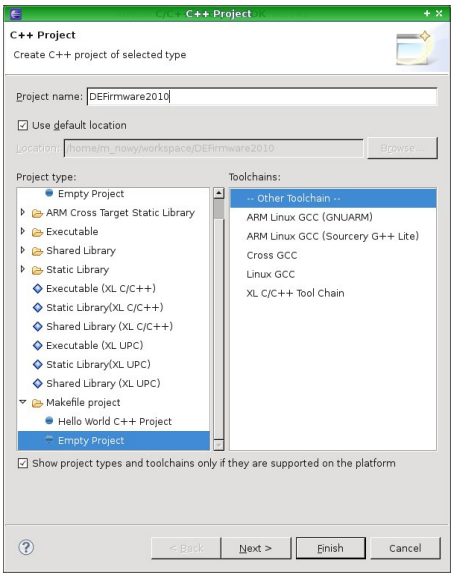
We wcześniejszej wersji oprogramowanie robota było rozwijane na systemie Microsoft Windows. Niestety zintegrowane środowisko programistyczne używane do tej pory nie jest multiplatformowe. Konieczne było zatem dobranie nowego IDE, które umożliwiało będzie rozwijanie stworzonego wcześniej kodu pod systemem Linux. Oczywiście biorąc pod uwagę niski budżet projektu, wszelkie płatne rozwiązania zostały prawie od razu odrzucone. Rozważaniom zostały poddane następujące środowiska: Eclipse, Netbeans, CodeWarrior, Kile, ARM Workbench IDE.

Ostatecznie zostało wybrane środowisko Eclipse, które jest dostępne na zasadach licencji: Eclipse Public License, odpowiadającej wymaganiom projektu. Największymi zaletami tego rozwiązania jest łatwość instalacji, konfiguracji oraz użytkowania. W podjęciu ostatecznej decyzji równie istotne było to, iż rozwiązania płatne takie jak np. CodeWarrior są bazowane na Eclips'ie. Plusem był także fakt iż dostępne są różnego rodzaju dodatki do Eclipse'a dedykowane do rozwoju oprogramowania na mikrokontrolery ARM.

4.2.2. Instalacja i konfiguracja Eclipse'a

Wybrane środowisko programistyczne (Eclipse) jest udostępniane pod adresem: `http://www.eclipse.org/downloads/`. Ściągnięte archiwum rozpakowujemy w wybranym przez nas miejscu. Przechodzimy następnie do katalogu który został wydobyty z archiwum i uruchamiamy Eclipse'a.

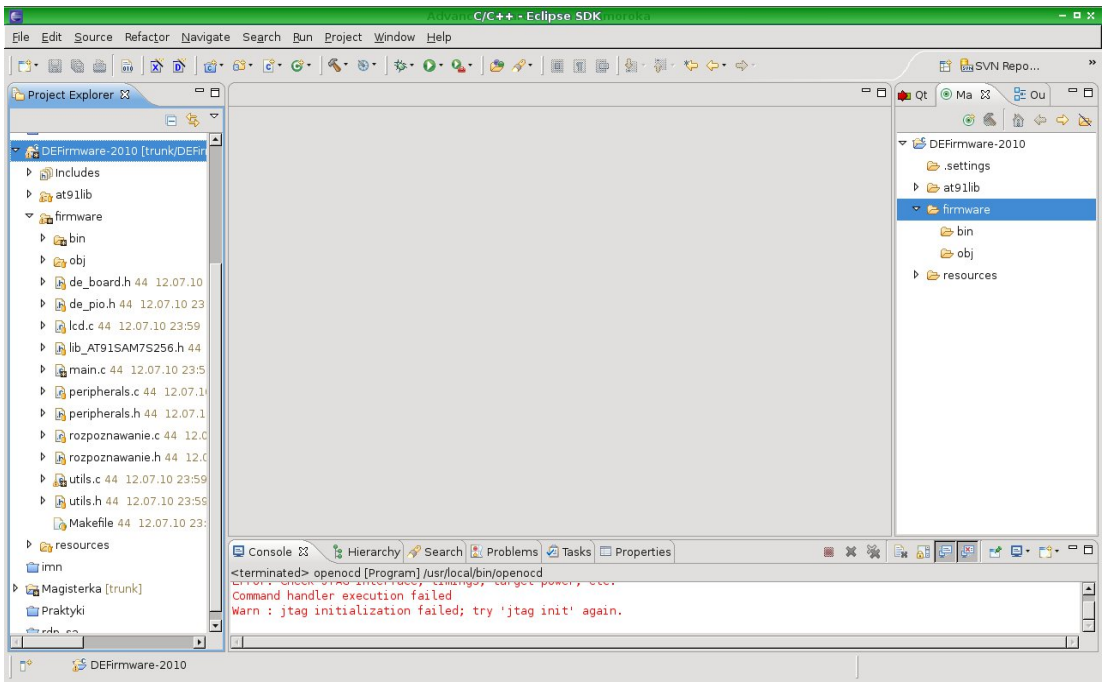
Oprogramowanie zaraz po pierwszym uruchomieniu zapyta nas o miejsce w którym będą przechowywane źródła naszego projektu tzw. Workspace. Dobrym pomysłem jest potwierdzenie ustawień domyślnych i zapamiętanie tej ścieżki.



Rysunek 4.5: Okno dialogowe C++ Project

Dodawanie projektu z firmware'm Dark Explorer'a

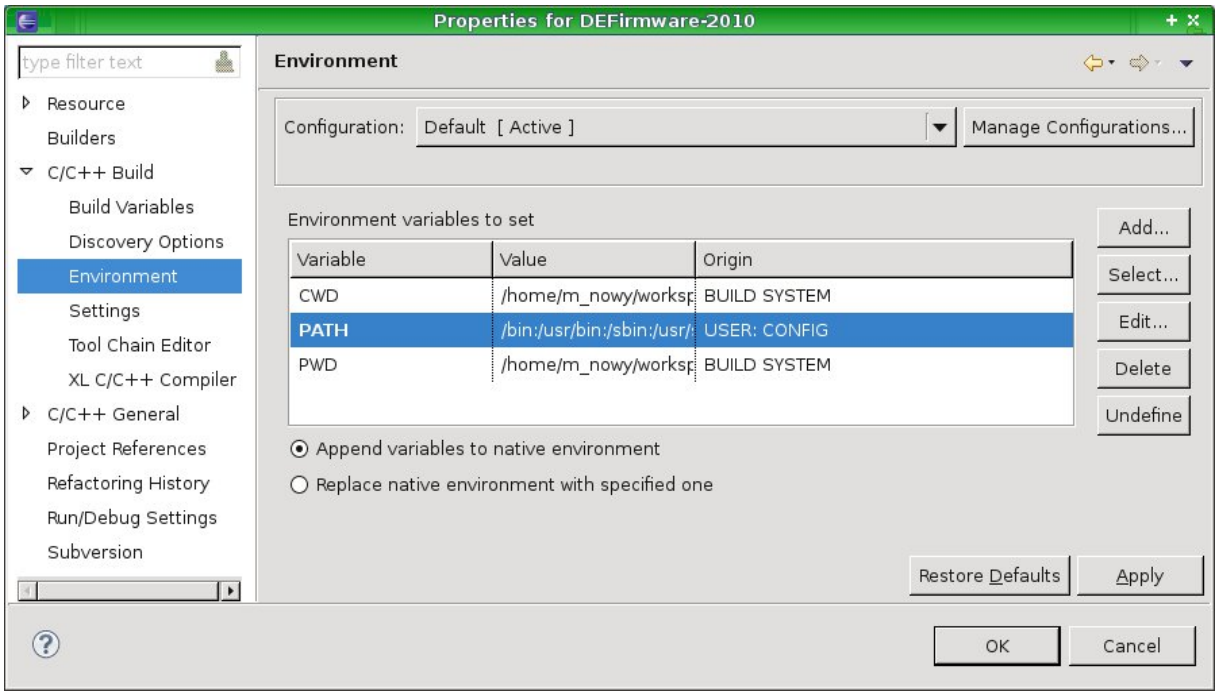
Do workspace'u Eclipse'a należy skopiować katalog z kodem sterującym robota. Następnie dodajemy nowy projekt w IDE wybierając kolejno *File –> New –> C++ Project*. W oknie dialogowym *C++ Project* (rysunek 4.5), należy podać nazwę projektu która powinna być zgodna z nazwą katalogu zawierającego kod robota. Następnie z listy Project type, wybieramy *Makefile project –> Empty Project*, a jako *Toolchain* wybieramy *Other toolchain*. Całą operację zatwierdzamy przyciskiem *Finish*.



Rysunek 4.6: Okno główne IDE Eclipse z dodanym projektem

Po poprawnym dodaniu projektu naszym oczom powinno się ukazać okno podobne do tego na rysunku 4.6.

W przypadku gdy przed uruchomieniem Eclipse'a nie ustawiliśmy ścieżki do odpowiedniego toolchain'a w zmiennych środowiskowych systemu należy podjąć dodatkowe kroki mające na celu uzupełnienie brakującej konfiguracji. Eclipse umożliwia konfigurowanie tych zmiennych dla pojedynczych projektów. W celu ustawienia wymaganej ścieżki klikamy prawym klawiszem myszy na nazwie naszego projektu w *Project Explorer* i wybieramy *Properties*.



Rysunek 4.7: Okno dialogowe Properties

Po ukazaniu się okna dialogowego *Properties* (rys. 4.7) wybieramy *C/C++ Build -> Environment*. Następnie modyfikujemy zmienną *PATH*, dodając na końcu ścieżkę do naszego toolchain'a.

W celu przetestowania poprawności naszej konfiguracji możemy spróbować skompilować kod, klikając prawym klawiszem myszy na nazwę projektu, a następnie wybierając z menu kontekstowego opcję *Build Project*. W wyniku powinniśmy otrzymać dwa pliki binarne w katalogu *firmware/bin*. Są to pliki gotowe do umieszczenia w pamięci flash robota.

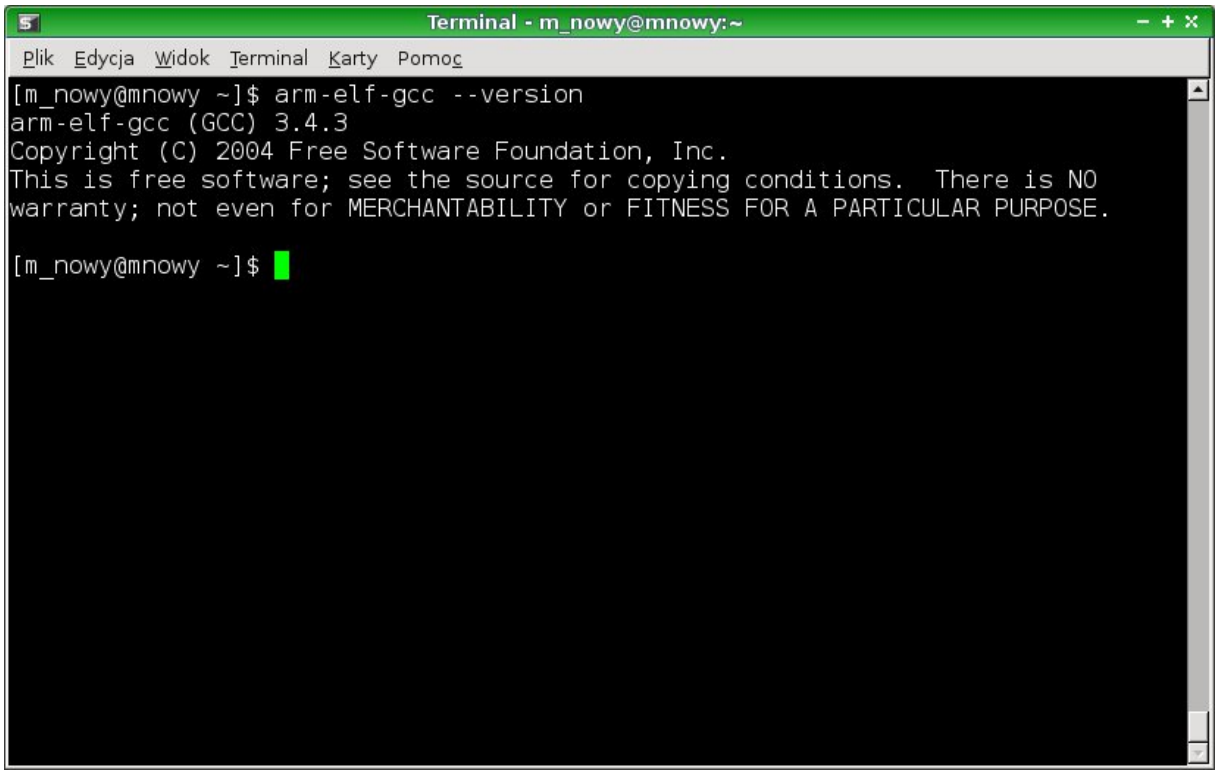
4.2.3. Instalacja i konfiguracja toolchain'a

W celu przetworzenia kodu do formy współpracującej z mikrokontrolerem ARM niezbędny nam jest odpowiedni toolchain, czyli zestaw narzędzi generujących pliki wykonywalne oraz pomagających w debugowaniu utworzonego oprogramowania. Tworzony kod był kompilowany przy pomocy GNU ARM toolchain w wersji 3.4.3 dostępnej na stronie `http://www.gnuarm.com/bu-2.15_gcc-3.4.3-c-c++-java_nl-1.12.0_gi-6.1.tar.bz2`

Po ściągnięciu archiwum ze strony producenta należy rozpakować je do dowolnego katalogu, na potrzeby tej pracy załóżmy że będzie to katalog `/usr/local/`. W celu zapewnienia dostępu do toolchain'a wszystkim programom wskazane jest dodanie ścieżki /usr/local/bin do zmiennej środowiskowej PATH (komenda `export PATH=$PATH:/usr/local/bin`). W celu sprawdzenia poprawności instalacji należy wykonać komendę:

```
arm-elf-gcc --version
```

której wynikiem powinien być komunikat podobny do tego na rysunku 4.8.



Rysunek 4.8: Okno pokazujące odpowiedź prawidłowo zainstalowanego kompilatora

Trzeba wziąć pod uwagę to, iż toolchain o którym mowa był przygotowany pod system 32–bitowy. W przypadku konfiguracji na systemie 64–bitowym konieczne jest zaopatrzenie się w 64–bitową wersje binarną toolchain'a lub skompilowanie go samodzielnie. Ewentualne dodatkowe informacje można znaleźć pod adresem `http://www.gnuarm.com`.

4.2.4. Open On–Chip Debugger – instalacja i konfiguracja

Pamięć robota była programowana przy pomocy tego samego narzędzia które służy do sprawdzania poprawności działania napisanego kodu. Mowa tu o oprogramowaniu Open On–Chip Debugger. Źródła programu należy pobrać z oficjalnej strony projektu: `http://sourceforge.net/projects/openocd/`. Autorzy programu nie zamieścili wersji binarnych, więc kompilacje będziemy musieli przeprowadzić sami.

Po ściągnięciu i rozpakowaniu źródeł uruchamiamy skrypt configure z odpowiednimi argumentami przy pomocy komendy:

```
./configure --prefix=/usr/local --enable-ft2232_libftdi
```

Dodatkowy argument powoduje włączenie obsługi urządzeń bazujących na układzie FT2232 używając biblioteki libftdi. Jest to niezbędne przy korzystaniu z programatora Triton JTAG A. W przypadku wykorzystywania programatora innej firmy możliwa będzie konieczność wprowadzenia innego argumentu do skryptu konfiguracyjnego. Argument prefix określa ścieżkę docelową instalacji oprogramowania.

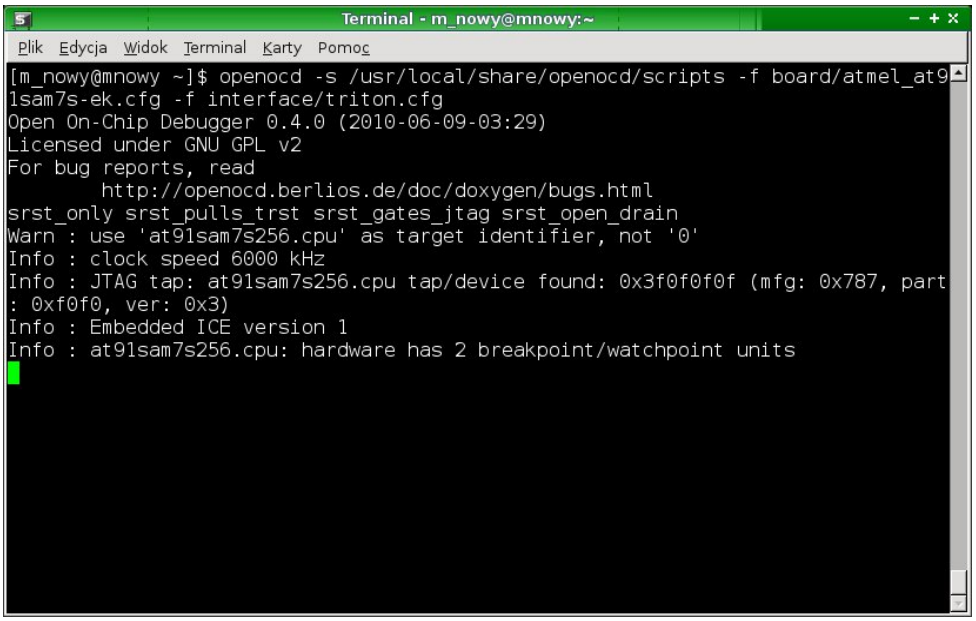
Gdy skrypt konfiguracyjny zakończy działanie z powodzeniem, możemy wywołać komendę `make && make install` w celu kompilacji i instalacji oprogramowania.

Po zakończeniu powyższych czynności należy skopiować plik triton.cfg (dodatek C) konfigurujący połączenie przy pomocy programatora Triton JTAG A do katalogu `/usr/local/openocd/interface`.

W celu przetestowania działania Open On–Chip Debugger'a należy podłączyć programator Triton JTAG A do portu usb komputera oraz portu JTAG robota. Po upewnieniu się że robot jest włączony wykonujemy komendę:

```
openocd -s /usr/local/share/openocd/scripts -f board/atmel_at91sam7s-ek.cfg
-f interface/triton.cfg
```

W wyniku wykonania polecenia na ekranie powinniśmy otrzymać widok podobny do tego który przedstawiony jest na rysunku 4.9.



Rysunek 4.9: Poprawnie uruchomiony OpenOCD

Programowanie pamięci Dark Explorer'a

W celu wgrania programu do pamięci robota musimy uruchomić dwa narzędzia: telnet oraz OpenOCD. Procedura instalacji i uruchamiania OpenOCD została opisana wcześniej w rozdziale 4.2.4. W pierwszym kroku uruchamiamy openocd w celu podłączenia się do robota. Następnie uruchamiamy program telnet za pomocą polecenia:

```
telnet localhost 4444
```

Zapewnia nam to możliwość wysyłania komend sterujących do openocd. W celu zaprogramowania pamięci flash Dark Explorer'a i uruchomienia nowej wersji firmware'u należy wykonać zestaw komend:

```
halt
flash write_image {ścieżka_do_pliku_elf}
reset init
resume
```

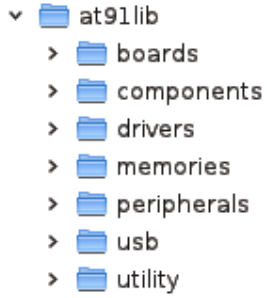
Natomiast w przypadku programowania pamięci RAM robota wykonujemy następujący zestaw poleceń:

```
halt
load_image {ścieżka_do_pliku_bin} {początkowy_adres_pamięci}
reset init
resume
```

4.3. Biblioteka funkcji mikrokontrolera ARM - AT91LIB

Programowanie mikrokontrolerów z rodziny ARM w ogólnym zarysie sprowadza się do odpowiedniego zarządzania rejestrami. W celu ułatwienia tego zadania programiście, stworzono bibliotekę która opakowuje proces przypisywania bitów do odpowiednich miejsc w rejestrach i udostępnia zrozumiałe dla człowieka funkcje, struktury i predefiniowane wartości.

Biblioteka o której mowa to AT91LIB [18] v.1.5 zaprojektowana przez firmę Atmel na potrzeby mikrokontrolerów ARM. Dzięki jej zastosowaniu nie jest konieczne dokładne zaznajomienie się ze strukturą rejestrów mikrokontrolera, przez co programista może skupić swoją uwagę na implementacji konkretnego rozwiązania. Udostępnia ona także mechanizm informujący programistę o wykonanym przez niego błędzie logicznym. Przykładem takiego błędu może być próba wykorzystania urządzenia peryferyjnego obsługującego interfejs TWI bez wcześniejszej inicjalizacji zegara wymaganego przez to urządzenie. W przypadku wykrycia takiej sytuacji AT91LIB, o ile to możliwe, poinformuje nas o błędzie, a następnie przerwie wykonywanie programu.



Rysunek 4.10: Struktura katalogów biblioteki AT91LIB v.1.5

Biblioteka AT91LIB jest podzielona na 7 katalogów (rys. 4.10) odpowiedzialnych za zarządzanie poszczególnymi elementami związanymi z mikrokontrolerem. W tabeli 4.1 przedstawiono opis funkcjonalności obsługiwanych przez kod zawarty w poszczególnych katalogach biblioteki.

Tabela 4.1: Opis poszczególnych katalogów biblioteki AT91LIB v.1.5

Nazwa katalogu	Opis
boards	obsługa płyt ewaluacyjnych oraz modułów mikrokontrolerów
components	dodatkowe komponenty zewnętrzne takie jak np. kontroler ethernet
drivers	wyspecjalizowane wysokopoziomowe funkcje zarządzające działaniem urządzeń peryferyjnych
memories	obsługa różnego rodzaju pamięci
peripherals	funkcje niskiego poziomu zarządzające urządzeniami peryferyjnymi
usb	obsługa USB
utility	narzędzia oraz algorytmy dodatkowe

Zastosowanie AT91LIB pozwoliło na stworzenie przejrzystego kodu sterującego robotem, który będzie można modyfikować bez dokładnego zagłębiania się w notę katalogową mikrokontrolera AT91SAM7S256.

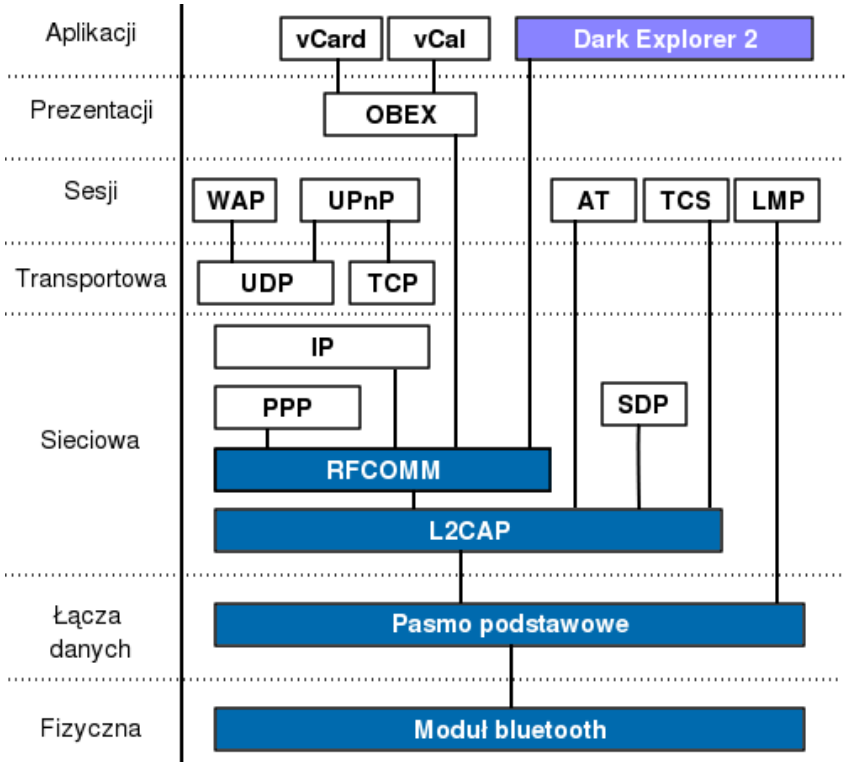
4.4. Protokół komunikacji bluetooth

Robot Dark Explorer komunikuje się z otaczającym go światem przy użyciu technologii Bluetooth. Zaimplementowany w pierwotnej wersji oprogramowania robota protokół komunikacji bazuje na ramce składającej się z 8 bitów nagłówka oraz 8 bitów danych. Za pomocą informacji zawartych w nagłówku robot rozpoznaje rodzaj akcji którą należy podjąć, natomiast po odczytaniu danych z kolejnego przesłanego bajtu urządzenie jest w stanie uruchomić odpowiedni wariant metody zdefiniowanej w nagłówku przesłanej ramki. Jeżeli wysłane polecenie wymusza odesłanie danych zwrotnych są one transmitowane przez robota w postaci czystego strumienia bajtów bez żadnych dodatkowych metadanych.

Rysunek 4.11: Schemat ramki komunikacyjnej w pierwotnej wersji robota

Zaprezentowane tutaj podejście jest bardzo wydajne i nie ogranicza efektywnej przepustowości łącza poprzez konieczność przesyłania dodatkowych danych związanych z obsługą protokołu komunikacji. Nie mniej jednak tego rodzaju komunikacja wymaga, od programisty tworzącego aplikacje klienckie, głębokiej znajomości sposobu realizacji poszczególnych poleceń, tak aby mógł on synchronizować komunikację zarówno pod względem czasowym jak i logicznym. Ma to szczególne znaczenie w przypadku wykonywania poleceń w sposób sekwencyjny gdzie aplikacja klienta powinna oczekiwać na zakończenie wykonania poprzedniego polecenia, jak ma to miejsce np. podczas pobierania obrazu z kamery. Kolejną znaczącą niedogodnością jest fakt, iż bardzo często nawet najdrobniejsze zmiany w oprogramowaniu robota wymuszają wprowadzanie zmian w aplikacji klienta pomimo tego, że sposób wymiany komunikatów pozostał niezmieniony. Swego rodzaju kłopotliwym problemem było również interpretowanie odpowiedzi przychodzących z robota, gdyż aplikacja klienta musiała przechowywać informację na temat rodzaju i formatu odpowiedzi która została odebrana w wyniku wykonania akcji. Co więcej dotychczasowy sposób wymiany danych nie gwarantował również mechanizmów wykrywania i zapobiegania błędom transmisji na poziomie aplikacji.

Zastosowana do wymiany danych technologia bluetooth posiada szereg standardowych protokołów komunikacyjnych działających w ramach różnych warstw modelu referencyjnego OSI[6]. Niestety protokoły dostępne w warstwie aplikacji ograniczają się jedynie do wymiany danych na temat kontaktów i synchronizacji danych kalendarzowych, a co za tym idzie nie nadają się do rozwiązania problemów zdalnego sterowania urządzeń. Pełna lista protokołów dostępnych w ramach technologii bluetooth widoczna jest na rysunku 4.12. Niezmiernie ważne jest aby przy projektowaniu schematu komunikacji w warstwie aplikacji, tak dobrać protokół warstwy niższej aby realizował on jak najwięcej wymagań postawionych przed systemem komunikacji.



Rysunek 4.12: Diagram protokołów bluetooth z podziałem na warstwy

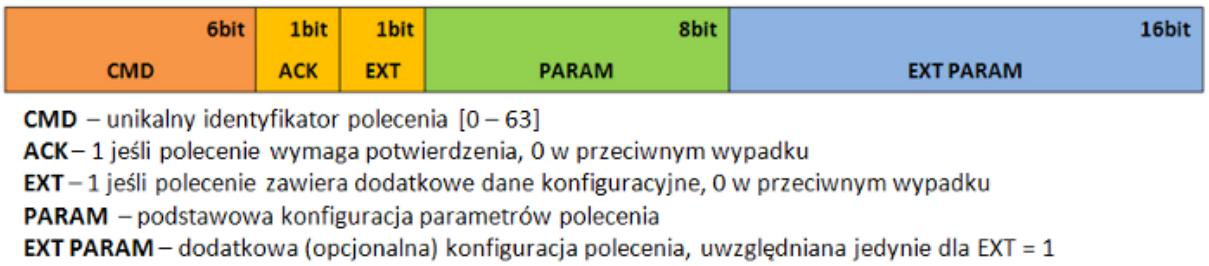
Aby zaadresować wszystkie napotkane w czasie rozwoju robota problemy konieczne okazało się zaprojektowanie i zaimplementowanie nowego protokołu warstwy aplikacji który ułatwiłby tworzenie oprogramowania współpracującego z rozbudowaną wersją robota Dark Explorer. Konieczność taka wynika z faktu iż w warstwie aplikacji nie istnieją protokoły które możnaby było zaadaptować na potrzeby sterowania robotem. Do implementacji protokołu komunikacji wykorzystany został protokół RFCOMM[7] gdyż gwarantuje on nam korekcję błędów na poziomie pakietów. Wszystkie powiązane z nim protokoły niższego poziomu zostały oznaczone kolorem na rysunku 4.12.

[6] OSI - Open Systems Interconnection - jest to model referencyjny standaryzujący zasady łączenia systemów otwartych, opisujący w szczególności strukturę komunikacji sieciowej

[7] Radio Frequency Communication

Projekt protokołu komunikacji bazuje na 16 bitowych ramkach bazowych. Każdy przesyłany komunikat musi posiadać 16 bitowy nagłówek w odpowiednim formacie który pozwoli na jego poprawne zinterpretowanie i przetworzenie. Komunikaty w niedozwolonym formacie lub zawierające nieprawidłowe dane będą przez robota ignorowane.

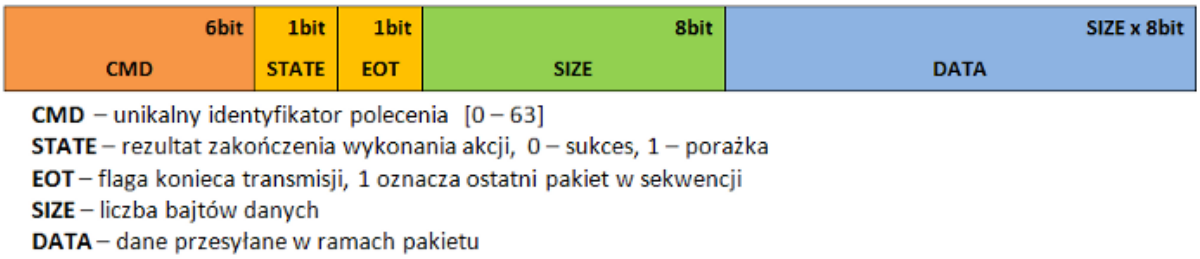
W ramach opracowanego standardu wyróżnić można dwa rodzaje ramek. Do pierwszej grupy zaliczyć można ramki żądań wysyłane przez urządzenia zewnętrzne w celu zlecenia robotowi wykonania manewru lub rozpoczęcia kolejnych etapów bardziej złożonej procedury. Diagram przedstawiający strukturę funkcjonalną ramki żądania widoczny jest na rysunku 4.13.



Rysunek 4.13: Schemat funkcjonalny ramki komunikacyjnej z żądaniem

Pierwsze 6 bitów zostało zarezerwowane na unikalny identyfikator polecenia które robot ma wykonać. Pozwala to na zdefiniowanie 64 niezależnych funkcjonalnie komend w ramach których, w wersji podstawowej, możliwe jest uruchomienie 256 niezależnych wariantów polecenia. W przypadku wykorzystania wersji rozszerzonej nagłówka programista ma do dyspozycji 24 bity które może przeznaczyć na dane polecenia i identyfikator wariantu komendy w proporcjach odpowiadających wymaganiom programisty. Kolejny bit nagłówka przeznaczony został na flagę potwierdzenia. Umieszczenie 1 na wspomnianym bicie spowoduje iż robot po zakończeniu wykonywania żądania prześle ramkę potwierdzającą ze statusem wykonania akcji. Ułatwia to programiście wykonywanie czynności sekwencyjnych takich jak np. wykonanie zdjęcia za pomocą kamery i przesłanie go do aplikacji klienta. Ósmy bit nagłówka zarezerwowany został na flagę z informacją o użyciu rozszerzonej wersji nagłówka. Wymuszenie 1 na tym bicie spowoduje, że robot będzie oczekiwał na dodatkowe 16 bitów danych żądania które mogą zostać wykorzystane do przesłania bardziej skomplikowanej konfiguracji wykonania polecenia. Kolejne osiem bitów stanowi tzw. podstawową konfigurację polecenia, która może służyć jako identyfikator przy uruchamianiu odpowiedniego wariantu polecenia lub jako nośnik danych potrzebnych do zrealizowania przesłanej komendy. W przypadku gdyby rozmiar bufora konfiguracyjnego nie pozwalał na przechowanie wszystkich wymaganych danych programista może wykorzystać dodatkowe 2 bajty konfiguracji rozszerzonej.

Drugi rodzaj ramek stanowią powiadomienia informujące użytkownika o rezultacie wykonania przesłanej akcji lub aktualnym stanie robota. Podobnie jak w przypadku ramki z żądaniem pierwsze 6 bitów zarezerwowane jest na unikalny identyfikator polecenia dla którego przesyłana jest odpowiedź. Kolejny bit zawiera flagę informującą o rezultacie zakończenia akcji. Prawidłowe zakończenie wykonania polecenia sygnalizowane jest ustawieniem zera na wspomnianym bicie. Wystąpienie błędu podczas realizacji polecenia lub przesłanie komendy w nieprawidłowym formacie spowoduje ustawienie 1 na bicie STATE. W przypadku gdy ilość danych do przesłania przekracza maksymalny dopuszczalny rozmiar pojedynczej ramki możliwe jest podzielenie danych na fragmenty i przesłanie ich za pomocą sekwencji komunikatów. Koniec sekwencji sygnalizowany jest za pomocą bitu EOT[8]. Dostępność tego rodzaju trybu pozwala na szybką transmisję większych ilości danych bez konieczności wysyłania żądań dla poszczególnych fragmentów.



Rysunek 4.14: Schemat funkcjonalny ramki komunikacyjnej z odpowiedzią

Kolejne 8 bitów nagłówka odpowiedzi stanowi informacja o ilości bajtów przesyłanych w sekcji danych ramki. Ogranicza to maksymalny rozmiar ramki przez co transmisja danych nie powoduje licznych retransmisji które bardzo często pojawiają się w przypadku transmisji większych ilości danych. Informacja o szerokości pola danych ramki może również służyć jako suma kontrolna pozwalająca na sprawdzenie czy wszystkie zadeklarowane dane zostały prawidłowo odebrane. Pozwala to programiście tworzącemu oprogramowanie klienta na monitorowanie transmisji i ewentualne reagowanie w przypadku pojawiania się problemów z jej płynnością.

[8] End of Transmission

4.5. Algorytm rekonstrukcji ścieżki powrotnej

Problem śledzenia aktualnego położenia na przykładzie robota mobilnego jest często poruszany. W poszukiwaniu rozwiązań tego problemu badacze oraz inżynierowie rozwinęli szeroką gamę systemów, sensorów i technik wyszukiwania aktualnej pozycji robota mobilnego. Można wyszczególnić siedem kategorii systemów do pozycjonowania:

1. Pozycjonowanie relatywne:
   - odometria
   - nawigacja inercjalna
2. Pozycjonowanie absolutne:
   - wykorzystanie kompasów magnetycznych
   - pozycjonowanie z zewnętrznym sygnałem nawigacyjnym (Active Beacon)
   - system globalnego pozycjonowania (GPS)
   - nawigacja przy pomocy zewnętrznego znacznika (Landmark Navigation)
   - wykrywanie pozycji na podstawie przygotowanych map (Map Matching)

Poniżej zostaną pokrótce opisane wyszczególnione techniki pozycjonowania robotów mobilnych.

Odometria bazuje na obliczaniu położenia robota przy pomocy wzorów transformujących informacje o obrotach jego kół na przemieszczenie relatywne w stosunku do podłoża. Rozwiązanie wydaje się być proste w realizacji oraz skuteczne, jednak posiada kilka wad. Podczas stosowania tej metody nie jest możliwe określenie położenia robota gdy nie dotyka on podłoża. Problemem jest również ślizganie się kół robota, co będzie skutkowało błędami podczas obliczania jego położenia.

Nawigacja inercjalna wykorzystuje czujniki przyspieszenia oraz żyroskopy w celu pomiaru odpowiednio przyspieszenia i prędkości kątowej robota. Dane te są następnie całkowane raz (lub w przypadku akcelerometru dwa razy) w celu wyznaczenia pozycji robota. Zaletami tej metody jest brak jakiejkolwiek zależności od świata zewnętrznego. Wszystkie pomiary są wykonywane bez użycia jakichkolwiek zewnętrznych punktów odniesienia, poza podaniem początkowej prędkości i punktu startowego robota. Niestety błędy wynikające niedoskonałości czujników oraz z całkowania, kumulują się w raz z

upływem czasu. Uniemożliwia to wykorzystanie nawigacji inercjalnej w zastosowaniach, wymagających określania pozycji robota w dłuższych okresach czasu.

Kompasy magnetyczne są wykorzystywane podczas stosowania metody pozycjonowania absolutnego tj. takiego, które w przeciwieństwie do pozycjonowania relatywnego nie wymaga informacji o położeniu początkowym robota. Magnetometry stosowane w kompasach pozwalają uzyskać informacje o zwrocie i kierunku robota. Próba wykorzystania magnetometru do pozycjonowania robota w pomieszczeniach nie jest jednak dobrym pomysłem. Magnetometr jest podatny na zakłócenia pola magnetycznego, generowane przez instalacje elektryczną oraz metalową konstrukcję budynku.

Pozycjonowanie z zewnętrznym sygnałem nawigacyjnym (Active Beacon) jest powszechnym sposobem wspomagania nawigacji na statkach oraz w samolotach. Technika ta jest również wykorzystywana w komercyjnych rozwiązaniach robotów mobilnych. Polega ona na ustalaniu pozycji robota, przy wykorzystaniu sygnałów zewnętrznych, jednoznacznie określających jego położenie na podstawie odległości od nadajników. Takie podejście jest bardzo dokładne, jednak wymaga przygotowania odpowiedniej infrastruktury, która może być bardzo kosztowna.

System globalnego pozycjonowania (GPS) jest odpowiednikiem metody Active Beacon na skalę globalną. W tym przypadku infrastruktura w postaci satelitów okołoziemskich jest udostępniana między innymi przez rząd Stanów Zjednoczonych. Eliminuje to konieczność tworzenia własnej infrastruktury tak jak w przypadku techniki Active Beacon. Niestety odbiorniki GPS nie są w stanie odebrać sygnału z satelitów wewnątrz pomieszczeń, co eliminuje tą metodę w przypadku zastosowań dla robotów mobilnych pracujących w budynkach.

Nawigacja przy pomocy zewnętrznego znacznika (Landmark Navigation) polega na wykrywaniu i rozpoznawaniu przez robota obiektów, będących kształtami geometrycznymi (np. linie, okręgi, prostokąty), których położenie jest dobrze znane lub zakodowane w samym znaczniku, np. przy pomocy kodu kreskowego lub QR Code'u[9]. Znaczniki można podzielić na dwa rodzaje: naturalne oraz stworzone specjalnie do spełnienia wymagań postawionych przez tą metodę pozycjonowania.

Wykrywanie pozycji na podstawie przygotowanych map (Map Matching), wykorzystuje czujniki wbudowane w robocie mobilnym w celu stworzenia mapy lokalnego otoczenia. Mapa ta jest następnie porównywana z wcześniej zapisaną przez robota mapą globalną. Jeżeli robot przyporządkuje zbudowaną przez siebie mapę lokalną do elementu

[9] QR Code – dwuwymiarowy kod kreskowy wynaleziony przez japońską firmę Denso-Wave w 1994 roku

mapy globalnej, może na tej podstawie obliczyć swoje aktualne położenie. Zaletą tej metody, jest wykorzystywanie naturalnych struktur znajdujących się wewnątrz budynku w celu określenia położenia oraz to, że robot mobilny może sam tworzyć nowe mapy i na ich podstawie określać swoją pozycję. W celu znalezienia punktów charakterystycznych podczas porównywania dwóch map, wymagana jest duża różnorodność struktury środowiska w którym porusza się robot mobilny, co jest dużym minusem. Konieczne jest także stosowanie bardzo dokładnych czujników budujących mapy.

4.5.1. Wybrane rozwiązanie

Ostateczne rozwiązanie zastosowane w robocie Dark Explorer jest połączeniem odometrii oraz nawigacji inercjalnej. Robot wykorzystuje wbudowany żyroskop w celu określenia kierunku w którym się porusza oraz akcelerometru do określenia dystansu o jaki się przesunął. Rozwiązanie to pozwala na określenie przybliżenia pozycji relatywnej względem punktu startowego. Metoda ta została wybrana ze względu na brak konieczności przygotowywania zewnętrznej infrastruktury, co zagwarantowało uniwersalność rozwiązania.

Robot Dark Explorer został wyposażony w czujniki, które mają na celu dostarczenie informacji niezbędnych do ustalenia toru ruchu robota. Niniejszy podrozdział opisuje algorytm wykorzystujący dane z czujników w celu wyznaczenia trasy od obecnego położenia do miejsca z którego robot został przyniesiony przez operatora.

Algorytm rekonstrukcji ścieżki powrotnej składa się z dwóch części. Pierwsza z nich odpowiedzialna jest za zapamiętywanie toru ruchu robota, druga natomiast za odtworzenie ścieżki na podstawie zgromadzonych danych. Cały algorytm opiera się na informacjach uzyskanych z dwóch czujników: żyroskopu oraz akcelerometru. Czujnik przyspieszenia został zastosowany w celu określenia odległości jaką przebył robot niesiony przez operatora. Natomiast żyroskop pozwala na uzyskanie informacji o zmianie kierunku.

4.5.2. Rejestracja trasy

Aby możliwe stało się zrealizowanie implementacji algorytmu rekonstrukcji ścieżki powrotnej konieczne jest w pierwszej kolejności zapamiętanie wszystkich punktów charakterystycznych trasy po której poruszał się operator.

Pierwotnym podejściem do rozwiązania problemu wykrycia toru ruchu ciała przenoszonego przez operatora było wyznaczanie przemieszczenia na podstawie informacji o zmianach przyspieszenia uzyskanej z akcelerometru. W celu obliczenia drogi po jakiej

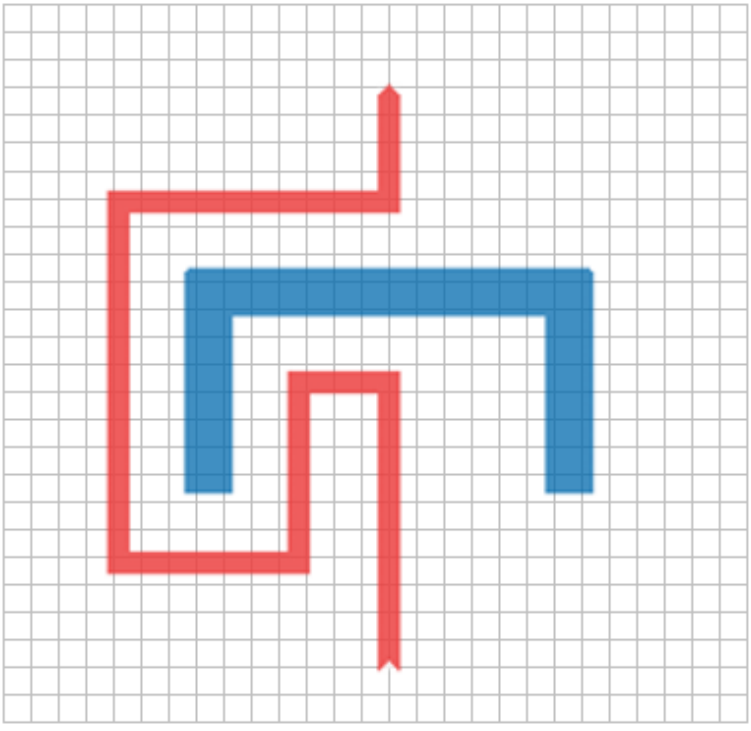
poruszało się ciało, mając jedynie informacje o przyspieszeniu, konieczne jest wykonanie dwukrotnego całkowania wartości otrzymanych z czujnika. Niestety operacja ta wymaga przeprowadzania pomiarów w bardzo krótkich odstępach czasu w celu zminimalizowania błędów całkowania. Nie mniej istotna jest także czułość i dokładność samego czujnika przyspieszenia. Po wykonaniu wstępnych testów tego rozwiązania stwierdzono iż otrzymywane rezultaty nie są zadowalające i nie pozwalają na stworzenie stabilnego rozwiązania w oparciu o opisywane podejście.

Rozwiązaniem pozwalającym na uzyskanie satysfakcjonujących rezultatów okazała się metoda wykrywania ilości kroków które wykonał operator robota podczas przemieszczania go w inne miejsce. Użytkownik korzystając z aplikacji sterującej rozpoczyna proces nagrywania. Podczas wykonywania kroków, operator robota wykonuje mimowolne ruchy ręką w górę i w dół które są rejestrowane przez czujniki robota. Dzięki wykrywaniu odpowiedniej sekwencji przyspieszeń jesteśmy w stanie określić czy operator wykonał krok. Po wykryciu każdego kolejnego kroku zapisywana jest wartość kąta pomiędzy kierunkiem ruchu z poprzedniego i obecnego kroku. Prezentowane podejście pozwala na zapisanie całej trasy w postaci sekwencji zmian kierunku następujących po przemieszczeniu się robota o odległość jednego kroku. Rozwiązanie to pozwala zarejestrowanie wszystkich punktów charakterystycznych trasy przy jednoczesnym ograniczeniu ilości danych potrzebnych do stworzenia jej logicznej reprezentacji.

Algorytm wykrywania kroków zrealizowany został w oparciu o zainstalowany w robocie akcelerometr trójosiowy. Szczegółowy opis zasady działania akcelerometru oraz sposobu implementacji procedury zliczania kroków zamieszczony został w rozdziale 3.2. Informacje na temat kąta obrotu jaki wykonał użytkownik pomiędzy dwoma punktami charakterystycznymi trasy mogą być pobierane z jednego z dwóch czujników. Domyślnym czujnikiem jest żyroskop, który pozwala z niezwykłą dokładnością wykonywać pomiary prędkości kątowej, a co za tym idzie umożliwia precyzyjne wyznaczenie kąta o jaki dokonany został obrót. Więcej informacji na temat zasady działania żyroskopu można znaleźć w rozdziale 3.3. Sensorem zapasowym jest magnetometr. Pozwala on wyznaczyć obecną orientację urządzenia względem bieguna magnetycznego ziemi. Robot jest w stanie na podstawie danych otrzymywanych z kompasu jednoznacznie wyznaczyć kąt o jaki został obrócony. Z zasadami działania magnetometru oraz napotkanymi problemami które dyskryminują ten czujnik jako podstawowy przy rejestrowaniu i odtwarzaniu toru ruchu można zapoznać się w rozdziale 3.4.

4.5.3. Rekonstrukcja trasy

Użytkownik sygnalizuje zakończenie nagrywania trasy za pomocą odpowiedniego komunikatu wysłanego za pośrednictwem aplikacji sterującej. Robot poinformuje użytkownika o zakończeniu nagrywania ścieżki wyświetlając na ekranie LCD informacje na temat liczby kroków zapamiętanych w trakcie ostatniej sesji. Następnie operator, ustawia robota na podłożu w kierunku którym robot będzie miał wracać. Rozpoczęcie procedury odtwarzania zapamiętanego toru ruchu rozpocznie się po wysłaniu polecenia inicjalizującego algorytm rekonstrukcji ścieżki powrotnej. Robot korzystając z informacji dostarczanych na bieżąco z żyroskopu oraz danych zapisanych w postaci nagranej ścieżki będzie starał się powrócić do miejsca z którego wystartował użytkownik. Dodatkowo, jeżeli na zapamiętanej ścieżce pojawi się przeszkoda uniemożliwiająca odtworzenie ścieżki, zostanie ona wykryta za pomocą dołączonych czujników odległości i rozpocznie się procedura mająca na celu uniknięcie kolizji z przeszkodą. W chwili gdy robot ominie przeszkodę będzie starał się, jeżeli to możliwe, powrócić na oryginalną ścieżkę ruchu. Procedura ta ma na celu zagwarantowanie, że robot prawidłowo dotrze do miejsca z którego wyruszył. W przypadku gdy robot stwierdzi, że powrót na oryginalną ścieżkę nie jest możliwy lub ominięcie przeszkody nie jest możliwe procedura rekonstrukcji zostanie automatycznie zakończona. Na rysunku 4.15 zobrazowany został sposób ominięcia przeszkody która znalazła się na drodze rekonstrukcji ścieżki.



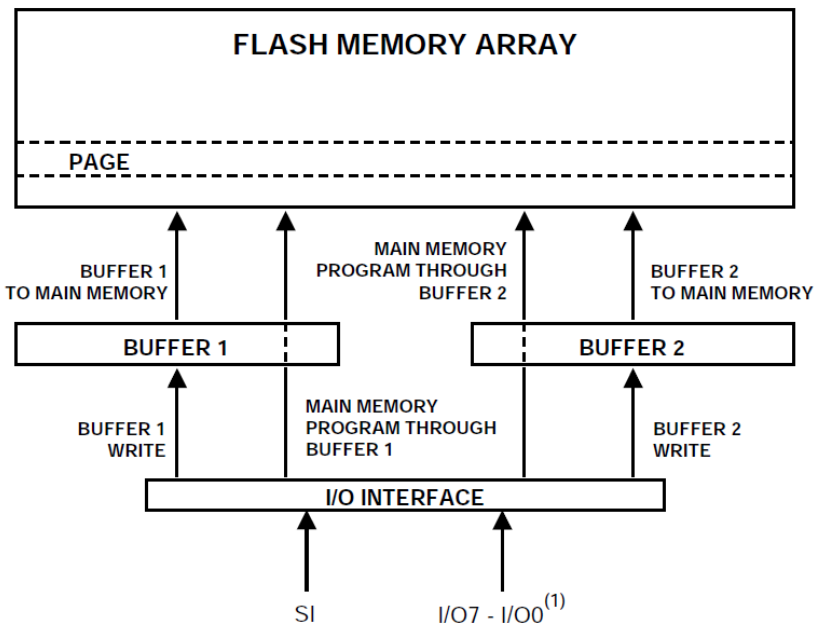
Rysunek 4.15: Diagram obrazujący ominięcie napotkanej przeszkody

Algorytm pozwalający robotowi uniknąć zderzenia z przeszkodą opisany został ze wszystkimi szczegółami w ramach rozdziału poświęconego czujnikom odległości. Opisany w ramach rozdziału 3.5 algorytm omijania przeszkód został, na potrzeby implementacji procesu rekonstrukcji ścieżki, uzupełniony o wywoływaną w sposób rekurencyjny metodę pozwalającą robotowi powrócić na oryginalną ścieżkę po ominięciu przeszkody. Dodatkowym zabezpieczeniem gwarantującym zakończenie procedury rekonstrukcji trasy, jest zatoczenie przez robota pełnego koła podczas próby omijania przeszkód lub przekroczenie zdefiniowanego progu złożoności przeszkody szacowanego na podstawie liczby rekurencyjnych wywołań potrzebnych do powrotu na pierwotny tor jazdy.

4.6. Modernizacja sposobu pobierania obrazu z kamery

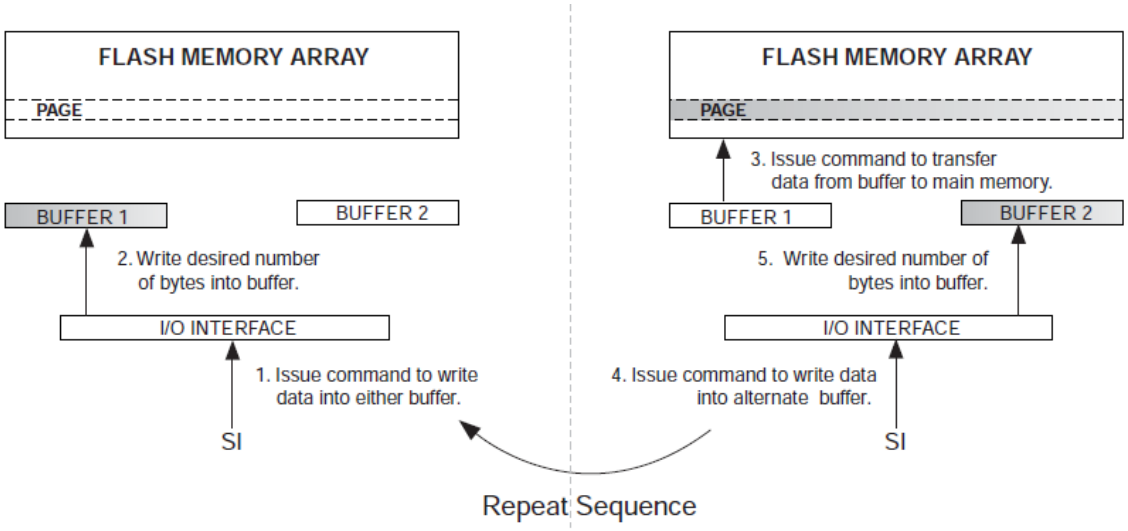
W celu zwiększenia możliwości robota Dark Explorer w dziedzinie przetwarzania obrazu niezbędne było polepszenie rozdzielczości zdjęć otrzymywanych z wbudowanej kamery. W wersji bazowej maksymalne rozdzielczości z jakimi było możliwe pobieranie obrazów to 160x100 pikseli w kolorze oraz 320x200 pikseli w odcieniach szarości. Wartości te zostały najprawdopodobniej narzucone przez ograniczenia pamięci podręcznej mikrokontrolera ARM, który posiada jedynie 64kB szybkiej pamięci SRAM. Ilość ta wystarczyła na pobranie maksymalnie $320 * 200 = 64000$ bajtów danych pozostawiając 1536 bajtów na zmienne niezbędne do poprawnego działania oprogramowania systemu wbudowanego.

Wykorzystywana kamera posiada osiem wyjść równoległych odpowiadających za jeden bajt danych. Wszystkie osiem bitów na wyjściu zmienia się z każdym taktem zegara sterującego kamery, podanego na jej wejście. W taki sposób z każdym cyklem zegara, kamera oddaje do naszej dyspozycji dane z kolejnej porcji obrazu. W konfiguracji początkowej zegar wejściowy kamery był tworzony w sposób czysto programowy. Jedno z wyjść GPIO mikrokontrolera było ustawiane na przemian raz w stan wysoki, a raz w stan niski. Niewątpliwie podejście to znacząco ułatwia synchronizację pomiędzy sygnałem zegarowym a momentem pobierania danych eliminując możliwość wczytania danych z wyjścia kamery w momencie w którym wyjścia te są w stanie nieustalonym.



Rysunek 4.16: Schemat struktury logicznej AT45DB321B wraz z zaznaczonymi operacjami zapisu [19].

Przedstawiony problem niedostatecznej ilości pamięci został rozwiązany poprzez wykorzystanie pamięci Data Flash wbudowanej w moduł mikrokontrolera. Użyty układ AT45DB321B [20] dostarcza 32 megabitów pamięci którą możemy zarządzać poprzez interfejs szeregowy SPI[10]. Zastosowana pamięć posiada dwa bufory po 528 bajtów pojemności. Struktura logiczna AT45DB321B wraz z zaznaczonymi operacjami zapisu została przedstawiona na rysunku 4.16. Podczas przenoszenia danych z jednego bufora do pamięci Data Flash możliwy jest zapis informacji do drugiego bufora. W ten sposób jest emulowany zapis danych do pamięci Data Flash w trybie ciągłym. Zarys algorytmu ciągłego zapisu danych do AT45DB321B przedstawia schemat na rysunku 4.17



Rysunek 4.17: Schemat procedury ciągłego zapisu do układu AT45DB321B [19].

Czas zapisu danych na stronie pamięci układu AT45DB321B nie jest stały. Wymusza to zmianę podejścia do zegara sterującego kamerą z koncepcji programowej na sprzętową, tak aby okres zegara był stały. Niejednorodny zegar spowodowałby przebarwienia na obrazie wynikające ze sposobu działania mechanizmu dobierania ekspozycji, wbudowanego w kamere PO6030K. W celu uzyskania sprzętowego zegara zostało użyte urządzenie peryferyjne mikrokontrolera AT91SAM7S generujące sygnał prostokątny o określonej częstotliwości. Z powodu multipleksowania tego urządzenia z jednym z pinów GPIO mikrokontrolera podłączonego do wyjścia danych z kamery, konieczne było zastosowanie przeplotu w taśmie podłączeniowej kamery w celu dostarczenia odpowiednich sygnałów do prawidłowych wejść/wyjść.

Pojawiły się również problemy podczas uruchamiania interfejsu szeregowego SPI za pomocą którego mikrokontroler komunikuje się z układem AT45DB321B. Okazało się bowiem, że wyjścia/wejścia odpowiedzialne za obsługę tego interfejsu są multipleksowane

[10] SPI – Serial Peripheral Interface Bus

z wyjściami kontrolera PWM[11] odpowiedzialnego za odpowiednie wysterowanie silników robota. W konfiguracji podstawowej silniki te były kontrolowane przez 4 niezależne sygnały PWM pozwalające na obracanie się każdego koła robota z inną prędkością. Stwierdzono, iż taka funkcjonalność nie jest niezbędna i wykorzystano trzy zbędne wyjścia sygnałów PWM do obsługi interfejsu SPI, a przy pomocy własnoręcznie wykonanej zworki dostarczono jeden sygnał PWM dla wszystkich czterech silników.

Dane z kamery są początkowo umieszczane w pamięci SRAM do momentu uzbierania grupy 512 bajtów (rozmiar strony pamięci układu AT45DB321B). Zabieg ten jest wykonywany w celu poświęcenia jak najmniejszej ilości czasu na zapis do pamięci Data Flash. Przygotowana grupa danych jest następnie przesyłana do AT45DB321B poprzez interfejs SPI z wykorzystaniem mechanizmu DMA[12].

Wszystkie podjęte kroki pozwoliły na odbiór obrazu o maksymalnych rozdzielczościach oferowanych przez układ PO6030K to znaczy 640x480 pikseli w odcieniach szarości oraz 640x480 pikseli w kolorze.

[11] PWM – Pulse Width Modulation
[12] DMA - Direct Memory Access

4.7. Lokalizacja twarzy na obrazie

W ciągu ostatnich lat obserwuje się bardzo dynamiczny rozwój biometrii. Aplikacje analizujące biometryczne cechy użytkowników znalazły swoje zastosowanie między innymi w systemach zabezpieczeń i monitoringu, robotyce, komputerach przenośnych, aparatach i kamerach cyfrowych czy nawet w nowoczesnych telefonach komórkowych. Pośród szeregu cech które mogą zostać poddane analizie, szczególnym zainteresowaniem cieszy się analiza ludzkiej twarzy. Systemy posiadające możliwość zlokalizowania na obrazie twarzy od pewnego czasu towarzyszą nam w życiu codziennym. Niemniej jednak większość istniejących algorytmów pozwalających na rozwiązanie tego problemu jest dosyć skomplikowana i ma bardzo konkretne wymagania zarówno co do platformy sprzętowej jak i jakości obrazu który będzie poddawany analizie. Dlatego też stworzenie implementacji dla platformy o bardzo ograniczonych zasobach obliczeniowych i pamięciowych jest dużym wyzwaniem. Celem niniejszego rozdziału jest przedstawienie podstawowych rodzajów algorytmów lokalizacji twarzy oraz wskazanie ich wad i zalet. W dalszej części znajduje się szczegółowy opis stworzonej implementacji algorytmu zastosowanego w realizowanej pracy magisterskiej.

4.7.1. Algorytmy

W ciągu ostatnich kliku lat rozwijanych było wiele różnych podejść poprawiających wydajność procesu lokalizacji ludzkiej twarzy. Wśród nich można wyróżnić klika głównych kategorii. Pierwszą z nich jest metoda opierająca się o bazę wiedzy. Metoda ta pozwala na odszukiwanie niezmiennych cech twarzy nawet w bardzo skomplikowanym otoczeniu, a przez to pozwala na ustalenie pozycji twarzy. Tego typu podejście do problemu eliminuje problemy z uwzględnianiem ułożenia i pozycji twarzy na obrazie. Statystyczny opis zależności pomiędzy poszczególnymi cechami twarzy pozwala na stworzenie modelu który może posłużyć do zbudowania szablonu twarzy dzięki któremu będziemy w stanie jednoznacznie stwierdzić czy na obrazie widoczna jest twarz czy nie. Z takiego ujęcia problemu wywodzi się kolejna metoda nazywana metodą porównywaniu szablonów (ang. template matching). Do stworzenia wspomnianego szablonu twarzy mogą posłużyć takie parametry jak na przykład kolor skóry, faktura twarzy czy też budowa twarzy. Wśród metod bazujących na porównywaniu szablonów można wyróżnić dwa podejścia. Pierwsze z nich oparte jest bezpośrednio na opisie cech twarzy i korzysta z opisu zależności pomiędzy parametrami twarzy jako szablonu, odszukując najbardziej pasujący do szablonu fragment obrazu. Drugim podejściem jest stosowanie globalnego szablonu bazującego na metrykach będących sumą poszczególnych cech za pomocą których jest opisany szablon. Pierwsze

podejście wymaga obrazu o stosunkowo dużej rozdzielczości ale ponieważ nie bazuje ono na wszystkich punktach obrazu będzie obliczeniowo bardziej wydajne niż podejście oparte o szablon globalny które może wymagać przeszukania potencjalnie dużo większej grupy punktów obrazu wejściowego. Główną wadą przestawionych do tej pory rozwiązań są ich dosyć wysokie wymagania. Każdy z omówionych do tej pory algorytmów wymagał wykonywania kosztownych obliczeń mających na celu, poprzez dokonywanie przekształceń, dopasowanie modelu szablonu do zawartości obrazu wejściowego. Tak więc wykorzystanie tego typu algorytmów wiąże się z koniecznością użycia stosunkowo mocnych jednostek obliczeniowych, co w przypadku platformy embeded mogłoby być problematyczne.

Zupełnie innym podejściem do problemu lokalizacji twarzy jest wykorzystywanie koloru skóry do odszukania fragmentów obrazu które potencjalnie mogłyby zwierać twarz. Rozwiązania oparte o analizę kolorów można podzielić na dwie grupy. W pierwszej znajdują się rozwiązania bazujące na jednolitym kolorze tła, w drugiej natomiast obraz wejściowy może zawierać tło o dowolnym stopniu złożoności. Pierwsza z przedstawionych grup w znakomity sposób upraszcza problem odnalezienia twarzy gdyż często usunięcie koloru tła z obrazu gwarantuje nam odnalezienie wszystkich twarzy, zakładając, że obraz zawiera jedynie twarze. Jednakże jest to podejście nakładające duże ograniczenia na warunki w jakich odbywa się akwizycja obrazu. Dlatego też znacznie częściej porusza się problem lokalizacji twarzy w oparciu o analizę kolorów, z użyciem obrazów zawierających niejednokrotnie tło o bardzo wysokim stopniu złożoności. Niemniej jednak wszystkie rozwiązania bazujące jedynie na segmentacji kolorów są w bardzo wysokim stopniu zależne od warunków w jakim pobierany jest obraz wejściowy. Ze względu na zmieniające się warunki otocznia może okazać się, że nasz algorytm nie uwzględnia wszystkich kolorów skóry. Dodatkowym utrudnieniem jest pojawianie się grup pikseli które zostały zakwalifikowane jako kolor skóry pomimo, że z twarzą nie mają nic wspólnego. Problem ten jest szczególnie uciążliwy w przypadku obrazów o małych rozmiarach, gdyż trudno jest wtedy odróżnić twarz od szumów powstałych wskutek warunków otocznia, w których obraz był pobierany. Pomimo licznych swoich licznych wad segmentacja kolorów jest jedną z najszybszych metod klasyfikacji obrazów pod kątem obecności twarzy na obrazie. Co więcej narzut obliczeniowy na zastosowanie klasyfikatora bazującego o model kolorów skóry jest proporcjonalny do rozmiarów pobranego obrazu. Niestety jak się okazuje w praktyce sama segmentacja kolorów jest niewystarczająca i wymaga użycia dodatkowych metod w celu potwierdzenia odnalezienia twarzy.

Kolejnym ciekawym rozwiązaniem pozwalającym na odszukanie twarzy na ruchomym obrazie jest analiza ruchów twarzy. W tego typu rozwiązaniach obecność twarzy w sekwencji wideo wykrywana jest na podstawie detekcji ruchów charakterystycznych

dla ludzkiej twarzy, takich jak na przykład ruchy powiek. Mruganie jest ruchem na tyle specyficznym, że wykrycie go pozwala jednoznacznie wnioskować o lokalizacji twarzy na obrazie. Niestety przetwarzanie kilkudziesięciu klatek na sekundę o rozmiarach pozwalających na zarejestrowanie ruchu powiek stawia dosyć wygórowane wymagania związane nie tylko z algorytmem ale również z ilością danych jakie klasyfikator musi przetworzyć w ciągu każdej sekundy nagrania. Istnieje jeszcze wiele innych rozwiązań bazujących na bardzo wyszukanych metodach, jak na przykład użycie sieci neuronowych, jednakże ze względu na ich wygórowane wymagania, wykorzystanie jednego z nich dla platformy embeded z ograniczonymi zasobami obliczeniowymi i pamięciowymi jest niezwykle trudne, a w niektórych przypadkach niemal całkowicie niemożliwe.

Algorytm implementowany w ramach pracy magisterskiej bazuje na podejściu opartym o segmentację kolorów. Aby wyeliminować niedoskonałości płynące z tego podejścia algorytm w ostatnie fazie wykorzystuje informację o najprostszych cechach twarzy w celu potwierdzenia obecności twarzy w obszarach zaklasyfikowanych jako kolor skóry. Zastosowanie tego typu połączenia dwóch technik zaowocowało postaniem algorytmu nie tylko szybkiego ale zarazem dającego stosunkowo dobre wyniki analizy.

4.7.2. Implementacja

Implementacja modułu do lokalizacji twarzy na obrazie pobranym z kamery robota została w całości oparta o algorytm zaproponowany w artykule zatytułowanym 'Simple Face-detection Algorithm Based on Minimum Facial Features' [21]. Autorzy do rozwiązania problemu używają obrazów o małej rozdzielczości opisanych za pomocą modelu kolorów RGB[13].

Zgodnie ze wskazówkami znalezionymi we wspomnianej publikacji, pierwszym krokiem na drodze do zlokalizowania twarzy na obrazie jest akwizycja danych z kamery. Przesłany z kamery obraz automatycznie jest poddawany podstawowej korekcji mającej na celu dopasowanie balansu bieli oraz dopasowanie ostrości. Tak przygotowany obraz poddawany jest binaryzacji. Celem wspomnianego procesu jest odnalezienie na obrazie obszarów będących w kolorze skóry oraz włosów. Zanim jednak detekcja interesujących nas obszarów będzie możliwa, konieczne jest przeprowadzenie normalizacji modelu RGB. Zabieg tego typu pozwoli na usunięcie zakłóceń wywołanych przez światła i cienie na obrazie pobranym z kamery robota.

[13] RGB (z ang. Red Green Blue) - popularny w grafice model przestrzeni barw

Opisany proces normalizacji przeprowadzany jest poprzez pobranie wartości poszczególnych kanałów RGB danego piksela obrazu, a następnie podzielenie tak otrzymanej liczby przez sumę wartości wszystkich kanałów danego punktu. Korzystając z takiego przepisu otrzymujemy zestaw równań 4.1, 4.2, 4.3 opisujący normalizację każdego kanału z palety RGB.

$$r = \frac{R}{R+G+B} \tag{4.1}$$

$$g = \frac{G}{R+G+B} \tag{4.2}$$

$$b = \frac{B}{R+G+B} \tag{4.3}$$

Na tak przygotowanym obrazie można już podjąć próbę odszukania punktów w kolorze skóry i włosów. Do tego celu użyte zostały równania zaproponowane przez autorów pracy źródłowej. Określają one rozkład kolorów ludzkiej skóry i włosów wyznaczonych, jak twierdzą autorzy, w sposób eksperymentalny. Bazując na takich założeniach ustalony został górny $F_1(r)$ i dolny $F_2(r)$ zakres kolorów w jakich może znaleźć się ludzka skóra (rów. 4.4, 4.5).

$$F_1(r) = -1.376r^2 + 1.0743r + 0.2 \tag{4.4}$$

$$F_2(r) = -0.776r^2 + 0.5601r + 0.18 \tag{4.5}$$

Należy jednak zauważyć, że w zdefiniowanych w ten sposób równaniach mieści się również kolor biały $r = g = 0.33$ dlatego też należy dodać następujący warunek (rów. 4.6) usuwający niechciany kolor z obszaru zainteresowania.

$$w = (r - 0.33)^2 + (g - 0.33)^2 > 0.001 \tag{4.6}$$

Po połączeniu obydwu równań otrzymujemy następujący układ równań opisujący zakres kolorów jakie powinny reprezentować ludzką skórę.

$$S = \begin{cases} 1 & \text{dla } g < F_1(r) \cap g > F_2(r) \cap w > 0.001 \\ 0 & \text{dla pozostałych} \end{cases} \tag{4.7}$$

Praktyczne doświadczenia z wykrywaniem obszarów skóry z zastosowaniem równania 4.7 wykazały iż niemal wszystkie obszary o kolorze skóry zostały prawidłowo zaklasyfikowane. Znakomita większość pikseli nie stanowiących koloru skóry została zastąpiona kolorem

czarnym. Niemniej jednak wiele pikseli o kolorze niebieskim, żółtym i szarym również została błędnie sklasyfikowana.

Aby wyeliminować tego rodzaju efekt należy wykorzystać kryteria klasyfikacji oparte o przestrzeń kolorów HSI [14]. W równaniu 4.8 przedstawiono zależność pomiędzy modelem RGB a HSI.

$$\begin{gathered}\phi = cos^{-1}\left\{\frac{0.5[(R-G)+(R-B)]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right\} \\ \begin{cases} H = \phi & \text{dla B } \leq G \\ H = 360^\circ - \phi & \text{dla B} > G \end{cases}\end{gathered} \tag{4.8}$$

Po skorzystaniu z przestrzeni HSI otrzymujemy następujący przepis (rów. 4.9) na kolor skóry.

$$S = \begin{cases} 1 & \text{dla } g < F_1(r) \cap g > F_2(r) \cap w > 0.001 \cap (H > 240 \cup H \leq 20) \\ 0 & \text{dla pozostałych} \end{cases} \tag{4.9}$$

Przy tak zdefiniowanym klasyfikatorze dokonywana jest binaryzacja całego obrazu. Po przeprowadzeniu binaryzacji przeprowadzana jest kwantyzacja obrazu za pomocą próbki o rozmiarze i wysokości pięciu pikseli. Pozwala to zmniejszyć rozdzielczość obrazu, a co za tym idzie wyeliminować pewne rodzaje szumu oraz znacznie uprościć dalsze obliczenia. Wspomniany proces kwantyzacji polega na przeliczeniu liczby pikseli w kolorze skóry i kolorze czarnym w ramach 25 pikseli pobranych jako próbka obrazu. A następnie na podstawie wartości progowej podjęcie decyzji czy cała próbka zostanie oznaczona kolorem skóry czy kolorem czarnym. W ramach implementacji wartość progu została ustalona na poziomie 12 pikseli, oznacza to, że wszystkie próbki zawierające więcej niż 12 pikseli czarnych są w całości oznaczane jako obszar niezawierający skóry i na odwrót.

Metodą weryfikacji pozwalającą stwierdzić czy znaleziony obszar jest elementem twarzy polega na odnalezieniu w sąsiedztwie obszarów będących w kolorze włosów. Jest to prymitywna metoda nie gwarantująca 100% poprawności, ale na potrzeby tej pracy magisterskiej jest w zupełności wystarczająca.

Do wykrywania potencjalnych obszarów w którym znajdują się włosy wykorzystywana jest składowa intensywności modelu HSI. Zależność pomiędzy składowymi modelu RGB a wspomnianą intensywnością opisana jest za pomocą równania 4.10

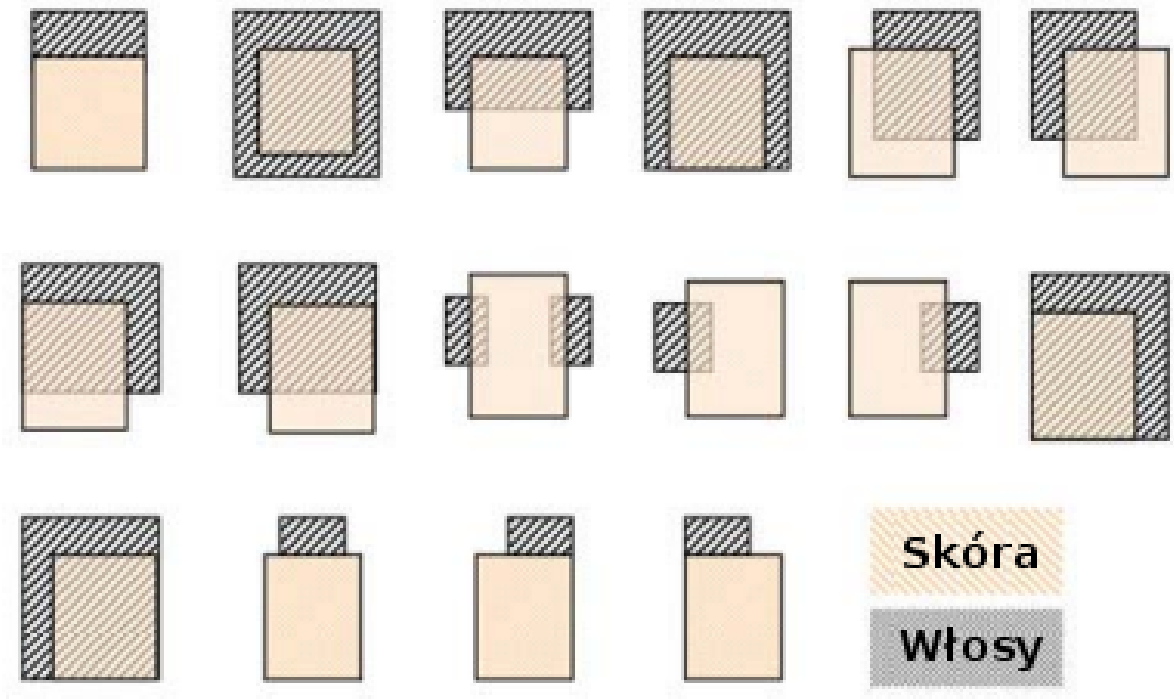
$$I = \frac{1}{3}(R + G + B) \tag{4.10}$$

[14] HSI (z ang. Hue Saturation Intensity) - model przestrzeni bar nawiązujący do sposobu w jaki widzi ludzki narząd wzroku

Tak zdefiniowany komponent intensywności może zostać wykorzystany do ewaluacji jasności obrazka, a co za tym idzie może posłużyć do odnalezienia ciemnych obszarów które mogą stanowić obszar włosów. W oparciu o dane eksperymentalne zostało zdefiniowane następujące równanie opisujące zakres kolorów w którym mogących stanowić włosy.

$$H = \begin{cases} 1 & \text{dla } (I < 80 \cap (B - G < 15 \cup B - R < 15)) \cup (20 < H \leq 40) \\ 0 & \text{dla pozostałych} \end{cases} \tag{4.11}$$

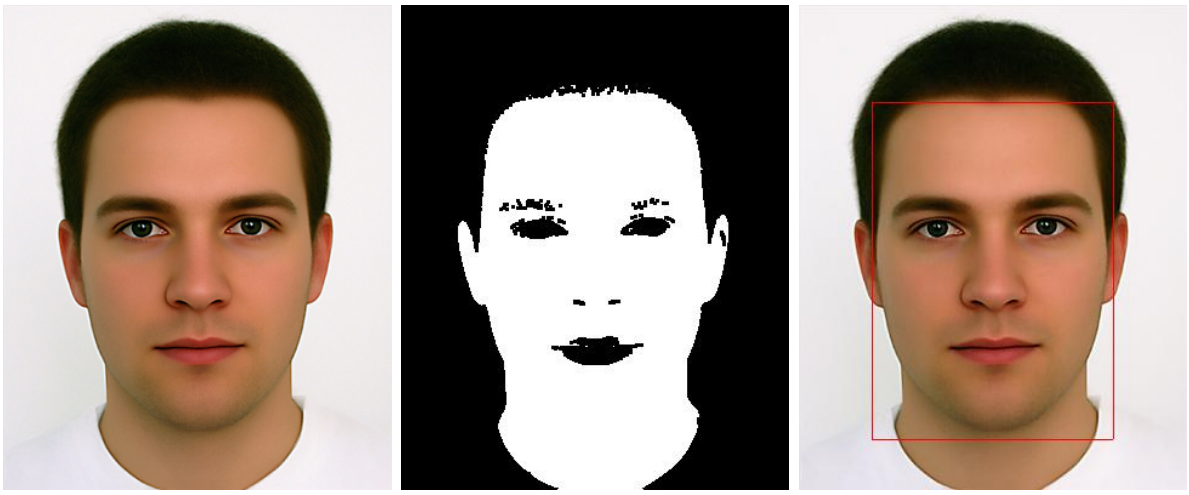
Po przeprowadzeniu binaryzacji za pomocą równania 4.11, podobnie jak to miało miejsce w przypadku detekcji skóry, przeprowadzana jest kwantyzacja otrzymanego obrazu. Następnym krokiem algorytmu jest wyszukanie wszystkich obszarów które zostały zaklasyfikowane jako kolor skóry i odnalezienie korespondującego z nimi obszaru włosów. Dla każdego obszaru wyznaczany jest minimalny prostokąt otaczający, a następnie sprawdzany jest w jakiej relacji jest on z pozostałymi elementami. W przypadku gdy obszar reprezentujący skórę oraz włosy znajdą się w jednej z dozwolonych konfiguracji (rys. 4.18) obszar jest klasyfikowany jako twarz.



Rysunek 4.18: Dopuszczalne relacje pomiędzy wykrytymi obszarami skóry i włosów (źródło: [21])

4.7.3. Wnioski i podsumowanie

Opisana implementacja poddawana była licznym testom mających na celu wyznaczenie skuteczności z jaką algorytm lokalizuje twarz na zdjęciach różnych osób w różnych warunkach oświetleniowych. Już pierwsze testy wykazały, że gdy akwizycja obrazu odbywa się w warunkach laboratoryjnych algorytm bez problemów znajduje twarze na zdefiniowanym obrazie (rys. 4.19).



(a) Obraz początkowy (b) Obraz po binaryzacji (c) Wykryta twarz

Rysunek 4.19: Weryfikacja działaniu algorytmu dla zdjęcia wykonanego w warunkach laboratoryjnych [15]

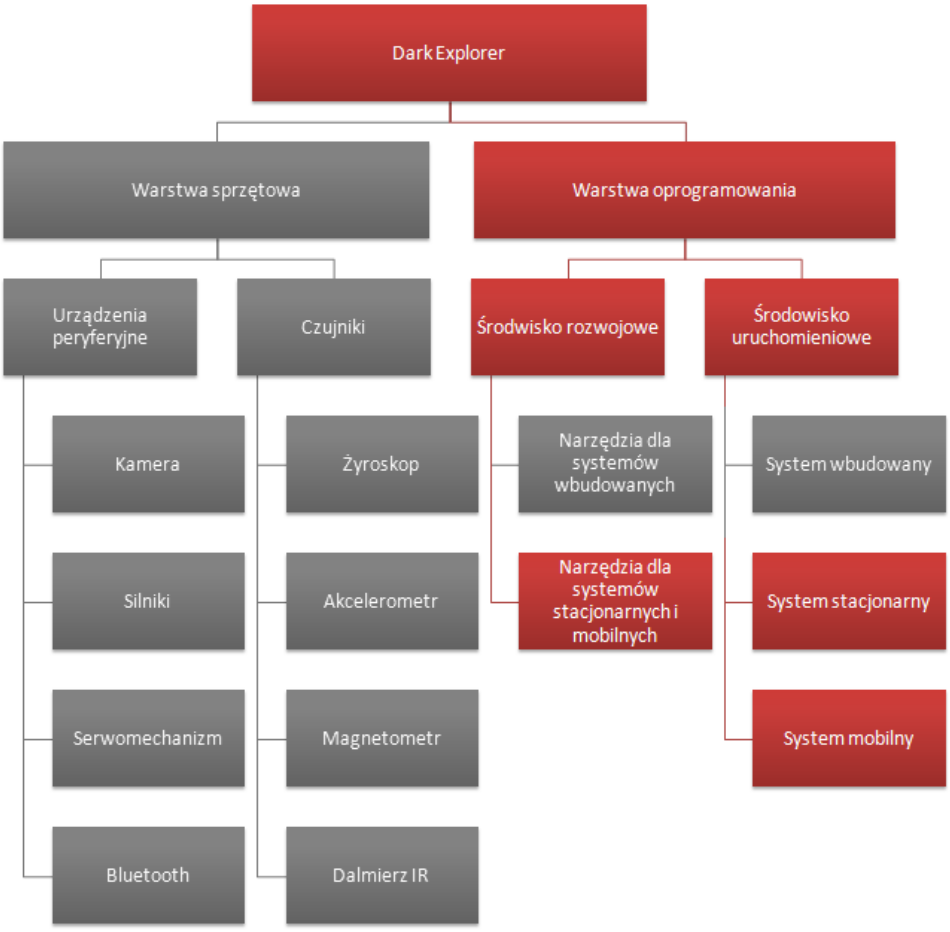
Niestety w przypadku gdy zdjęcie zostało wykonane w złych warunkach oświetleniowych lub w tle znajdują się elementy w drewniane lub czarne algorytm nie sprawdza się i z bardzo małą skutecznością lokalizuje twarz. Źródłem takiego stanu rzeczy jest oparcie działania całego algorytmu o analizę kolorów. W celu osiągnięcia większej skuteczność w działaniu algorytmu konieczne byłoby wykorzystanie metod bazujących nie tylko na kolorach ale również na geometrii twarzy. Aby to stało się możliwe konieczne jest zwiększenie dostępnej ilości pamięci ponieważ w obecnej konfiguracji jest ona nie wystarczająca. Należałoby rozważyć również wykorzystanie szybszego procesora gdyż niektóre metody bazujące np. na porównywaniu szablonów wymagają przeprowadzania transformacji geometrycznych na obrazie co przy wykorzystaniu obecnej jednostki obliczeniowej może być niezwykle czasochłonne. Dzięki udoskonalonemu procesowi akwizycji danych z kamery możliwe jest zrezygnowanie z analizowania obrazu na mikrokontrolerze i przeprowadzenie profesjonalnej lokalizacji twarzy za pomocą komputera. Jedynym ograniczeniem takiego podejścia jest przepustowość obecnego kanału komunikacyjnego.

[15] źródło obrazu początkowego: Beauty Check - `http://www.uni-regensburg.de/`

Rozdział 5

Aplikacje zarządzające robotem dla PC i urządzeń mobilnych

Niniejszy rozdział opisuje zakres działań podjętych w celu stworzenia aplikacji sterujących robotem Dark Explorer (rys. 5.1). Przedstawiony jest zestaw narzędzi niezbędnych do rozwoju oprogramowania dedykowanego dla robota na urządzenia mobilne oraz komputery stacjonarne. Opisano także możliwości stworzonych aplikacji, sposób instalacji i użytkowania.



Rysunek 5.1: Struktura platformy robota mobilnego po zakończeniu prac. Kolorem czerwonym oznaczono zakres prac opisanych w bieżącym rozdziale.

5.1. Biblioteki programistyczne do zarządzania robotem

Jednym z poważniejszych problemów zauważonych podczas analizy pierwotnej konfiguracji robota był brak bibliotek umożliwiających tworzenie oprogramowania, które pozwalałoby na dowolne wykorzystanie możliwości oferowanych przez konfigurację sprzętową robota. Brak tego rodzaju narzędzi znacząco ogranicza możliwości rozwoju robota, gdyż każda próba tworzenia nowego oprogramowania wymaga od programisty zagłębiania się w szczegóły implementacji systemu wbudowanego, który kontroluje działanie robota. Aby usunąć tak poważne ograniczenie, zaprojektowana została biblioteka mająca na celu udostępnienie narzędzi, pozwalających programiście na skupienie się jedynie na wysokopoziomowej funkcjonalności, bez konieczności szczegółowej analizy protokołu komunikacji i zasad działania poszczególnych funkcji robota.

Przed przystąpieniem do implementacji konieczne jest podjęcie rozważnej decyzji z wyborem środowiska i języka programowania w jakim biblioteka zostanie napisana. Biorąc pod uwagę ciągle rosnącą popularność obiektowych języków programowania oczywistym wydaje się być wybór języka właśnie z tej rodziny. Decydującym aspektem, wpływającym na ostateczny wybór docelowej platformy rozwojowej, jest więc ilość dostępnych bibliotek oraz przenośność kodu pomiędzy dostępnymi na rynku platformami sprzętowymi. Po przeprowadzeniu wnikliwej analizy ostateczny wybór padł na dwa rozwiązania: język C# z platformą .NET oraz język Java. Wybór motywowany jest faktem iż platforma .NET jest jednym z najdynamiczniej rozwijających środowisk programistycznych ostatnich lat natomiast Java jest niesamowicie popularna i dostępna w obecnych środowiskach uruchomieniowych.

5.1.1. Platforma .NET

Firma Microsoft dostarcza szereg bibliotek dodatkowych oraz narzędzi pozwalających na szybkie tworzenie oprogramowania działającego zarówno na urządzeniach mobilnych jaki i stacjonarnych. Dzięki pracy programistów w ramach projektu Mono[1] powstała platforma umożliwiająca uruchamianie aplikacji napisanych w języku C# nie tylko pod kontrolą systemu Windows, ale również pod systemami z rodziny Linux i Mac. Dodatkowym atutem platformy .NET jest bardzo duża liczba bibliotek dodatkowych dostarczanych przez środowiska programistów opensource.

[1] Więcej informacji na temat projektu Mono można znaleźć pod adresem strony internetowej http://www.mono-project.com/

W ramach pracy magisterskiej zaprojektowana została biblioteka programistyczna w języku C#. Docelową platformą uruchomieniową dla przygotowanej biblioteki są systemy z rodziny Windows, Windows Mobile oraz Linux. Przy tworzeniu biblioteki brane pod uwagę były najnowsze trendy w dziedzinie programistycznych wzorców projektowych przy jednoczesnym zachowaniu spójności i uniwersalności kodu dla poszczególnych środowisk uruchomieniowych. W wyniku implementacji powstała wielowątkowa biblioteka oparta na zdarzeniach pozwalająca na dostęp do wszystkich funkcji robota za pomocą intuicyjnego interfejsu programistycznego. Biblioteka udostępnia szerokie spektrum metod pozwalających na swobodne sterowanie i zarządzanie dostępnymi funkcjami robota. Wśród metod bibliotecznych znaleźć można funkcje pozwalające na bezpośrednią interakcję z poszczególnymi podzespołami bazowymi, jak również takie które umożliwiają wykonywanie predefiniowanych sekwencji zadań przewidzianych przez autorów projektu. Biblioteka pokrywa swoją funkcjonalnością nie tylko wsparcie dla wszystkich dostępnych rozszerzeń sprzętowych robota, ale również dostarcza interfejs pozwalający na automatyczną obsługę połączenia z robotem z wykorzystaniem technologii bluetooth. Fakt ten jest o tyle istotny, że proces komunikacji okazał się być najbardziej wrażliwym elementem podczas migracji biblioteki pomiędzy poszczególnymi platformami softwareowymi. Szczegółowa lista wszystkich dostępnych funkcji wraz z niezbędnym komentarzem, zamieszczona została w dodatku poświęconym kodowi źródłowemu, stworzonemu w ramach pracy magisterskiej. Taki sposób implementacji pozwala na tworzenie oprogramowania współpracującego z nową wersją robota nawet przez osoby nie posiadające dostatecznej wiedzy i umiejętności tworzenia oprogramowania do obsługi systemów wbudowanych.

W ramach pracy magisterskiej stworzona została również przykładowa aplikacja sterująca prezentująca wszystkie możliwości oferowane zarówno przez warstwę aplikacyjną jak i sprzętową robota Dark Explorer. Do wykonania aplikacji klienckiej wykorzystana została omawiana powyżej biblioteka programistyczna. Stanowi ona przewodnik dla początkującego programisty pokazujący w jaki sposób można wykorzystać udostępnione funkcje biblioteczne. Okno główne aplikacji widoczne jest na rysunku 5.2. Program umożliwia sterowanie zarówno manualne jak i sterowanie półautomatyczne po włączeniu trybu rekonstrukcji ścieżki czy rozpoznawania obrazu. Korzystając z menu głównego aplikacji użytkownik może dokonywać kalibracji czujników i przeprowadzać za ich pomocą dowolne pomiary. Dodatkowo w aplikacji udostępniona została historia wykonywanych akcji. Jest to bardzo przydatne narzędzie pozwalające na bieżąco śledzić nie tylko stan aplikacji ale również robota. Dla wygody użytkownika większość dostępnych funkcji została zgrupowana w menu głównym aplikacji. Dzięki takiemu podejściu użytkownik może bardzo szybko uzyskać dostęp do interesującej go funkcji bez konieczności przedzierania się przez dużą ilość kontrolek w oknie głównym aplikacji.

Rysunek 5.2: Okno główne aplikacji sterującej napisanej w języku C#

5.1.2. Biblioteka zarządzająca dla języka Java

Drugim rozwiązaniem programistycznym umożliwiającym sterowanie robotem za pomocą urządzeń zewnętrznych jest biblioteka napisana w języku Java. Pozwala ona na komunikowanie się z robotem przy pomocy prostych funkcji, które nie wymagają od programisty żadnej wiedzy na temat mechanizmów przesyłu danych pomiędzy robotem a resztą świata. Podobnie jak w przypadku biblioteki przygotowanej pod platformę .NET możemy bezpośrednio sterować poszczególnymi podzespołami bazowymi wysyłając własne komendy lub też korzystać z funkcji wbudowanych, odpowiedzialnych za wykonywanie konkretnych czynności, np.: manewrowanie robotem, pobieranie zdjęcia w określonej rozdzielczości, kontrolowanie serwomechanizmu.

Biblioteka została podzielona na dwie klasy. Jedna z nich jest odpowiedzialna za obsługę komunikacji bluetooth pomiędzy aplikacją a robotem Dark Explorer. Druga natomiast pozwala w łatwy sposób wykorzystywać wszystkie funkcje robota.

Biblioteka była tworzona docelowo na telefony komórkowe z wbudowaną maszyną wirtualną Java, które implementują API do komunikacji bluetooth JSR 82[2]. Możliwe jest również wykorzystanie tej biblioteki na komputerach stacjonarnych. W tym celu konieczne jest wyposażenie tworzonej aplikacji w bibliotekę wspierającą JSR 82. Dostępnych jest kilka tego rodzaju bibliotek (tab. 5.1).

Tabela 5.1: Biblioteki wspierające JSR-82 [22]

Nazwa	Wsparcie bluetooth[3]	Wsparcie OBEX[4]	Platformy Java	Obsługiwane systemy	Cena
Avetana	Tak	Tak	J2SE	Win-32, Mac OS X Linux, Pocket PC	25€
BlueCove	Tak	Tak	J2SE	Win-32, Mac OS X	darmowa
Electric Blue	Tak	Tak	J2SE	Win XP SP2	15$
Harald	Nie	Nie	każda z javax.comm	wiele	darmowa
JavaBluetooth.org	Tak	Nie	każda z javax.comm	wiele	darmowa
Rococo	Tak	Tak	J2SE J2ME	Linux, Palm OS	2500€

[2] JSR 82 - Java Specification Request, zbiór reguł których należy się trzymać podczas implementacji API do komunikacji bluetooth

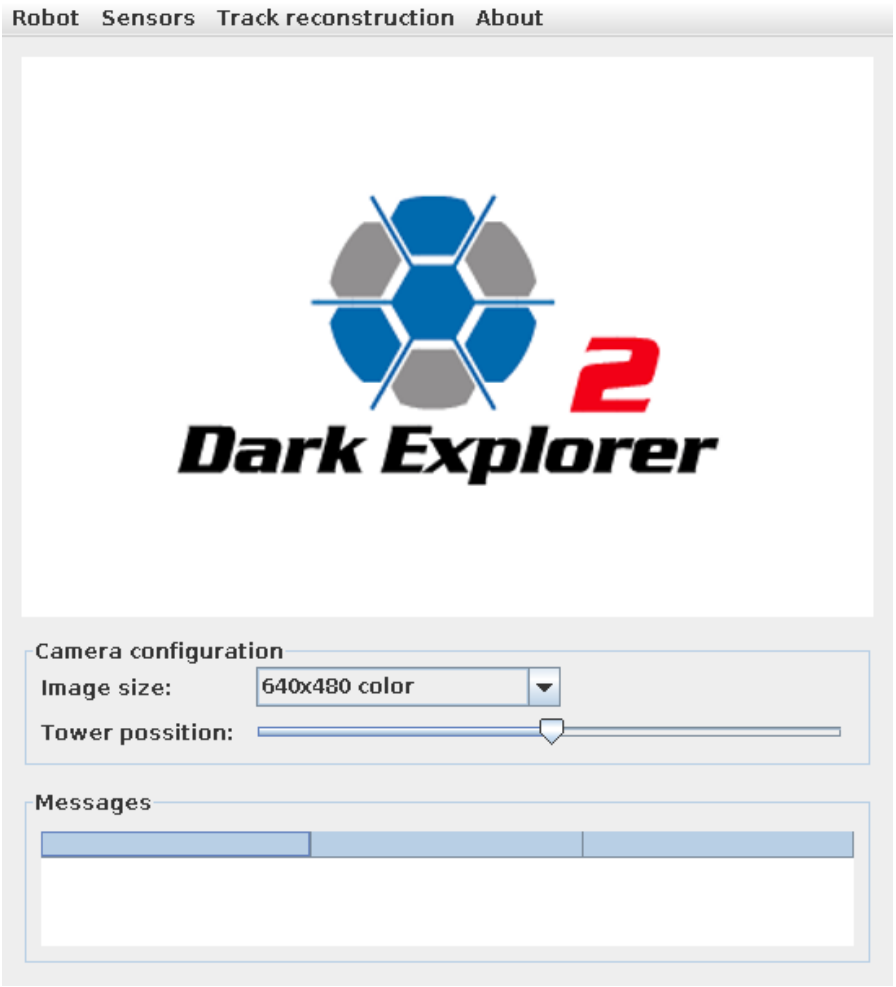
[3] Mowa tutaj o klasach i interfejsach dostępnych w ramach pakietu javax.bluetooth zgodnych z wymaganiami JSR 82

[4] Mowa tutaj o klasach i interfejsach dostępnych w ramach pakietu javax.obex zgodnych z wymaganiami JSR 82

5.2. Aplikacja dla komputerów stacjonarnych

Na potrzeby prezentacji biblioteki zarządzającej robotem Dark Explorer napisanej w języku Java, stworzono aplikację dla komputerów stacjonarnych. Można ją uruchomić na wielu systemach operacyjnych, między innymi na systemach Windows oraz Linux. Aplikacja demonstracyjna korzysta z przygotowanej na rzecz tej pracy magisterskiej biblioteki zarządzającej robotem oraz z biblioteki wspierającej JSR 82. Z pośród wielu bibliotek przedstawionych w tabeli 5.1 wybrana została bibloteka BlueCove [23]. Działa ona pod wieloma systemami operacyjnymi, jest darmowa dla zastosowaniach niekomercyjnych oraz posiada dokumentację wraz z przykładowymi aplikacjami.

Przygotowany program zarządzający robotem pozwala na zaprezentowanie wszystkich jego funkcjonalności stworzonych podczas rozwoju tej pracy magisterskiej. Rysunek 5.3 przedstawia ekran główny wspomnianej aplikacji. Funkcjonalności tej aplikacji są identyczne z tymi opisanymi w rozdziale 5.1.1.



Rysunek 5.3: Okno główne aplikacji sterującej napisanej w języku Java

5.3. Platforma mobilna (Windows Mobile 6.1)

Mobilne urządzenia przenośne z dnia na dzień zyskują na popularności. Każdego dnia spotykamy się z nimi w domu, w pracy czy spacerując po parku. Z całą pewnością można stwierdzić, iż większa część społeczeństwa obecnie jest w posiadaniu telefonu komórkowego, komputera przenośnego czy też jakiegoś innego urządzenia mobilnego. Wszystkie z wspomnianych urządzeń posiadają charakterystyczną dla siebie platformę software'ową. Do najlepiej znanych we współczesnym świecie zaliczyć można między innymi: Windows Mobile, iPhone, BlackBerry, Symbian OS, Android, Maemo, OpenMoko itp. Każda z wymienionych platform posiada inną genezę jak również ma swoje mocne i słabe strony.

Platformy takie jak Windows Mobile, BlackBerry czy iPhone ograniczone są do urządzeń dedykowanych docelowo do współpracy z wspomnianymi środowiskami. Obok różnorakich problemów z jakimi zmagają się wspomniane wcześniej platformy do jednego z najpoważniejszych zaliczyć można bardzo ograniczone w niektórych aspektach API[5]. Nawet tak przenośna platforma jak Java na urządzeniach przenośnych nie zawsze się sprawdza ze względu na liczne braki oraz różnice w API zmuszające programistów do tworzenia kodu dedykowanego dla konkretnego urządzenia. Symbian oraz Windows Mobile wypadają na tym tle nieco lepiej ponieważ wspierają szerszą gamę urządzeń jak również ich API daje więcej możliwości niż ma to miejsce na przykład w przypadku Javy. Głównym powodem takiego stanu rzeczy jest bardzo szeroki i różnorodny asortyment platform sprzętowych utrudniający stworzenie jednolitej i w pełni wykorzystującej wszystkie możliwości urządzenia platformy programistycznej. Dostępne w chwili obecnej OpenSource'owe i wieloplatformowe rozwiązania znajdują się ciągle we wczesnej fazie rozwoju i nie są jeszcze powszechnie znane przez środowiska twórców oprogramowania.

Firma Microsoft wypuściła po raz pierwszy na światło dzienne swoją platformę mobilną w latach 90-tych [24]. Natomiast w roku 2002 pojawiła się pierwsza platforma Windows CE.NET. Zapoczątkowało to popularyzację urządzeń Pocket PC opartych o system Windows CE 3.0 oraz późniejsze wersje. Dalszy rozwój bezprzewodowych technologii telekomunikacyjnych pozwolił na integrację telefonu z komputerem osobistym. Wspomniane urządzenia Pocket PC z 2002 roku wspierały między innymi standard GSM[6], GPRS[7], bluetooth oraz umożliwiały użytkownikom dostęp do sieci bezprzewodowych.

[5] Application Programming Interface - interfejs programowania aplikacji, specyfikacja instrukcji pozwalających na dostęp do zasobów dostarczanych przez zewnętrzny program

[6] Global System for Mobile Communication, pierwotnie Groupe Special Mobile - najpopularniejszy obecnie standard telefonii komórkowej

[7] General Packet Radio Service - technologia pakietowego przesyłania danych popularnie stosowana w sieciach GSM

W między czasie rozwojowi ulegały urządzenia typu SmartPhone które koncepcyjnie były bardzo zbliżone do Pocket PC jednakże były one bardziej zbliżone do telefonu niż komputera osobistego. Podstawową różnicą pomiędzy Smartphone i Pocket PC jest fakt iż urządzenia Pocket PC posiadają ekran dotykowy, a Smartphone wyposażone są jedynie w przyciski umożliwiające sterowanie urządzeniem. Każde z tych urządzeń posiadało inny zestaw aplikacji pomocniczych oraz wspierało inne standardy i technologie.

W chwili obecnej większość urządzeń Pocket PC oraz Smartphone działają w oparciu o system Windows Mobile 5 oraz Windows Mobile 6. Nowoczesne urządzenia Pocket PC wyposażone są w procesor o taktowaniu 500-600 MHz oraz 64-128 MB pamięci RAM. Najnowsze urządzenia z tej grupy wyposażane są w 1 GHz procesor oraz 512 MB pamięci.

5.3.1. Środowisko rozwojowe

Tworzenie aplikacji działających na urządzeniach pod kontrolą systemu Windows Mobile jest niemal tak samo proste jak tworzenie zwykłych aplikacji na komputery stacjonarne. Niemniej jednak do stworzenia w pełni funkcjonalnego środowiska rozwojowego (rys. 5.4) konieczne jest przejście przez klika kroków przygotowawczych związanych z instalacją potrzebnych aplikacji narzędziowych.

Rysunek 5.4: Elementy składowe środowiska rozwojowego Windows Mobile

Przed rozpoczęciem przygody z tworzeniem aplikacji dla systemu Windows Mobile konieczne jest zainstalowanie Microsoft Visual Studio. Zaleca się aby Visual Studio było w wersji 2005 lub 2008. Niestety narzędzia umożliwiające rozwijanie aplikacji mobilnych nie są poprawnie wykrywane przez Visual Studio 2010 oraz poprzednie wydania w wersji Express. Dlatego też koniecznością jest instalacja środowiska w wersji Standard lub Professional. Każda z tych wersji może zostać pobrana w wersji czasowej ze stron firmy Microsoft lub w wersji pełnej z MSDNAA[8]. Visual Studio posłuży nam nie tylko do edycji kodu aplikacji ale pozwoli również w prosty sposób budować, debugować oraz przygotować instalator finalnej wersji aplikacji. Po poprawnym zainstalowaniu środowiska rozwojowego konieczne jest pobranie i zainstalowanie dostępnych paczek serwisowych dostępnych dla wybranej wersji Visual Studio. Pozwoli to uniknąć nieprzyjemnych niespodzianek podczas instalacji bibliotek narzędziowych i późniejszej pracy.

Jeżeli posiadamy już zainstalowaną kopię Visual Studio możemy przystąpić do instalacji narzędzi pomocniczych które pomogą nam w tworzeniu aplikacji. Pierwszą niezbędną biblioteką jest .NET Compact Framework 2.0 SP1. Jest to zestaw narzędzi wykorzystywanych do uruchamiania aplikacji na platformach opartych o Windows Mobile. Aby ułatwić sobie proces budowania, debugowania i uruchamiania aplikacji na urządzeniu konieczne jest zainstalowanie w systemie Windows ActiveSync. Dzięki ActiveSync możliwe stanie się uruchamianie projektowanej aplikacji, bezpośrednio z IDE, nie tylko na prawdziwym urządzeniu ale również emulatorze.

Ostatnim, ale i zarazem najważniejszym krokiem jest instalacja Windows Mobile SDK[9]. Na stronach firmy Microsoft dostępne są dwie wersje SDK, Standard oraz Professional. Wersja Standard zawiera w sobie tylko wsparcie dla urządzeń z Windows Mobile Classic lub Standard natomiast wersja Professional obejmuje wszystkie dostępne środowiska. Wybór SDK można sprowadzić do następującej zasady. Jeżeli zamierzamy tworzyć oprogramowanie dla urządzeń Smartphone bez ekranu dotykowego w zupełności wystarczy nam wersja standardowa. Jeżeli natomiast planujemy napisane aplikacje uruchamiać na PocketPC lub dotykowych SmartPhone'ach będziemy potrzebować bibliotek systemu Windows Mobile Classic lub Professional, tak więc konieczne jest użycie SDK w wersji Professional. Tak jak w przypadku Visual Studio, również tutaj zaleca się instalację

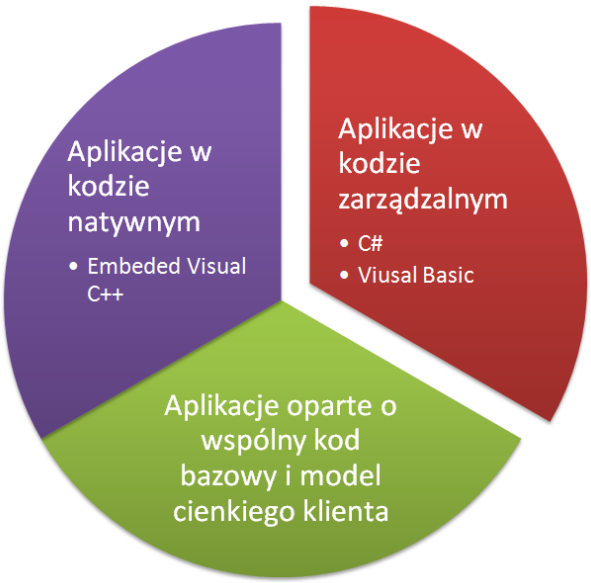
[8] Microsoft Developer Network Academic Alliance - program firmy Microsoft skierowany do studentów i pracowników naukowych w ramach którego uczestnicy mogą pozyskać darmowe kopie oprogramowania firmy Microsoft

[9] Software Development Kit - zestaw narzędzi programistycznych niezbędnych do tworzenia aplikacji korzystających z funkcjonalności dostarczonej przez daną bibliotekę

wszystkich dostępnych na stronie producenta aktualizacji i poprawek. Jest to szczególnie istotne podczas pracy z emulatorami urządzeń. Po zrealizowaniu tych kroków otrzymujemy w pełni funkcjonalne środowisko do rozwoju aplikacji mobilnych dla urządzeń smartphone.

Modele aplikacji

Istnieje klika modeli rozwoju aplikacji dla Windows Mobile (rys. 5.5), a wybór docelowego modelu został pozostawiony programiście. Pierwszy z nich służy do tworzenia aplikacji w kodzie natywnym. Aplikacje pisane zgodnie z tym modelem cechują się wysoką wydajnością, bezpośrednim dostępem do sprzętu oraz małym zużyciem zasobów. Do rozwoju tego rodzaju aplikacji korzysta się z reguły ze środowiska do rozwijania aplikacji z użyciem Embeded Visual C++. Główną wadą tego modelu jest niska przenośność pomiędzy różnymi platformami zwłaszcza jeżeli aplikacja korzysta z urządzeń specyficznych dla danego modelu urządzenia. Docelowo więc za pomocą tego modelu tworzy się biblioteki i narzędzia ułatwiające tworzenie bardziej skomplikowanych aplikacji. Jeżeli więc interesuje nas tworzenie wysokopoziomowych aplikacji z GUI, skierowanych bezpośrednio do użytkowników, zaleca się tworzenie tego typu aplikacji za pomocą kodu zarządzalnego z użyciem takich języków jak C# czy Visual Basic.



Rysunek 5.5: Modele tworzenia aplikacji dla Windows Mobile.

Rozwijanie aplikacji w oparciu o kod zarządzalny pozwala na stworzenie programu który będzie mógł w pełni wykorzystywać możliwości oferowane przez Microsoft .NET Compact Framework. Umożliwia to programiście tworzenie rozproszonych systemów mobilnych pracujących zarówno w modelu ze stałym połączeniem jak i bez. Spora część

narzędzi dostępnych w ramach .NET Compact Framework jest również wykorzystywana do rozwoju aplikacji na komputery stacjonarne. Biblioteka została zaprojektowana docelowo na urządzenia o ograniczonych zasobach co w połączeniu z możliwościami języków z rodziny .NET oraz integracją z Visual Studio daje nam profesjonalny zestaw narzędzi do tworzenia aplikacji mobilnych.

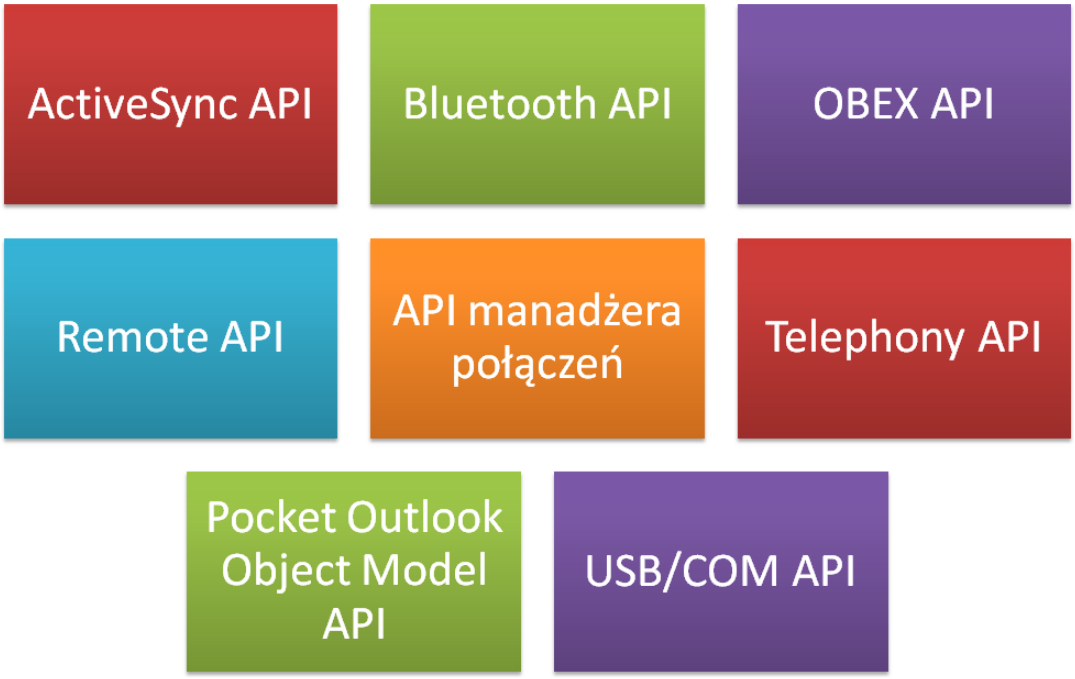
Trzecim modelem tworzenia programów pod Windows Mobile jest wykorzystywanie kodu serwera do pracy z wieloma różnymi typami urządzeń poprzez jeden wspólny kod bazowy i model cienkiego klienta. Oczywiście tego typu podejście ma sens jedynie gdy możemy zagwarantować stabilny kanał komunikacyjny pomiędzy urządzeniem klienta, a serwerem. Każdy z przedstawionych modeli idealnie sprawdza się jeżeli tylko w prawidłowy sposób wybierzemy model najbardziej odpowiadający potrzebom naszej aplikacji.

Graficzny interfejs użytkownika

Dzięki wygodnemu systemowi projektowania GUI dostępnego w Visual Studio, tworzenie interfejsu użytkownika dla platformy mobilnej jest niemal tak proste jak w przypadku tradycyjnych aplikacji, a jedyną różnicą jest ilość i rodzaj dostępnych kontrolek. Różnice te wynikają z faktu iż niektóre urządzenia mobilne posiadają ekrany dotykowe, a inne nie. Co za tym idzie, rozwój interfejsu użytkownika staje się bardziej skomplikowany zwłaszcza jeżeli interesuje nas rozwój aplikacji wspólnej dla obydwu platform sprzętowych. W tym miejscu nie może zabraknąć informacji, że oprogramowanie zbudowane dla PocketPC nie uruchomi się na urządzeniach SmartPhone natomiast sytuacja odwrotna jest możliwa do momentu w którym aplikacja nie zacznie korzystać ze specyficznych funkcji SmartPhone. Naturalnym stanem rzeczy wydaje się fakt, iż wiele funkcji i komponentów graficznych znanych z aplikacji desktopowych zostało usunięte z bibliotek Windows Mobile, aby zapewnić jej wydajność i niewielki rozmiar. Dlatego też pozostawiono tylko niezbędne, najprostsze komponenty. Ponieważ wydajność i zasoby pamięciowe urządzeń stale rosną, również ilość narzędzi dostępnych w SDK jest systematycznie zwiększana, a co za tym idzie różnice pomiędzy kolejnymi wersjami .NET Compact Framework są bardzo duże. Dlatego też posiadanie jak najbardziej aktualnej wersji SDK ma niebagatelne znaczenie dla wygody tworzenia aplikacji. Podsumowując, rozwój GUI dla platformy mobilnej nie różni się bardzo od tworzenia interfejsu użytkownika dla aplikacji dekstopowej. Istnieje również możliwość rozwijania GUI w oparciu o silniki 3D.W chwili obecnych dostępne są takie rozwiązania jak GAPI (Game API), OpenGL ES (Embeded Systems), Open VG (Vector Grapics). Jednakże rozwój takich aplikacji jest niezwykle trudny gdyż wymaga od programisty tworzenia maksymalnie optymalnego kodu, ze względu na ograniczone możliwości niektórych urządzeń.

Komunikacja

Nowoczesne urządzenia mobilne posiadają szeroką gamę możliwości komunikacyjnych (rys. 5.6). Posiadają one dostęp do szybkich sieci bezprzewodowych w standardzie 802.11 WiFi. Umożliwiają one również komunikację za pomocą portu podczerwieni, bluetooth czy USB. Podejmując decyzję na temat wyboru kanału komunikacji należy brać pod uwagę nie tylko parametry techniczne, ale również liczbę standardów i protokołów dostępnych dla danego kanału komunikacji. Firma Microsoft dostarcza szereg interfejsów programistycznych (API) umożliwiających niemal błyskawiczny dostęp do funkcji komunikacyjnych urządzenia. AcitveSync API dostarcza funkcjonalność umożliwiającą komunikację za pomocą protokołu synchronizacji. Natomiast Bluetooth API dostarcza zestaw narzędzi umożliwiających nawiązywanie komunikacji bezprzewodowej pomiędzy telefonami jak i urządzeniami peryferyjnymi.



Rysunek 5.6: Lista dostępnych na platformie Windows Mobile API komunikacyjnych

API Managera połączeń dostarcza zestaw usług umożliwiających automatyzację procesu nawiązywania połączenia oraz zarządzanie ich aktywnością. API wymiany obiektów (OBEX API) dostarcza funkcjonalność umożliwiającą wymianę danych pomiędzy urządzeniami za pomocą efektywnego, kompaktowego protokołu binarnego dedykowanego dla urządzeń z ograniczonymi zasobami. Remote API (RAPI) dostarcza funkcję do zarządzania oraz zdalnego wywoływania metod po stronie urządzenia klienta. Dostępne są m.in. funkcje dostępu do rejestru, plików, bazy danych czy też konfiguracji urządzenia. Najważniejszą funkcjonalnością jest jednak możliwość zdalnego wywoływania procedur.

Za pomocą funkcji CeRapiInvoke() przesyłamy do urządzenia nazwę biblioteki dynamicznej wraz z nazwą metody która ma zostać wywołana na urządzeniu mobilnym. Kolejnym zestawem narzędzi jest Pocket Outlook Object Model API. Dostarcza ono funkcje do zarządzania obiektami Pocket Outlook, co umożliwia synchronizację zadań, kalendarza czy kontaktów za pomocą prostego i intuicyjnego interfejsu. Dostępne jest również Telephony API (TAPI) które zawiera w sobie biblioteki umożliwiające zarządzanie kartą SIM oraz wiadomościami SMS. TAPI udostępnia również zestaw funkcji umożliwiających dostęp do funkcji telefonowania oraz protokołu WAP. Nie zabrakło również narzędzi do pracy z portami USB oraz COM. Część z dostępnych portów COM jest zarezerwowana dla urządzeń wewnętrznych, ale pozostałe dostępne są do pełnej dyspozycji użytkownika.

Debugowanie

Microsoft Visual Studio umożliwia debugowanie aplikacji działających pod kontrolą Windows Mobile niemal w taki sam sposób jak ma to miejsce w przypadku tradycyjnych aplikacji desktopowych. Ponadto programista ma do swojej dyspozycji następujące narzędzia: emulator, panel zarządzania emulowanymi urządzeniami, panel punktów przerwań i wątków. Niestety w Visual Studio nie uda się nam jednocześnie debugować kodu natywnego i zarządzalnego. Możliwe jest natomiast uruchomienie zarówno projektu napisanego w Visual C++ jak i projektu opartego o kod zarządzalny, a dzięki funkcjonalności „Dołącz do procesu" możliwe jest zdalne dołączenie się i monitorowanie procesu działającego na urządzeniu lub emulatorze urządzenia. Narzędziem umożliwiającym komunikację pomiędzy urządzeniem a systemem jest ActiveSync instalowany wraz ze środowiskiem rozwojowym. Za pomocą narzędzia ActiveSync możemy łączyć się nie tylko z rzeczywistymi urządzeniami ale również z emulatorami. Umożliwia to pełną wirtualizację urządzeń mobilnych i znacznie ułatwia testowanie funkcjonalności zwłaszcza pomiędzy różnymi platformami urządzeń (SmartPhone, PocketPC). Jedynym ograniczeniem tego procesu jest możliwość utrzymania tylko jednego aktywnego połączenia co uniemożliwia debugowanie na wielu urządzeniach jednocześnie. Co więcej, Visual Studio umożliwia debugowanie jedynie aplikacji stworzonej przez programistę, nie możliwe jest z poziomu IDE debugowanie aplikacji i usług systemowych działających na urządzeniu. Do tego typu debugowania konieczne byłoby zbudowanie własnej wersji systemu Windows Mobile przy użyciu Platform Buildera. Narzędzie to umożliwia również tworzenie własnego SDK dla Visual Studio i platformy Windows CE. Dodatkową możliwością dostępną z poziomu emulatora jest emulowanie połączenia z siecią GSM oraz wsparcie dla GPS. Umożliwia to testowanie, debugowanie i rozwijanie szerokiego spektrum aplikacji bez konieczności posiadania urządzenia fizycznie.

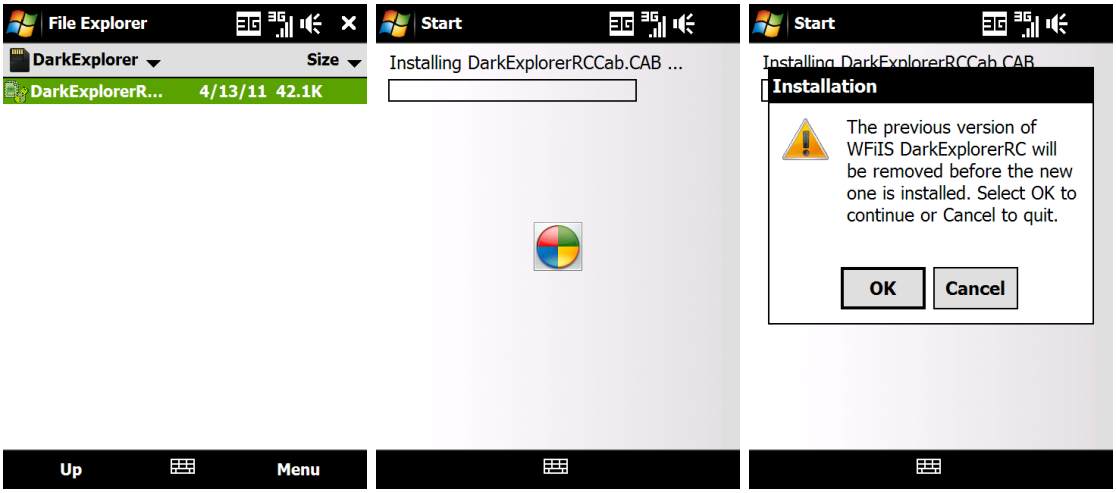
5.3.2. Środowisko uruchomieniowe aplikacji mobilnej

Docelowym środowiskiem na którym uruchamiana będzie aplikacja sterująca robotem Dark Explorer będą urządzenia działające pod kontrolą systemu Windows Mobile. Jedynym wymaganiem stawianym przed urządzeniem na którym podjęta zostanie próba uruchomienia aplikacji jest posiadanie uprzednio zainstalowanej biblioteki .NET Compact Framework w wersji co najmniej 2.0 SP1. Wersję instalacyjną biblioteki dla której były przeprowadzane testy można znaleźć na dołączonej do pracy płycie.

Instalacja aplikacji sterującej

Przed przystąpieniem do korzystania z mobilnej aplikacji sterującej konieczne jest jej zainstalowanie na urządzeniu za pomocą którego użytkownik chciałby komunikować się z robotem mobilnym. Aby tego dokonać konieczne jest skopiowanie pliku instalatora do pamięci telefonu. Po zakończeniu procesu kopiowania za pomocą menadżera plików zainstalowanego w telefonie uruchamiamy instalator aplikacji zgodnie z rysunkiem 5.7a. W przypadku gdy dokonujemy aktualizacji wersji aplikacji instalator poinformuje użytkownika o wykrytej wersji aplikacji i będzie wymagał potwierdzenia przed usunięciem poprzedniej wersji i kontynuacją instalacji obecnej wersji. Komunikat którego może się spodziewać użytkownik widoczny jest na zdjęciu 5.7c. Kolejnym etapem instalacji będzie wybór katalogu docelowego w którym zainstalowana zostanie aplikacja (rys. 5.7d). Po rozpoczęciu pracy instalator będzie informował użytkownika o stanie procesu za pomocą widocznego na rysunku 5.7e paska postępu. Instalacja może zająć od kilku do kilku dziesięciu sekund w zależności od aktulanego obciążenia i konfiguracji sprzętowej telefonu. Poprawne zakończenie instalacji zostanie potwierdzone poprzez wyświetlenie komunikatu widocznego na rysunku 5.7f. Po poprawnym zainstalowaniu aplikacji konieczne jest jeszcze skonfigurowanie połączenia bluetooth, aby program sterujący potrafił z niego poprawnie skorzystać.

W tym celu należy uruchomić moduł bluetooth zarówno w telefonie jak i robocie. Następnie należy sparować urządzenia, aby umożliwić im bezpośrednią wymianę danych bez konieczności potwierdzania przesłania każdego komunikatu. Domyślnym kodem potrzebnym do nawiązania połącznia bluetooth jest kod „0000”. Kolejnym etapem konfiguracji jest uruchomienie usługi komunikacji za pomocą protokołu RFCOMM oraz stworzenie nowego portu wychodzącego poprzez który przesyłane będą dane pomiędzy robotem, a urządzeniem sterującym. Wszystkie wspomniane kroki konfiguracyjne zaprezentowane są kolejno na rysunkach 5.7g, 5.7h oraz 5.7i. Po zakończeniu tego etapu możliwe jest już rozpoczęcie korzystania z wszystkich możliwości oferowanych przez aplikację.



(a) Uruchomienie instalatora (b) Przygotowanie instalacji (c) Potwierdzanie reinstalacji

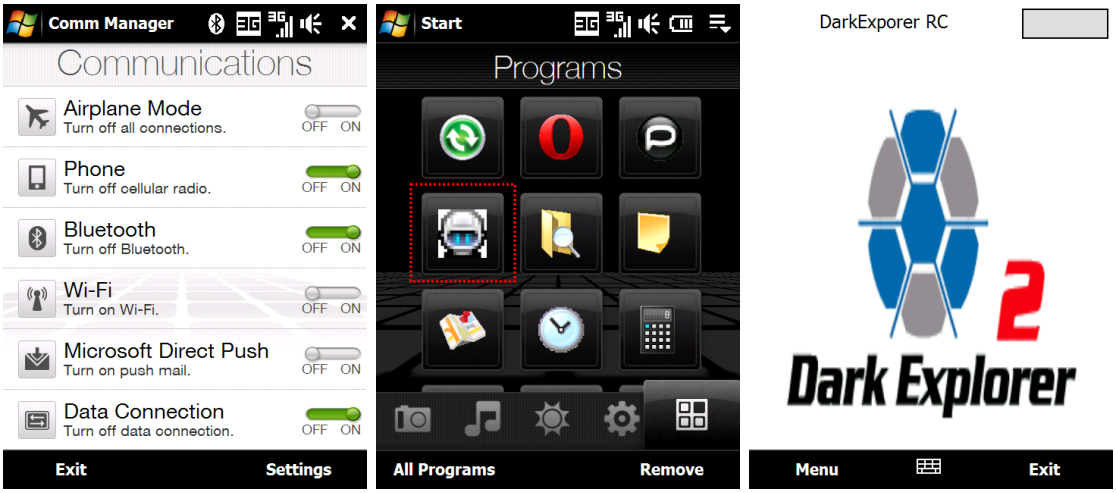
(d) Wybór miejsca instalacji (e) Instalator w trakcie działania (f) Poprawne ukończenie instalacji

(g) Parowanie urządzenia (h) Uruchamianie usługi (i) Konfiguracja portu COM

Rysunek 5.7: Kroki wymagane do przeprowadzenia poprawnej instalacji i konfiguracji mobilnej aplikacji sterującej

Uruchomienie aplikacji sterującej

Jeżeli posiadamy już zainstalowaną i skonfigurowaną aplikację sterującą to możemy rozpocząć korzystanie z jej możliwości. W tym celu należy włączyć robota i uruchomić moduł bluetooth dostępny w telefonie. Następnie na liście dostępnych aplikacji należy odszukać program pod nazwą „Dark Explorer RC" ikona programu widoczna jest na rysunku 5.8b. Wybierając program z listy zainicjujemy proces jego uruchamiania. Po załadowaniu wszystkich komponentów aplikacji na ekranie wyświetli się okno główne aplikacji widoczne na rysunku 5.8c. Aby uprościć dostęp do funkcji programu zostały one wszystkie zebrane w menu dostępnym w lewym dolnym rogu aplikacji. Wszystkie dostępne opcje pogrupowano pod względem tematycznym, tak aby nawet użytkownik uruchamiający aplikację po raz pierwszy nie miał trudności z odszukaniem interesujących go funkcji.



(a) Uruchomienie bluetooth (b) Ikona startowa (c) Okno główne programu

Rysunek 5.8: Uruchomienie mobilnej aplikacji sterującej

Aby uczynić aplikację jeszcze bardziej przyjazną użytkownikowi część funkcji dostępna jest również pod sprzętowymi przyciskami telefonu. Dobrym przykładem tego rodzaju funkcji jest sterowanie kierunkiem jazdy robota. Dostęp do tej funkcjonalności można uzyskać bezpośrednio za pomocą przycisków strzałek góra, dół, prawo, lewo. Ponad to naciskając przycisk środkowy użytkownik ma możliwość pobrania danych z kamery zgodnie z aktualną konfiguracją aplikacji. Dodatkowym atutem programu jest możliwość sterownia kierunkiem jazdy robota za pomocą akcelerometru jeżeli urządzenie mobilne jest w takowy wyposażone. Aktualny stan robota oraz aplikacji prezentowany jest w pasku statusu widocznym u góry okna głównego aplikacji. Pozwala to użytkownikowi na bieżąco monitorować nie tylko stan aplikacji ale również to co w chwili obecnej dzieje się na robocie.

5.4. Platforma mobilna (Java ME)

Jednym z elementów rozwoju możliwości robota mobilnego Dark Explorer było stworzenie aplikacji zarządzającej na urządzenia przenośne. W ramach tego zadania stworzono aplikację działającą na platformie Java ME[10]. Platforma ta jest dedykowana dla aplikacji tworzonych na urządzenia mobilne o bardzo ograniczonych zasobach, takich jak telefony komórkowe.

W związku z niską wydajnością procesorów oraz małą ilością pamięci w telefonach komórkowych Java ME posiada ograniczony w stosunku do Java SE[11] zbiór klas nazywanych konfiguracją. W Java ME wyróżniamy twa typy konfiguracji: CDC[12] dla urządzeń o lepszych parametrach (smartfony) oraz CLDC[13] dla urządzeń o słabych parametrach (proste telefony komórkowe). W tej pracy wykorzystana została konfiguracja CLDC.

Konfiguracje Java ME są uzupełniane przez profile MIDP[14] które dodają swoje własne klasy do klas istniejących w konfiguracji. Klasy te zapewniają wykonywanie odpowiednich zadań na konkretnych elementach urządzenia mobilnego. Aplikacje wykorzystujące MIDP nazywane są MIDletami i są uruchamiane w środowisku KVM[15].

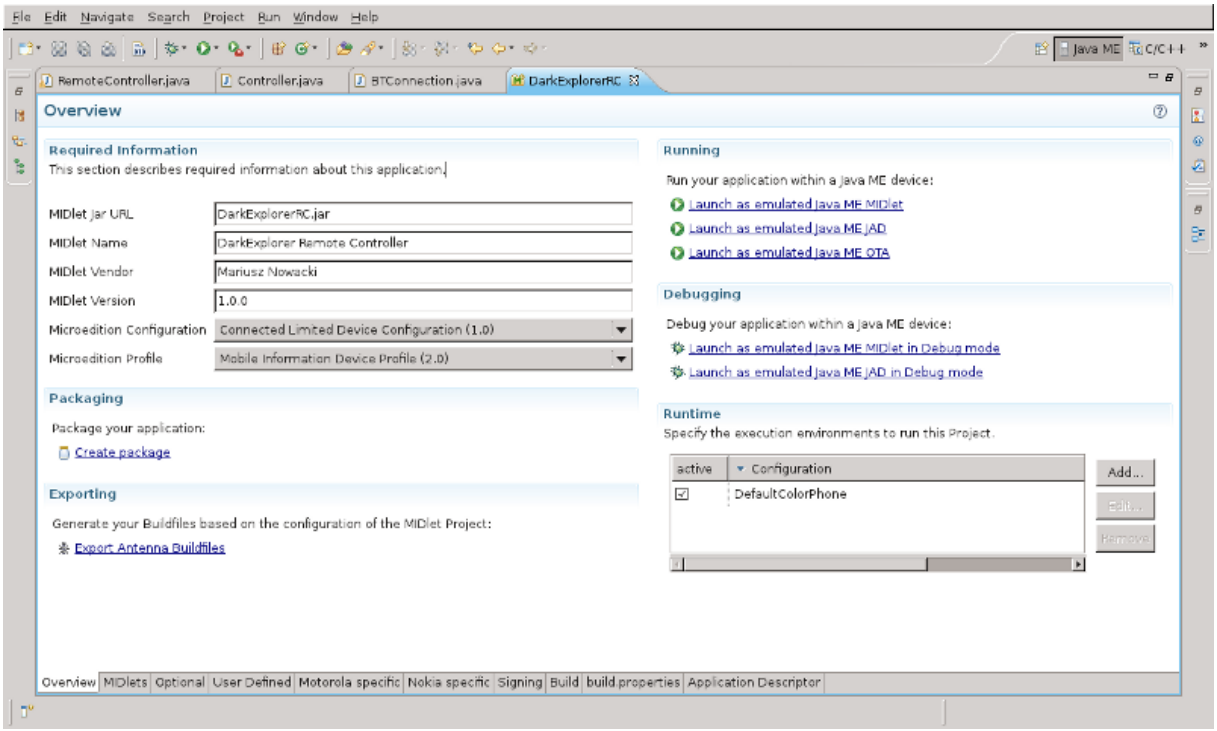
K Virtual Machine jest niczym innym jak wirtualną maszyną Java opracowaną dla konfiguracji CLDC. Jest ona bardzo ograniczona przez co posiada mniejsze wymagania sprzętowe w porównaniu ze swoimi odpowiednikami z komputerów klasy PC. Każdy producent urządzeń mobilnych musi zadbać o własną implementację maszyny wirtualnej na której będą uruchamiane MIDlety.

5.4.1. Narzędzia programistyczne

Do rozwoju oprogramowania na platformę mobilną Java ME niezbędne jest przygotowanie odpowiedniego środowiska programistycznego. Na potrzeby tej pracy zostało wykorzystane zintegrowane środowisko programistyczne Eclipse, którego krótki opis instalacji i konfiguracji znajduje się w rozdziale 4.2.2. Dużym ułatwieniem pracy jest zainstalowanie Mobile Tools for Java, dodatku do aplikacji Eclipse. Pozwala on na łatwe zarządzanie projektem oprogramowania na urządzenie mobilne.

[10] Java ME - Java Micro Edition
[11] Java SE - Java Standard Edition, wersja platformy Java na komputery stacjonarne
[12] CDC – Connected Device Configuration
[13] CLDC – Connected Limited Device Configuration
[14] MIDP – Mobile Information Device Profile
[15] KVM - K Virtual Machine

Przykładowe okno, przedstawiające przegląd właściwości projektu, wyświetlone dzięki temu dodatkowi przedstawia rys. 5.9. Informacje na temat instalacji i konfiguracji dodatku można znaleźć na stronie: `http://www.eclipse.org/mtj/`.



Rysunek 5.9: Okno konfigurujące właściwości projektu Java ME.

Najważniejszym elementem przygotowywanego środowiska jest J2ME Wireless Toolkit czyli zestaw narzędzi do tworzenia oprogramowania na urządzenia mobilne w którego skład wchodzi między innymi emulator telefonu komórkowego. W chwili pisania tej pracy istniał nowszy zestaw narzędzi o nazwie Java Platform Micro Edition Software Development Kit 3.0. Nie spełniał jednak wymagań tej pracy co do multiplatformowości gdyż jest dostępny w wersji tylko dla systemów Windows. Po pobraniu Java Wireless Toolkit ze strony `http://www.oracle.com/technetwork/java/download-135801.html` oraz jego instalacji można przejść do projektowania aplikacji mobilnej.

5.4.2. Aplikacja mobilna

W ramach tej pracy magisterskiej napisano aplikację działającą na telefonach komórkowych z zainstalowaną maszyną wirtualną Java. Została ona zaprojektowana w taki sposób aby funkcjonalnością przypominać pilota do zdalnego sterowania. Aplikacja korzysta z biblioteki zarządzającej robotem opisanej w rozdziale 5.1.2. Rysunek 5.4 przedstawia uruchomioną aplikację mobilną.

Rysunek 5.10: Przykładowe okno aplikacji sterującej robotem dla JavaME.

Procedura instalacja aplikacji jest uzależniona od modelu telefonu na którym będzie wykorzystywana. Należy pamiętać o włączeniu modułu bluetooth przed uruchomieniem aplikacji sterującej robotem. Po przygotowaniu telefonu można zacząć korzystać z aplikacji mobilnej.

Podsumowanie

Celem pracy był rozwój platformy sprzętowej i programowej robota mobilnego wraz z oprogramowaniem umożliwiającym zdalne sterowanie robotem. Charakter pracy wymagał podzielenia procesu realizacji zadań na klika etapów z których najważniejsze zamieszczone zostały poniżej.

— Zapoznanie się z dotychczasowymi możliwościami robota Dark Explorer oraz analiza prawdopodobnych ścieżek rozwoju.

— Odszukanie niezbędnych narzędzi oraz przygotowanie kompletnego środowiska programistycznego dla systemu Windows oraz Linux które umożliwia rozwój oprogramowania sterującego pracą mikrokontrolera zainstalowanego w robocie.

— Wybór zestawu czujników dodatkowych pozwalających rozszerzyć funkcjonalność robota bez konieczności ingerencji w jego konfigurację bazową.

— Wykonanie obudowy oraz elektroniki umożliwiającej szybkie podłączenie wszystkich czujników do wolnych portów robota.

— Stworzenie modułowego systemu wbudowanego umożliwiającego sterowanie robotem wraz z użyciem dodatkowych czujników.

— Zaprojektowanie i wykonanie bibliotek zewnętrznych umożliwiających swobodne tworzenie oprogramowania sterującego robotem dla urządzeń stacjonarnych i przenośnych.

— Przygotowanie przykładowych aplikacji sterujących pozwalających na zaprezentowanie możliwości robota po zakończeniu prac.

Rozbudowany w ramach pracy magisterskiej robot w pełni realizuje cele założone przez temat pracy. Unowocześniona wersja nie tylko znacząco rozszerza funkcjonalność swego poprzednika ale również dzięki poczynionym modyfikacjom w oprogramowaniu i elektronice robota znacząco ułatwia jego dalszą rozbudowę.

Rozbudowana wersja robota została wyposażona w dodatkowe czujniki wśród których znaleźć można: akcelerometr, żyroskop, magnetometr oraz dalmierze IR. Wszystkie wspomniane sensory są wykorzystywane przez robota do rejestracji oraz rekonstrukcji trasy po której poruszał się operator trzymający robota w ręku. Zaimplementowany algorytm odtwarzania ścieżki nie wymaga do swojego działania żadnej zewnętrznej infrastruktury. Dodatkowym atutem jest fakt, iż robot wykorzystuje czujniki nie tylko podczas nagrywania przebytej drogi, ale również podczas jej odtwarzania. Umożliwia to robotowi analizowanie parametrów środowiska w którym się porusza w czasie rzeczywistym co gwarantuje, że robot będzie w stanie ominąć przeszkody które pojawią się na jego drodze w czasie odtwarzania zapamiętanej trasy. Mechanizm rekonstrukcji ścieżki wymaga od użytkownika, aby po zakończeniu nagrywania ścieżki robot był zwrócony w kierunku powrotnym. Mimo dołożonych starań zaprojektowane rozwiązanie nie gwarantuje, że robot zawsze prawidłowo dotrze do punktu startowego. Wynika to z braku informacji o długości kroku wykonanego przez operatora która na potrzeby obecnego rozwiązania została przyjęta jako wartość stała wyznaczona w sposób doświadczalny i nie zawsze musi odpowiadać stanowi faktycznemu. Dodatkowym utrudnieniem są ruchy wykonywane przez użytkownika w trakcie zapamiętywania trasy takie jak np. nieutrzymywanie czoła robota równolegle do kierunku ruchu operatora. Powoduje to rejestrowanie błędnych parametrów trasy, co skutkuje deformacją odtwarzanego toru ruchu robota podczas powrotu.

Rozbudowana została również warstwa oprogramowania robota. Zebrano i usystematyzowano wiedzę i narzędzia potrzebne do tworzenia oprogramowania uruchamianego na nowej wersji robota Dark Explorer. Zaprojektowano oraz zrealizowano modułową wersję oprogramowania sterującego pracą robota. Poprawia ona działanie dotychczasowych funkcji (np. komunikacja bluetooth, obsługa kamery), ale umożliwia również korzystanie z modułów rozszerzeń i czujników dodanych w trakcie realizacji pracy magisterskiej. Dołożono wszelkich starań, aby możliwe stało się tworzenie aplikacji klienckich nie tylko na urządzenia stacjonarne, ale również mobilne do których zaliczyć można na przykład współczesne telefony komórkowe czy tablety. Zaprojektowane klienckie biblioteki programistyczne umożliwiają korzystanie z funkcji robota, bez konieczności zagłębiania się w szczegóły jego działania.

Podczas pracy nad robotem udało się pokonać szereg problemów związanych z ograniczeniami narzuconymi przez pierwotną konfigurację sprzętową i programową. Mała ilość wejść oraz wyjść zarówno cyfrowych jak i analogowych, została rozszerzona przy pomocy odpowiednio ekspandera GPIO dla interfejsu I^2C oraz multipleksera analogowo–cyfrowego. Przy wykorzystaniu tej samej metody, możliwe jest rozszerzanie wejść/wyjść robota w praktycznie nieograniczony sposób.

Udało się także otrzymać zdjęcia w maksymalnej rozdzielczości, oferowanej przez kamerę wbudowaną w robocie. Umożliwia to wykorzystanie kamery do przeprowadzania bardziej złożonej analizy obrazu, niż było to możliwe w przypadku poprzedniej wersji robota. W ramach pracy podjęto próbę implementacji algorytmu lokalizacji twarzy, który jest uruchamiany bezpośrednio przez mikrokontroler wbudowany w robota co przy obecnej ilości dostępnych zasobów było niebagatelnym wyzwaniem.

Kolejne etapy rozwoju robota mogłyby wiązać się z przyspieszeniem akwizycji oraz transmisji obrazu z kamery. W przypadku transmisji konieczne byłoby wykorzystanie innego interfejsu komunikacji z modułem bluetooth (np. USB) lub całkowita wymiana tego modułu na technologię umożliwiającą wydajniejszą transmisję danych. Przyspieszenie akwizycji obrazu może zostać zrealizowane za pomocą dedykowanego programowalnego układu FPGA lub poprzez rozszerzenie dostępnej pamięci podręcznej mikrokontrolera. Drugie podejście najprawdopodobniej będzie wiązało się z wymianą samego mikrokontrolera, w takiej sytuacji należałoby dokładnie przemyśleć możliwość wymiany mikrokontrolera na szybszy co będzie miało kluczowe znaczenie podczas przetwarzania danych z kamery. W celu polepszenia działania inercjalnego systemu nawigacyjnego, istotne byłoby rozważenie wymiany obecnie używanego akcelerometru na model z lepszymi parametrami oraz podjęcie próby wyznaczenia za jego pomocą dystansu jaki przebył robot. Można również zastanowić się nad wykorzystaniem filtrów Kalmana do eliminowania niedoskonałości zastosowanych czujników i metod uzyskiwania z nich danych. Następnym elementem poprawiającym obecną funkcjonalność robota, może być dołączenie do niego kolejnych czujników odległości, co pozwoliłoby na lepsze omijanie przeszkód, a w przypadku wykorzystania sonaru możliwe stałoby się tworzenie mapy pomieszczenia w którym robot się porusza.

Dodatek A

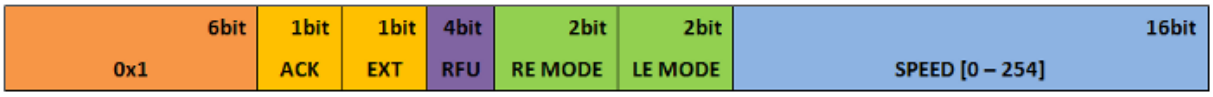
Specyfikacja poleceń protokołu komunikacji

W ramach niniejszego dodatku zostanie przedstawiona szczegółowa dokumentacja protokołu komunikacyjnego zaprojektowanego na potrzeby sterowania robotem Dark Explorer przy użyciu technologii bluetooth. Zasady przesyłania poleceń wraz z ogólnym zarysem sposobu podziału komunikatów został szczegółowo omówiony w ramach rozdziału 4.4. W tej części zaprezentowane zostaną wszystkie dostępne polecenia sterujące wraz z przykładami formatów odpowiedzi w przypadku prawidłowego wykonania polecenia.

A.1. Komunikaty sterujące

W tej części zamieszczone zostały wszystkie komunikaty pozwalające w sposób bezpośredni sterować zachowaniem robota oraz przełączać go pomiędzy różnymi trybami pracy. Każde dostępne polecenie oraz powiązany z nim komunikat został zobrazowane za pomocą graficznej reprezentacji jego logicznej struktury. Dodatkowo każdy element struktury komunikatu został opatrzony testowym komentarzem dokładnie wyjaśniającym jego funkcje. W przypadkach gdy jest to konieczne zmieszczono również komunikat format komunikatu zwrotnego.

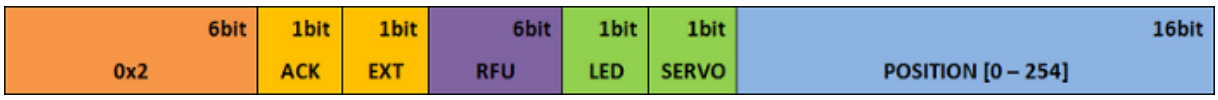
A.1.1. Sterowanie silnikami



Rysunek A.1: Schemat ramki odpowiedzialnej za sterowanie silnikami

ACK Wartość 0x1 jeśli komunikat wymaga potwierdzenia
EXT Wartość 0x1 jeśli komunikat zawiera dane o mocy silników
RFU Bity zarezerwowane do przyszłego użycia
RE MODE Tryb pracy silników prawych (0x0 - stop, 0x1 - wprost, 0x2 - wstecz)
LE MODE Tryb pracy silników lewych (0x0 - stop, 0x1 - wprost, 0x2 - wstecz)
SPEED Moc silników. Wartość z zakresu [0 - 254]

A.1.2. Sterowanie serwomechanizmem



Rysunek A.2: Schemat komunikatu odpowiedzialnego za sterowanie serwomechanizmem

ACK Wartość 0x1 jeśli komunikat wymaga potwierdzenia
EXT Wartość 0x1 jeśli komunikat zawiera dane o położeniu
RFU Bity zarezerwowane do przyszłego użycia
LED Wartość 0x1 jeśli status diody LED ma zostać zmieniony na przeciwny
SERVO Wartość 0x1 jeśli stan serwomechanizm ma zostać zmieniony na przeciwny
POSITION Pozycja (kąt) pod jakim ma zostać ustawiona wieżyczka z kamerą.

A.1.3. Akwizycja obrazu z kamery

Rysunek A.3: Schemat komunikatu odpowiedzialnego za akwizycję obrazu z kamery

ACK Wartość 0x1 jeśli komunikat wymaga potwierdzenia

EXT Wartość 0x1 jeśli komunikat zawiera dane o położeniu

RFU Bity zarezerwowane do przyszłego użycia

TYPE Typ operacji (0x0 - akwizycja obrazu, 0x1 transmisja pobranego obrazu)

MODE Tryb pracy kamery (0x0 - odcienie szarości, 0x1 - kolor)

RES Identyfikator rozmiaru obrazu według następującej specyfikacji:

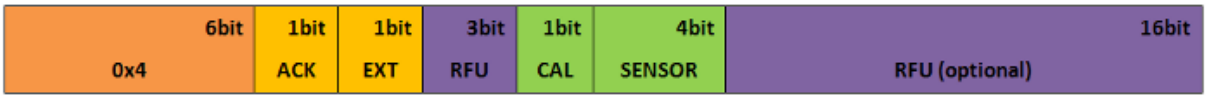
- **0x0** - 160x120
- **0x1** - 320x240
- **0x2** - 640x480

PART NO Numer fragmentu obrazu do pobrania w przypadku włączonego trybu transmisji obrazu

Format transmisji danych obrazu

Ze względu na duży rozmiar danych do przesłania klient musi wysyłać żądania pobrania każdego z elementów oddzielnie. Do zadań klienta należy pobranie wszystkich elementów obrazu i złożenie ich w odpowiedniej kolejności. Zdjęcie przesyłane jest w postaci kolejnych wierszy obrazu pobranego z kamery. Do zakodowania każdego z pikseli obrazu wykorzystany został model kolorów YCbCr.

A.1.4. Sterowanie czujnikami



Rysunek A.4: Schemat komunikatu odpowiedzialnego za odpytywanie czujników

ACK Wartość 0x1 jeśli komunikat wymaga potwierdzenia

EXT Wartość 0x1 jeśli komunikat zawiera dane rozszerzone

RFU Bity zarezerwowane do przyszłego użycia

CAL Ustawienie wartości 0x1 spowoduje uruchomienie procedury kalibrującej dla sensora podanego w sekcji SENSOR

SENSOR Unikalny identyfikator typu pomiaru (czujnika) który ma zostać przeprowadzony, według następującej specyfikacji:

- **0x0** - pomiar napięcia na baterii
- **0x1** - pomiar temperatury
- **0x2** - pomiar przyspieszenia statycznego
- **0x3** - informacje o odległości od najbliższej przeszkody
- **0x4** - informacje o położeniu na podstawie magnetometru
- **0x5** - informacje o kącie obrotu na podstawie żyroskopu[1]

Format odpowiedzi dla poszczególnych czujników

W zależności od wybranego typu czujnika znaczenie oraz typ przesyłanych w odpowiedzi danych może być różny. Dlatego też informacja o formacie danych została zróżnicowana ze względu na rodzaj pomiaru jaki czujnik przeprowadza.

Bateria Na pierwszych 2 bajtach znajduje się wartość napięcia odczytanego na baterii (unsigned short), na kolejnym bajcie znajduje się wartość procentowa reprezentująca poziom naładowania baterii.

Czujnik temperatury Wartość w zakresie [-127, 127] reprezentująca temperaturę w stopniach Celsjusza.

Akcelerometr Kolejne pary bajtów zawierają wartość napięcia (unsigned short) odczytane na kolejnych osiach X, Y, Z czujnika przyspieszenia w wyniku działania przyspieszenia statycznego.

Czujniki odległości Na pierwszych 4 bajtach znajduje się napięcie odczytane kolejno na lewym i prawym czujniku odległości (unsigned short). Kolejne 2 bajty zawierają odległość od przeszkody odpowiednio dla lewego i prawego czujnika wyrażoną w centymetrach (unsigned char).

Magnetometr Kolejne pary bajtów zawierają wartość (short) składowych X oraz Y wektora indukcji pola magnetycznego.

Żyroskop Kolejne pary bajtów zawierają wartość kąta (short) o jaki robot został obrócony odpowiednio wokół osi X, Y oraz Z.

[1] Pierwsze żądanie powoduje wyzerowanie stanu żyroskopu, drugie spowoduje pobranie kąta obrotu zarejestrowanego przez czujnik

A.1.5. Obsługa nagrywania i rekonstrukcji ścieżki powrotnej

Rysunek A.5: Schemat ramki odpowiedzialnej za obsługę nagrywania i rekonstrukcji ścieżki powrotnej

ACK Wartość 0x1 jeśli komunikat wymaga potwierdzenia

EXT Wartość 0x1 jeśli komunikat zawiera dane rozszerzone

RFU Bity zarezerwowane do przyszłego użycia

CMD Unikalny identyfikator akcji która ma zostać wykonana w związku z obsługą algorytmu rekonstrukcji ścieżki, według następującej specyfikacji:

- **0x0** - rozpoczęcie nagrywania ścieżki
- **0x1** - zakończenie nagrywania ścieżki
- **0x2** - usunięcie danych ścieżki
- **0x3** - rozpoczęcie odtwarzania ścieżki
- **0x4** - zakończenie (przerwanie) odtwarzania ścieżki
- **0x5** - przesłanie do klienta danych ścieżki

A.1.6. Obsługa trybu autonomicznego

Rysunek A.6: Schemat ramki odpowiedzialnej za obsługę trybu autonomicznego

ACK Wartość 0x1 jeśli komunikat wymaga potwierdzenia

EXT Wartość 0x1 jeśli komunikat zawiera dane rozszerzone

RFU Bity zarezerwowane do przyszłego użycia

CMD Unikalny identyfikator akcji która ma zostać wykonana w związku z obsługą trybu autonomicznego, według następującej specyfikacji:

- **0x0** - wyłączenie trybu autonomicznego
- **0x1** - włączenie trybu autonomicznego dla kształtów
- **0x2** - włączenie trybu autonomicznego dla twarzy
- **0x3** - pobranie współrzędnych zlokalizowanego obiektu
- **0x4** - pobranie binarnej wersji obrazu

Format komunikatów zwrotnych

Pobieranie zbianryzowanej wersji obrazu powoduje przesłanie strumienia 0 i 1 będących binarną reprezentacją analizowanego obrazu. Obraz przesyłany jest w postaci strumienia, więc aplikacja klienta nie musi wysyłać żądań dla kolejnych części obrazu.

Współrzędne zlokalizowanego obiektu są przesyłane w postaci 2 par punktów pozwalających na jednoznaczne wyznaczenie prostokąta (bounding box) wokół zlokalizowanego na obrazie obiektu.

A.2. Komunikaty specjalne

W tej części zamieszczone zostały komunikaty nie wpływające bezpośrednio na zachowanie robota, ale mające za zadanie przekazanie informacji o aktualnym stanie robota. Znajdują się tutaj między innymi komunikaty informujące o błędach powstałych w czasie wykonania poleceń, jak również takie które pozwalają sprawdzić aktualny stan połączenia oraz robota.

A.2.1. Komunikat powitalny

Celem komunikatu jest sprawdzenie poprawności nawiązanego połączenia. Polecenie może zostać również wykorzystane do podtrzymania aktywności połączenia bluetooth lub sprawdzenia poprawności działania robota po wystąpieniu krytycznego błędu.

Rysunek A.7: Schemat ramki komunikatu powitalnego (kontroli połączenia)

ACK Wartość 0x1 jeśli komunikat wymaga potwierdzenia

EXT Wartość 0x1 jeśli komunikat zawiera dane o mocy silników

RFU Bity zarezerwowane do przyszłego użycia

PATTERN Losowa sekwencja bitów która ma zostać odesłana przez robota w celu potwierdzenia poprawności połączenia i gotowości robota do dalszej pracy.

Definicja odpowiedzi

Prawidłową odpowiedzią na komunikat powitalny powinna być ramka z polem STATE ustawionym na wartość 0 i EOT na 1. W polu SIZE ramki odpowiedzi powinien znaleźć się wzorzec który został przesłany do robota w ramach żądania. Wszystkie nieprawidłowości w stosunku do tej definicji powinny być uznawane za błędy. Polecenie zwrotne powinno zostać odesłane natychmiastowo, w przypadku przedłużania się czasu odpowiedzi należy uznać to za błąd komunikacji.

A.2.2. Komunikaty błędów

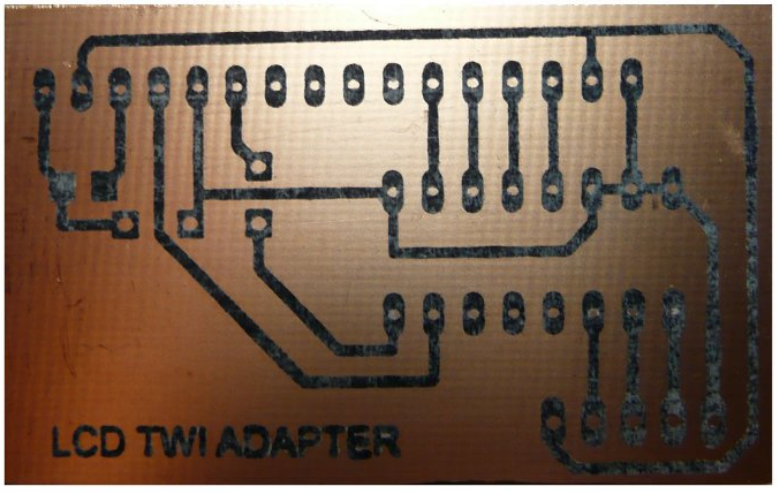
W przypadku wystąpienia błędów ogólnych w działaniu robota lub w działaniu poszczególnych funkcji robot może informować o wystąpieniu błędu za pomocą specjalnego komunikatu. Pakiet zawierający informację o błędzie w swoim nagłówku będzie zawierał identyfikator polecenia które wywołało błąd lub 0x0 w przypadku błędów ogólnych. Dodatkowo w polu STATE i EOT będzie znajdować się 1. W polu SIZE natomiast zamieszczony zostanie unikalny kod błędu identyfikujący problem lub przyczynę powstania błędu.

Dodatek B

Wykonanie płytek drukowanych układów elektronicznych

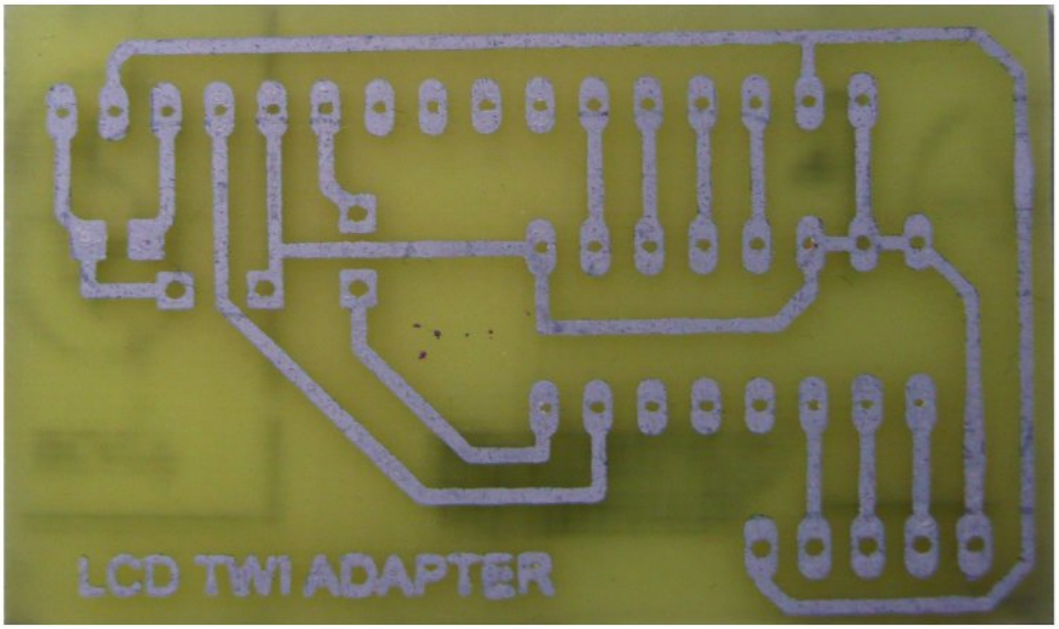
Wszystkie płytki stworzone na rzecz tej pracy magisterskiej zostały wykonane przez autorów pracy w warunkach domowych. W tym dodatku został opisany sposób wytwarzania płytek.

Do wykonania płytek PCB musimy posiadać maskę ścieżek, które umieścimy na laminacie. W tym celu użyto programu Eagle w wersji edukacyjnej z której można korzystać za darmo. Po zaprojektowaniu schematu oraz layoutu płytki maska ścieżek, była drukowana przy pomocy drukarki laserowej na papierze kredowym. Istotne jest dobre przygotowanie layoutu przed drukiem. Jeżeli ścieżki mają znajdować się na górnej stronie płytki, czyli po tej samej stronie po której będą umieszczane elementy elektroniczne, należy stworzyć odbicie lustrzane layoutu. Natomiast w przypadku ścieżek umiejscowionych na warstwie dolnej nie wykonujemy tego kroku. Bardzo przydatnym w tym momencie okazuje się jakiś znacznik, na przykład tekst, który na wydruku powinien zawsze być odbiciem lustrzanym. Tak przygotowany layout drukujemy przy użyciu czarno–białej drukarki laserowej na papierze kredowym.



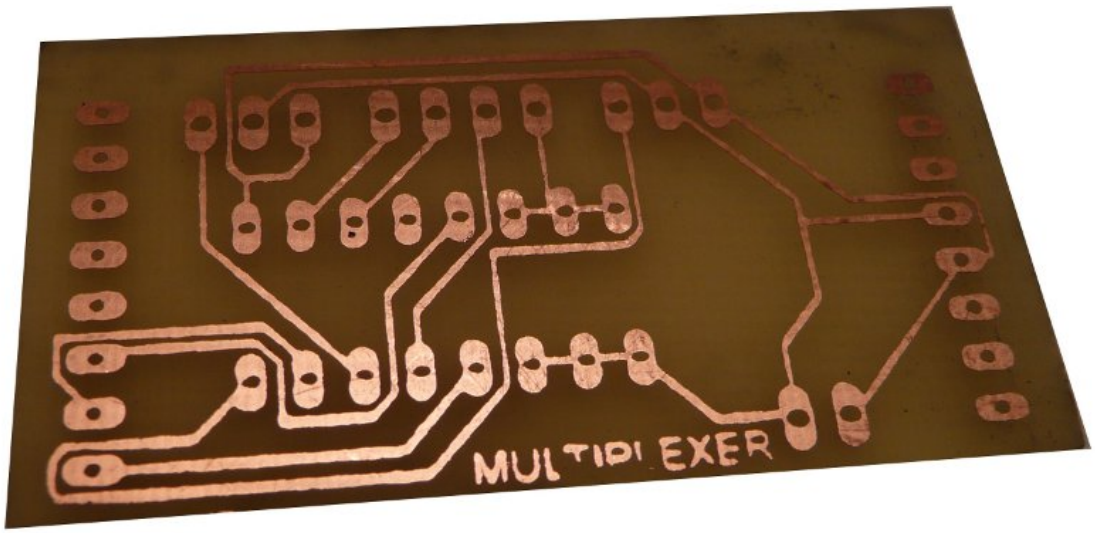
Rysunek B.1: Płytka z naniesionym tonerem

Po wydrukowaniu maski konieczne jest odpowiednie przygotowanie laminatu. W pierwszym kroku używamy papieru ściernego w celu zmatowienia laminatu. Następnie czyścimy laminat płynem do mycia naczyń w celu odtłuszczenia oraz osuszamy go. Po tych czynnościach możemy przystąpić do nakładania maski na laminat. Wycinamy jeden z wydrukowanych layoutów i przykładamy go zadrukowaną stroną do laminatu. Po tych przygotowaniach prasujemy całość żelazkiem rozgrzanym do temperatury $150°C$ około 5 minut. Operacja ta powinna spowodować przeniesienie tonera z papieru na laminat (rys. B.1). W celu usunięcia papieru kredowego wrzucamy płytkę do gorącej wody z dodatkiem proszku do prania. Papier powinien z łatwością się oderwać. Ewentualne przerwania w ścieżkach można poprawić markerem wodoodpornym. Jeżeli uzyskany efekt nie jest satysfakcjonujący powtarzamy całą operację usuwając wcześniej toner przy pomocy zmywacza do paznokci.



Rysunek B.2: Wytrawiona płytka

Istnieje wiele środków trawiących miedź. Autorzy tej pracy stosowali B327 czyli nadsiarczan sodowy. Ten środek trawiący działa najlepiej w temperaturze $50°C$. W celu osiągnięcia optymalnych warunków do trawienia B327, należy rozcieńczyć go w wodzie i umieścić w jakimś naczyniu. Następnie całość wkładamy do drugiego naczynia z gorącą wodą. W takich warunkach trawienie płytki nie powinno trwać dłużej niż 30 minut.



Rysunek B.3: Gotowa płytka

Po zakończeniu procesu trawienia (rys. B.2), należy usunąć toner przy pomocy zmywacza do paznokci i wywiercić otwory montażowe. Dzięki tej metodzie możemy tworzyć płytki z bardzo cienkimi ścieżkami wymagającymi precyzji, której nie da się osiągnąć przy ręcznym rysowaniu maski na laminacie. Kolejnymi atutami tej metody jest praktycznie zerowy koszt produkcji płytki oraz krótki czas realizacji. Płytkę gotową do umieszczenia na niej elementów elektronicznych przedstawia rysunek B.3.

Dodatek C

Plik konfiguracyjny programatora: triton.cfg

```
#define our ports
   telnet_port  4444
   gdb_port  3333

#commands specific

   source [find interface/turtelizer2.cfg]
   source [find board/atmel_at91sam7s-ek.cfg]

   ft2232_device_desc  "Triton JTAG A"
```

Kod źródłowy

W tym załączniku zamieszczony został kod źródłowy części oprogramowania, które zostało stworzone w trakcie realizacji pracy magisterskiej. Wszystkie zamieszczone w tym rozdziale fragmenty kodu, mają charakter poglądowy i służyć mogą jedynie jako podstawowa dokumentacja, pozwalająca zrozumieć architekturę zaimplementowanych rozwiązań. Rozdział ten należy traktować jako uzupełnienie informacji przedstawionych w ramach poprzednich rozdziałów. Czytelnik zainteresowany dostępem do pełnego kodu źródłowego wraz z dokumentacją programisty, może takowy znaleźć na dołączonym do pracy nośniku CD. Wspomnianą dokumentację wraz z najaktualniejszą wersją źródeł można również pobrać ze strony `http://lumifun.ftj.agh.edu.pl/doku.php?id=user:hanusiak_nowacki:start`.

C.1. Dokumentacja kodu źródłowego robota

C.1.1. Moduły urządzeń peryferyjnych

Listing C.1: Interfejs modułu obsługi GPIO (pio_helper.h)

```
1  #ifndef PIO_HELPER_H_
2  #define PIO_HELPER_H_
3
4  #include <at91sam7s256/AT91SAM7S256.h>
5  #include <board.h>
6  #include <pio/pio.h>
7
8  #define CAM_D7          AT91C_PIO_PA27
9  #define CAM_D6          AT91C_PIO_PA20
10 #define CAM_D5          AT91C_PIO_PA19
11 #define CAM_D4          AT91C_PIO_PA18
12 #define CAM_D3          AT91C_PIO_PA17
13 #define CAM_D2          AT91C_PIO_PA16
14 #define CAM_D1          AT91C_PIO_PA22
15 #define CAM_D0          AT91C_PIO_PA24
16
17 #define PIN_CAM_RESET
18 {AT91C_PIO_PA23,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_0,PIO_PULLUP}
19 #define PINS_CAM_DATA
20 {CAM_D0 | CAM_D1 |CAM_D2 | CAM_D3 | CAM_D4 | CAM_D5 | CAM_D6 | CAM_D7,\
21 AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_INPUT,PIO_DEFAULT}
22 #define PIN_CAM_HSYNC
23 {AT91C_PIO_PA26 ,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_INPUT,PIO_DEFAULT}
24 #define PIN_CAM_VSYNC
25 {AT91C_PIO_PA25 ,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_INPUT,PIO_DEFAULT}
26 #define PIN_CAM_PCLK
27 {AT91C_PIO_PA30,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_INPUT,PIO_DEFAULT}
28 #define PIN_CAM_MCLK
29 {AT91C_PA21_PCK1,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_PERIPH_B,PIO_DEFAULT}
30 #define PIN_DIODE1
31 {AT91C_PIO_PA28,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_1,PIO_DEFAULT}
32 #define PIN_DIODE2
33 {AT91C_PIO_PA29,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_1,PIO_DEFAULT}
34 #define PIN_PWM1
35 {AT91C_PA11_PWM0,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_PERIPH_B,PIO_DEFAULT}
36 #define PIN_ENGINE1
37 {AT91C_PIO_PA7,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_0,PIO_PULLUP}
38 #define PIN_ENGINE2
39 {AT91C_PIO_PA8,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_0,PIO_PULLUP}
40 #define PIN_ENGINE3
41 {AT91C_PIO_PA9,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_0,PIO_PULLUP}
```

```
42 #define PIN_ENGINE4
43 {AT91C_PIO_PA10,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_0,PIO_PULLUP}
44 #define PIN_SERWO_PWM
45 {AT91C_PIO_PA0,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_1,PIO_DEFAULT}
46 #define PIN_SERWO_POWER
47 {AT91C_PIO_PA2,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_0,PIO_DEFAULT}
48 #define PIN_LED_POWER
49 {AT91C_PIO_PA1,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_1,PIO_DEFAULT}
50 #define PIN_CD4053B_CTRL
51 {AT91C_PIO_PA15,AT91C_BASE_PIOA,AT91C_ID_PIOA,PIO_OUTPUT_0,PIO_DEFAULT}
52
53 /**
54  * Metoda konfigurujaca wyjscia oraz wejscia
55  * wykorzystywane przez robota
56  */
57 void Configure_DarkExplorer_Pins();
58
59 #endif /* PIO_HELPER_H_ */
```

Listing C.2: Interfejs modułu obsługi bluetooth (usart_helper.h)

```
1  #ifndef USART_HELPER_H_
2  #define USART_HELPER_H_
3
4  /**
5   * Metoda konfigurujaca modul bluetooth
6   *
7   * \param baudrate Szybkosc pracy modulu bluetooth
8   */
9  void UART0_DMA_Configure(unsigned long baudrate);
10
11 /**
12  * Metoda wysylajaca dane za posrednictwem modulu bluetooth
13  *
14  * \param data Bufor z danymi
15  * \param size Ilosc danych do wyslania
16  */
17 void UART0_DMA_Write(char *data, int size);
18
19 #endif /* USART_HELPER_H_ */
```

Listing C.3: Interfejs modułu obsługi ADC (adc_helper.h)

```
1  #ifndef ADC_HELPER_H_
2  #define ADC_HELPER_H_
3
4  #define BOARD_ADC_FREQ 6000000
5  #define ADC_VREF 3300
6
7  /**
8   * Metoda konfigurujaca modul konwertera analogowo-cyfrowego
9   */
10 void ADC_Configure();
11 /**
12  * Metoda rozpoczynajaca konwersje dla pojedynczego kanalu
13  *
14  * \param channel identyfikator kanalu dla ktorego ma sie odbyc konwersja
15  * \param callback metoda uruchamiana po zakonczeniu konwersji
16  */
17 void ADC_StartSingleChannelConversion(unsigned int channel,
18     void(*callback)(unsigned int));
19 /**
20  * Metoda rozpoczynajaca konwersje dla dwoch kanalow
21  *
22  * \param channel_a identyfikator 1 kanalu dla ktorego ma sie odbyc konwersja
23  * \param channel_b identyfikator 2 kanalu dla ktorego ma sie odbyc konwersja
24  * \param callback metoda uruchamiana po zakonczeniu konwersji
25  */
26 void ADC_StartDoubleChannelConversion(unsigned int channel_a,
27     unsigned int channel_b, void(*callback)(unsigned int, unsigned int));
28 /**
29  * Metoda rozpoczynajaca konwersje dla trzech kanalow
30  *
31  * \param channel_a identyfikator 1 kanalu dla ktorego ma sie odbyc konwersja
32  * \param channel_b identyfikator 2 kanalu dla ktorego ma sie odbyc konwersja
33  * \param channel_c identyfikator 3 kanalu dla ktorego ma sie odbyc konwersja
34  * \param callback metoda uruchamiana po zakonczeniu konwersji
35  */
36 void ADC_StartTripleChannelConversion(unsigned int channel_a,
37     unsigned int channel_b, unsigned int channel_c,
38     void(*callback)(unsigned int, unsigned int, unsigned int));
39
40 unsigned char ADC_IsDataReady();
41
42 #endif /* ADC_HELPER_H_ */
```

Listing C.4: Interfejs modułu obsługi PWM (pwm_helper.h)

```
1  #ifndef PWM_HELPER_H_
2  #define PWM_HELPER_H_
3
4  #define PWM_CHANNEL 0
5
6  // Tryb pracy silnikow: STOP
7  #define STOP_GEAR 0
8  // Tryb pracy silnikow: DO PRZODU
9  #define FORWARD_GEAR 1
10 // Tryb pracy silnikow: WSTECZ
11 #define REVERSE_GEAR 2
12
13 // Silniki lewe
14 #define LEFT_ENGINES 1
15 // Silniki prawe
16 #define RIGHT_ENGINES 2
17
18 /**
19  * Konfiguracja kanalu PWM dla silnikow
20  */
21 void PWM_Configure();
22
23 /**
24  * Sterowanie kierunkiem obrotu silnikow
25  *
26  * \param engines identyfikator grupy silnikow
27  * \param gear identyfikator trybu pracy silnikow
28  */
29 inline void PWM_SetGear(unsigned char engines, unsigned char gear);
30
31 /**
32  * Konfiguracja pracy silnikow
33  *
34  * \param gear identyfikator trybu pracy silnikow
35  * \param l_speed predkosc silnikow lewych
36  * \param r_speed predkosc silnikow prawych
37  */
38 inline void ConfigureEngines(unsigned char gear, unsigned char l_speed, unsigned
39
40 #endif /* PWM_HELPER_H_ */
```

C.1.2. Moduły płyty rozszerzeń

Listing C.5: Interfejs modułu obsługi kamery (po6030k.h)

```
1  #ifndef PO6030K_H_
2  #define PO6030K_H_
3
4  // Adres kamery do zapisu przez TWI
5  #define PO6030K_DEVICE_ID 0x6E
6
7  // Pobiera informacje tylko o 25 pinie (VSYNC)
8  #define GET_VSYNC (AT91C_BASE_PIOA->PIO_PDSR & 0x2000000)
9
10 // Pobiera informacje tylko o 26 pinie (HSYNC)
11 #define GET_HSYNC (AT91C_BASE_PIOA->PIO_PDSR & 0x4000000)
12
13 // Pobiera informacje tylko o 30 pinie (PCLK)
14 #define GET_PSYNC (AT91C_BASE_PIOA->PIO_PDSR & 0x40000000)
15
16 #define MAX_BUFF_IDX 3
17 #define SHIFT_BUFF_IDX(idx) (idx = ((idx)+1 < MAX_BUFF_IDX) ? ((idx)+1) : 0)
18 #define PREV_BUFF_IDX(idx) (((idx)-1 < 0) ? MAX_BUFF_IDX-1 : (idx)-1)
19 #define READ_CAM_DATA(data) {
20   register unsigned int buff= (AT91C_BASE_PIOA->PIO_PDSR & 0x95F0000) >> 14;\
21       data = (( buff & 0x7C ) )\
22       | ((buff & 0x2000) >> 6)\
23         | ((buff & 0x400) >> 10)\
24         | ((buff & 0x100) >> 7);\
25    }
26 #define IS_COMMAND_SENT(pSpid) ((pSpid)->pSpiHw->SPI_SR & 0x40)
27 #define RELEASE_SPI_WHEN_READY(pSpid){  while (!IS_COMMAND_SENT(pSpid)); }
28 #define SYNC_WAIT(psync) {  while ((psync)); }
29
30 /**
31  * Metoda inicjalizujaca moduly kamery
32  * wbudowanej w robota
33  */
34 void PO6030K_Initalize();
35 /**
36  * Metoda inicjalizujaca wlasciwosci kamery
37  * odpowiezialne za rodzaj zdjecia
38  *
39  * \param twid Wskaznik do interfejsu TWI
40  */
41 void PO6030K_InitRegisters(Twid *twid);
```

```
42 /**
43  * Metoda pobierajaca zdjecie z kamery i zapisujaca
44  * je w pamieci flash robota
45  */
46 void PO6030K_TakePicture();
47
48 #endif /* PO6030K_H_ */
```

Listing C.6: Interfejs modułu dalmierza (sharp_gp2d12.h)

```
1  #ifndef SHARP_GP2D12_H_
2  #define SHARP_GP2D12_H_
3
4  /*
5   * Na podstawie podanego napiecia i charakterystki
6   * podanej w dokumentacji wylicza odleglosc od przeszkody
7   *
8   * \param output Wartosc napiecia odczytana z dalmierza
9   */
10 unsigned int GP2D12_ComputeDistanceFromCharacterisic(unsigned int output);
11
12 #endif /* SHARP_GP2D12_H_ */
```

Listing C.7: Interfejs modułu magnetometru (mmc212xm.h)

```
1  #ifndef MMC212XM_H_
2  #define MMC212XM_H_
3
4  #define MMC2120M_ADDRESS    0x30
5  #define MMC2120M_CMD_TM     0x1
6  #define MMC2120M_CMD_SET    0x1 << 1
7  #define MMC2120M_CMD_RESET  0x1 << 2
8  #define MMC2120M_MAX_VAL    0x1 << 11
9
10 #define BIT12TO32(b1,b2) -((MMC2120M_MAX_VAL) - ((b1) << 8 | (b2)))
11
12 /**
13  * Struktura z danymi wyjsciowymi z magnetometru
14  */
15 typedef struct _mag_info {
16   // Wartosc skladowej X indukcji pola magnetycznego
17   double x;
18   // Wartosc skladowej Y indukcji pola magnetycznego
19   double y;
20 } mag_info;
```

```
21
22 /**
23  * Metoda pobierajaca dane wysciowe z magnetometru
24  *
25  * \param twid Wskaznik do interfejsu TWI
26  * \return struktura z danymi wyjsciowymi
27  */
28 mag_info MMC212xM_GetMagneticFieldInfo(Twid *twid);
29 /**
30  * Metoda
31  *
32  * \param twid Wskaznik do interfejsu TWI
33  */
34 void MMC212xM_SendSetCmd(Twid *twid);
35 /**
36  * Metoda resetujaca modul magnetometru
37  *
38  * \param twid Wskaznik do interfejsu TWI
39  */
40 void MMC212xM_SendResetCmd(Twid *twid);
41 /**
42  * Metoda kalibrujaca magnetometr
43  *
44  * \param twid Wskaznik do interfejsu TWI
45  */
46 void MMC212xM_Calibrate(Twid *twid);
47
48 #endif /* MMC212XM_H_ */
```

Listing C.8: Interfejs modułu żyroskopu (l3g4200d.h)

```
1  #ifndef L3G4200D_H_
2  #define L3G4200D_H_
3
4  #define L3G4200D_ADDRESS                        0x69
5  #define L3G4200D_CTRL_REG1                      0x20
6  #define L3G4200D_CTRL_REG4                      0x23
7  #define L3G4200D_CMD_CTRL_REG1_POWER_ON         0x0F
8  #define L3G4200D_CTRL_REG5                      0x24
9  #define L3G4200D_CMD_CTRL_REG5_FIFO_EN          0x40
10 #define L3G4200D_FIFO_CTRL_REG                  0x2E
11 #define L3G4200D_CMD_FIFO_CTRL_REG_STREAM_MODE  0x40
12 #define L3G4200D_READ_REG                       0xA6
13 #define L3G4200D_FIFO_SRC_REG                   0x2F
14
```

```
15 #define BIT8TO16(b1,b2) ((b1) << 8 | (b2))
16
17 /**
18  * Struktura z danymi wyjsciowymi z zyroskopu
19  */
20 typedef struct _gyro_data {
21   // Kat obrotu wykryty na osi X
22   short sAngle_x;
23   // Kat obrotu wykryty na osi Y
24   short sAngle_y;
25   // Kat obrotu wykryty na osi Z
26   short sAngle_z;
27
28   // Aktualna temperatura otoczenia
29   char sTemperature;
30   // Maska statusu uzadzenia
31   char status;
32 } gyro_data;
33
34 /**
35  * Metoda pobierajaca dane o obrocie zarejestrowane
36  * przy pomocy zyroskopu
37  *
38  * \return struktura z daymi wyjsciowymi
39  */
40 gyro_data L3G4200D_GetData();
41 /**
42  * Metoda odczytujaca dane o predkosci katowej
43  * z bufora zyroskopu i rejsetrujaca je w lokalnej pamieci
44  *
45  * \param twid Wskaznik do interfejsu TWI
46  */
47 void L3G4200D_ReadData(Twid *twid);
48 /**
49  * Metoda konfigurujaca zyroskop
50  *
51  * \param twid Wskaznik do interfejsu TWI
52  */
53 void L3G4200D_PowerOn(Twid *twid);
54 /**
55  * Metoda uruchamiajaca strumieniowy odczyt
56  * danych z zyroskopu
57  *
58  * \param twid Wskaznik do interfejsu TWI
59  */
```

```
60 void L3G4200D_StreamMode(Twid *twid);
61 /**
62  * Metoda resetujaca dane w zyroskopie oraz
63  * w lokalnej pamieci
64  *
65  * \param twid Wskaznik do interfejsu TWI
66  */
67 void L3G4200D_Reset(Twid *twid);
68 /**
69  * Metoda odpowiedzialna za kalibracje modulu
70  * zyroskopu
71  *
72  * \param twid Wskaznik do interfejsu TWI
73  */
74 void L3G4200D_Calibrate(Twid *twid);
75
76 #endif /* MMC212XM_H_ */
```

Listing C.9: Interfejs modułu obsługi wyświetlacza LCD (hy1602f6.h)

```
1  #ifndef HY1602F6_H_
2  #define HY1602F6_H_
3
4  #define AT24C_ADDRESS   56
5
6  /**
7   * Metoda inicjalizujaca konfiguracje wyswietlacza LCD
8   */
9  void HY1602F6_Init(void);
10 /**
11  * Metoda wypisujaca na ekran tekst podany jako parametr
12  *
13  * \param text tekst do wypisania
14  */
15 void HY1602F6_PrintText(const char* text);
16 /**
17  * Metoda usuwajaca caly tekst z wyswietlacza
18  */
19 void HY1602F6_ClearDisplay(void);
20 /**
21  * Metoda ustawiajaca kursor na podanej pozycji
22  *
23  * \param pos pozycja kursora
24  */
25 void HY1602F6_SetCursorPos(unsigned char pos);
```

```
26 /**
27  * Metoda ustawiajaca kursor na poczatku pierwszego wiersza
28  */
29 void HY1602F6_SetCursorHome(void);
30 /**
31  * Metoda ustawiajaca kursor na poczatku drugiego wiersza
32  */
33 void HY1602F6_StartNextLine();
34 /**
35  * Metoda animujaca pasek postepu zapomoca wyswietlacza
36  *
37  * \param val aktualna wartosc
38  * \param clear 1 jezeli wyswietlacz ma zostac w calosci
39  *          wyczyszczony przed aktualizacja
40  */
41 void HY1602F6_SetProgress(unsigned char val, unsigned char clear);
42 /**
43  * Metoda pozwalajaca na wypisanie dwu linijkowego tekstu
44  *
45  * \param line1 tekst w pierwszej linijce
46  * \param line2 tekst w drugiej linijce
47  */
48 void HY1602F6_Log(const char* line1, const char* line2);
49
50 #endif /* HY1602F6_H_ */
```

Listing C.10: Interfejs modułu akcelerometru (freescale_mma7260.h)

```
1  #ifndef FREESCALE_MMA7260_H_
2  #define FREESCALE_MMA7260_H_
3
4  /**
5   * Struktura opisujaca wartosci przyspieszenia
6   * odczytane za pomoca akcelerometru
7   */
8  typedef struct {
9    // znormalizowana wartosc napiecia dla osi X,Y,Z
10   // wartosc [-850mV, 850mV] odpowiada odpowiednio[-1g, 1g]
11   int x_normal_mv;
12   int y_normal_mv;
13   int z_normal_mv;
14
15   // wartosc napiecia odczytana bezposrednio z akcelerometru
16   unsigned int x_mv;
17   unsigned int y_mv;
```

```
18    unsigned int z_mv;
19  } MMA7260_OUTPUT;
20
21  /**
22   * Metoda korygujaca wartosci odczytane bezposrednio z akcelerometru
23   * o roznice ustalone w procesie kalibracji urzadzenia
24   *
25   * \param channel_x wartosc napiecia odczytana na osi X
26   * \param channel_y wartosc napiecia odczytana na osi Y
27   * \param channel_z wartosc napiecia odczytana na osi Z
28   * \param callback funkcja uruchamiana po odczytaniu danych
29   *          z akcelerometru
30   */
31  void MMA7260_ReadOutput(unsigned int channel_x, unsigned int channel_y,
32      unsigned int channel_z, void(*callback)(MMA7260_OUTPUT));
33
34  #endif /* FREESCALE_MMA7260_H_ */
```

Listing C.11: Interfejs modułu obsługi multipleksera (cd4053b.h)

```
1   #ifndef CD4053B_H_
2   #define CD4053B_H_
3
4   #define CD4053B_X_OUTPUTS 0
5   #define CD4053B_Y_OUTPUTS 1
6
7   /**
8    * Inicjalizajcja multipleksera obslugujacego
9    * analogowa czesc plyty rozszerzen
10   */
11  void CD4053_Initialize();
12  /**
13   * Metoda przelaczajaca odpowiednie wejscia
14   * i wyjsca multipleksera
15   *
16   * \param output_id identyfikator wyjsc multipleksera
17   *          ktore maja zostac wlaczone
18   */
19  void CD4053_EnableOutput(char ouput_id);
20  /**
21   * Metoda przelaczajaca multiplekser w tryb
22   * odbioru danych z akcelerometru
23   */
24  void CD4053_EnableAccelerometer();
25  /**
```

```
26  * Metoda przelaczajaca multiplekser w tryb
27  * odbioru danych z czujnikow odleglosci
28  */
29 void CD4053_EnableIRSensors();
30
31 #endif /* CD4053B_H_ */
```

Listing C.12: Interfejs modułu pamięci flash (at45db321d.h)

```
1  #ifndef AT45DB321D_H_
2  #define AT45DB321D_H_
3
4  #define PAGES_AMOUNT 1500
5  #define BLOCKS_AMOUNT (PAGES_AMOUNT / 8)
6  #define PAGE_SIZE 512
7  #define BLOCK_SIZE (8 * PAGE_SIZE)
8
9  #define AT45DB321_MAX_BUFF_IDX 3
10 #define AT45DB321_SHIFT_BUFF_IDX(idx)
11     ( idx = ((idx)+1 < AT45DB321_MAX_BUFF_IDX) ? ((idx)+1) : 0)
12
13 /**
14  * Metoda inicjalizujaca konfiguracje
15  * wbudowanej pamieci flash
16  */
17 void AT45DB321D_Initalize();
18 /**
19  * Metoda usuwajaca wszystkie dane z pamieci
20  */
21 void AT45DB321D_ClearChip();
22 /**
23  * Metoda przeprowadzajca test poprawnosci
24  * zapisu i odczytu z pamieci
25  */
26 void AT45DB321D_SelfTest();
27 /**
28  * Metoda odczytujaca dane zapisane w pamieci
29  */
30 void AT45DB321D_Read(unsigned char *pBuffer,
31     unsigned int size, unsigned int address);
32 /**
33  * Metoda zwaracajaca wskaznik do pamieci flash
34  */
35 At45* AT45DB321D_GetPointer();
36 /**
```

```
37  * Metoda pobierajaca dane o aktualnym statusie pamieci
38  */
39 inline unsigned char AT45_GetStatus(At45 *pAt45);
40 /**
41  * Wyslanie komendy zapisu do bufora pamieci
42  * przy uzyciu DMA, poprzez interfejs SPI
43  */
44 inline void AT45_WriteBuffer(At45* pAt45, short int siBuffNr,
45       unsigned char* pBuff, unsigned int uiSize);
46
47 #endif /* AT45DB321D_H_ */
```

C.1.3. Moduły algorytmów systemu wbudowanego

Listing C.13: Interfejs modułu omijania przeszkód (obstacle_avoidance.h)

```
1  #ifndef OBSTACLE_AVOIDANCE_H_
2  #define OBSTACLE_AVOIDANCE_H_
3
4  #define STOP_GEAR 0
5  #define FORWARD_GEAR 1
6  #define REVERSE_GEAR 2
7  /**
8   * Struktura opisujaca wyjsciowa konfiguracje silnikow
9   * pozwalajaca na bezpieczne ominiecie przeszkody
10  */
11 typedef struct {
12   // Predkosc silnikow lewych
13   unsigned int speed_left;
14   // Predkosc silnikow prawych
15   unsigned int speed_right;
16
17   // Tryb pracy silnikow lewych
18   unsigned int gear_left;
19   // Tryb pracy silnikow prawych
20   unsigned int gear_right;
21 } OAA_OUTPUT;
22 /**
23  * Struktura przechowywujaca konfiguracje algorytmu
24  * omijania przeszkod
25  */
26 typedef struct {
27   // Predkosc na 1 poziomie odleglosci
28   unsigned int speed_level_1;
29   // Predkosc na 2 poziomie odleglosci
30   unsigned int speed_level_2;
31   // Predkosc na 3 poziomie odleglosci
32   unsigned int speed_level_3;
33
34   // Definicja 1 poziomu odleglosci
35   unsigned int distance_level_1;
36   // Definicja 2 poziomu odleglosci
37   unsigned int distance_level_2;
38   // Definicja 3 poziomu odleglosci
39   unsigned int distance_level_3;
40 } OAA_CONFIG;
41
```

```
42 /**
43  * Metoda inicjalizujaca konfiguracje algorytmu
44  * omijania przeszkod
45  */
46 void init_oa_configuration();
47 /**
48  * Metoda obliczajaca maske poziomow na podstawie danych
49  * o odleglosci robota od przeszkody
50  *
51  * \param rs_data dane o odleglosci z prawego czujnika
52  * \param ls_data dane o odleglosci z lewego czujnika
53  *
54  * \return maska bitowa z opisem poziomow odleglosci
55  */
56 unsigned int create_level_mask(unsigned int rs_data, unsigned int ls_data);
57
58 /**
59  * Metoda wyznaczajaca na podstawie maski poziomow konfiguracje
60  * silnikow pozwlajaca na bezpieczne ominiecie przeszkody
61  *
62  * \param level_mask maska bitowa z opisem poziomow odleglosci
63  * \return konfiguracja silnikow pozwalajaca na ominiecie przeszkody
64  */
65 OAA_OUTPUT avoid_obstacles(unsigned int level_mask);
66
67 #endif /* OBSTACLE_AVOIDANCE_H_ */
```

Listing C.14: Interfejs modułu liczenia kroków (pedometer.h)

```
1  #ifndef PEDOMETER_H_
2  #define PEDOMETER_H_
3
4  #define POSITIVE_PEAK 1
5  #define NEGATIVE_PEAK 2
6  #define ABS(n) (((n) < 0) ? -(n) : (n))
7
8  /**
9   * Struktura przechowywujaca aktualna konfiguracje wykrywacza krokow
10  */
11 typedef struct {
12   // prog napiecia ponizej ktorego
13   // przyspieszenie bedzie traktowane jako ujemne
14   unsigned int negative_thld;
15
16   // prog napiecia powyzej ktorego
```

```
17    // przyspieszenie bedzie traktowane jako dodatnie
18    unsigned int positive_thld;
19
20    // czas w ktorym musi wystapic zmiana +/-
21    // aby zostala uznana za krok
22    unsigned int sample_time;
23  } PEDOMETER_CONFIG;
24
25  /**
26   * Metoda inicjalizujaca konfiguracje algorytmu
27   * do wykrywania krokow
28   */
29  void init_pedometer_config();
30  /**
31   * Metoda uruchamiajaca algorytm zliczania
32   * kolejnych krokow
33   *
34   * \param onStepCallback funkcja wywolywana po wykryciu kroku
35   */
36  void start_steps_detection(void (*onStepCallback)(void));
37  /**
38   * Metoda zatrzymujaca algorytm rozpoznawania
39   * i zliczania krokow
40   */
41  unsigned int stop_counting_steps();
42  /**
43   * Metoda zwracajaca aktualnie wykryta liczbe krokow
44   *
45   * \return liczbe wykrytych w aktualnej sesji krokow
46   */
47  unsigned int get_step_count();
48
49  #endif /* PEDOMETER_H_ */
```

Listing C.15: Interfejs modułu rekonstrukcji ścieżki (reverse_track_reconstruction.h)

```
1  #ifndef REVERSE_TRACK_RECONSTRUCTION_H_
2  #define REVERSE_TRACK_RECONSTRUCTION_H_
3
4  // Maksymalna ilosc krokow jaka moze zapamietac robot
5  #define MAX_STEPS 300
6
7  #define SIGN(n) (((n) < 0) ? -1 : 1)
8  #ifndef ABS
9    #define ABS(n) ( ((n) < 0) ? -(n) : (n))
```

```
10 #endif
11
12 /**
13  * Metoda rozpoczynajaca nagrywanie sciezki
14  */
15 void start_recording_track();
16 /**
17  * Metoda konczaca proces nagrywania sciezki
18  */
19 void stop_recording_track();
20 /**
21  * Metoda rozpoczynajaca proces rekonstrukcji sciezki
22  */
23 void reconstruct_reverse_track();
24
25 #endif /* REVERSE_TRACK_RECONSTRUCTION_H_ */
```

C.2. Dokumentacja biblioteki dla języka Java

Listing C.16: Źródła klasy sterującej (Controller.java)

```
1  /**Klasa zarzadzajaca konkretnymi funkcjonalosciami robota */
2  public class Controller
3  {
4    /** Obiekt klasy zarzadzajacej polaczeniem bluetooth @see BTConnection */
5    private BTConnection btconn;
6    /** Aktualna pozycja wiezy */
7    private int towerPosition;
8    /** Aktualny stan diody oswietleniowej*/
9    private boolean ledState;
10   /** Os ukladu wspolrzednych */
11   public enum Axis {X,Y,Z}
12   /** Identyfikatory dalmierzy */
13   public enum IR {IR_Left, IR_Right};
14
15   /** Konstruktor */
16   public Controller() throws IOException
17
18   /** Funkcja zwracajaca informacje o aktualnej pozycji wiezy z kamera
19    * @return wartosc odchylenia wierzy od poziomu*/
20   public int getTowerPosition()
21
22   /** Zwraca informacje o stanie diody oswietleniowej
23    * @return stan diody oswietleniowej*/
24   public boolean isLenOn()
25
26   /** Zatrzymanie silnikow */
27   public void stopEngines() throws IOException
28
29   /** Poruszanie robotem do przodu */
30   public void goForward() throws IOException
31
32   /** Poruszanie robotem do tylu */
33   public void goBackward() throws IOException
34
35   /** Poruszanie robotem w lewo */
36   public void turnLeft() throws IOException
37
38   /** Poruszanie robotem w prawo */
39   public void turnRight() throws IOException
40
41   /** Zmiana stanu diody oswietleniowej wlacz/wylacz */
```

```
42    public void ledSwitch() throws IOException
43
44    /** Podnies wieze */
45    public void moveTowerUp() throws IOException
46
47    /** Opusc wieze */
48    public void moveTowerDown() throws IOException
49
50    /** Wylacz zasilanie serwomechanizmu wiezy */
51    public void towerPowerOff() throws IOException
52
53    /** Pobiera okreslony obraz z robota za pomoca modulu bluetooth.
54     * Funkcja bedzie czekala az robot przesle okreslona ilosc danych
55     * @param cmd komenda inicjalizujaca wysylanie obrazu na robocie
56     * @param packSize rozmiar paczki z danymi
57     * @param width szerokosc pobieranego obrazu
58     * @param height wysokosc pobieranego obrazu
59     * @param isColor parametr okreslajacy to czy obraz ma byc kolorowy czy tez w od
60     * @return tablica bajtow z pikselami obrazu*/
61    private byte[] getImage(char cmd, int packSize, int width, int height, boolean i
62
63    /** Rozmiar paczki do pobrania z robota */
64    public static final int packSize = 1280;
65
66    /** Pobiera kolorowy obraz 640x480 z robota
67     * @return tablica bajtow z pikselami obrazu*/
68    public byte[] getImageFullResColor() throws IOException
69
70    /** Pobiera kolorowy obraz 320x240 z robota
71     * @return tablica bajtow z pikselami obrazu*/
72    public byte[] getImageHalfResColor() throws IOException
73
74    /** Pobiera kolorowy obraz 160x120 z robota
75     * @return tablica bajtow z pikselami obrazu*/
76    public byte[] getImageQuaterResColor() throws IOException
77
78    /** Pobiera obraz 640x480 w odcieniach szarosci
79     * @return tablica bajtow z pikselami obrazu*/
80    public byte[] getImageFullRes() throws IOException
81
82    /** Pobiera obraz 320x240 w odcieniach szarosci
83     * @return tablica bajtow z pikselami obrazu*/
84    public byte[] getImageHalfRes() throws IOException
85
86    /** Pobiera obraz 160x120 w odcieniach szarosci
```

```
87      * @return tablica bajtow z pikselami obrazu*/
88     public byte[] getImageQuaterRes() throws IOException
89
90     /** Wykrywa i pobiera kolorowy obraz w rozdzielczosci 640x480
91      * @return tablica bajtow z pikselami obrazu*/
92     public byte[] getImageWithFaceDetection() throws IOException
93
94     /** Pobiera informacje o temperaturze
95      * @return temperatura*/
96     public double getBatteryStatus()
97
98     /** Pobiera dane z czujnika przyspieszenia
99      * @param ax okresla os z ktorej chcemy uzyskac informacje
100     * @return odczyt z danej osi czujnika*/
101    public double getAccelerometerAxisData(Axis ax)
102
103    /** Uruchamianie procedury kalibracji akcelerometru*/
104    public void calibrateAccelerometer()
105
106    /** Pobiera dane z zyroskopu
107     * @param ax okresla os z ktorej chcemy uzyskac informacje
108     * @return odczyt z danej osi czujnika*/
109    public double getGyroscopeAxisData(Axis ax)
110
111    /** Uruchamianie procedury kalibracji zyroskopu*/
112    public void calibrateGyroscope()
113
114    /** Pobiera dane z czujnika odleglosci
115     * @param ir okresla czujnik z ktorgo chcemy uzyskac informacje
116     * @return odczyt z danego czujnika */
117    public double getIRData(IR ir)
118
119    /** Pobiera dane z zyroskopu
120     * @param ax okresla os z ktorej chcemy uzyskac informacje
121     * @return odczyt z danej osi czujnika */
122    public double getMagnetometerData(Axis ax)
123
124    /** Uruchamianie procedury kalibracji magnetometru */
125    public void calibrateMagnetometer()
126
127    /** Pobiera dane z czujnika temperatury
128     * @return odczyt czujnika*/
129    public double getTemperature()
130
131    /** Wyswietla tekst na ekranie LCD robota
```

```
132     * @param lcdText tekst do wyswietlenia*/
133   public void setLCDText(String lcdText)
134
135   /** Wlaczenie trybu autonomicznego
136    * @param mode rodzaj trybu autonomicznego*/
137   public void runAutonomousMode(int mode)
138
139   /** Wlaczenie nagrywania sciezki do algorytmu rekonstrukcji trasy powrotnej*/
140   public void startRecordingTrack()
141
142   /** Wylaczenie nagrywania sciezki dla algorytmu rekonstrukcji trasy powrotnej*/
143   public void stopRecordingTrack()
144
145   /** Uruchomienie algorytmu rekonstrukcji sciezki powrotnej*/
146   public void reconstructTrack()
147 }
```

Listing C.17: Źródła klas obsługi połączenia bluetooth (BTConnection.java)

```
1  /** Singleton obslugujacy polaczenie bluetooth*/
2  public class BTConnection {
3
4    /** Instancja klasy */
5    private static BTConnection instance = null;
6    /** Strumien polaczenia */
7    private StreamConnection stream = null;
8    /** String polaczenia z adresem MAC oraz portem Dark Explorera */
9    private final String url = "btspp://00126F0437A2:1";
10   /** Strumien wyjscowy */
11   private OutputStream outstream = null;
12   /** Strumien wejscowy */
13   private InputStream instream = null;
14
15   /** Konstruktor */
16   private BTConnection()
17
18   /** Metoda zwracajaca instancje singletonu zarzadzajacego polaczeniem bluetooth
19    * @return instancja klasy BTConnection*/
20   public static BTConnection getInstance()
21
22   /** Nawiazywanie polaczenia*/
23   public void createNewConnection() throws IOException
24
25   /** Wysylanie danych do robota
26    * @param c komenda
```

```
27     * @param d dane
28     * @throws IOException*/
29    public void writeBtData(char c,int d) throws IOException
30
31    /** Odbieranie danych od robota
32     * @param data odebrane dane
33     * @param currOffset obecny offset (ile danych odebrano do tej pory)
34     * @param len dlugosc paczki do odebrania*/
35    public void readBtData(byte[] data, int currOffset, int len) throws IOException
36
37    /** Zamykanie polaczenia bluetooth*/
38    public void closeConnection()
39 }
```

C.3. Dokumentacja biblioteki dla języka C#

Listing C.18: Źródła klasy sterującej (DarkExplorer.cs)

```
1  public class DarkExplorer
2  {
3    /// Obsluga zdarzenia odebrania danych nagranej sciezki
4    public event EventHandler TrackDataReceived;
5    /// Obsluga zdarzenia odebrania danych na temat temperatury
6    public event EventHandler TemperatureDataReceived;
7    /// Obsluga zdarzenia odebrania danych pomiarowych
8    /// z akcelerometru
9    public event EventHandler AccelerationDataReceived;
10   /// Obsluga zdarzenia odebrania danych pomiarowych
11   /// z dalmierzy
12   public event EventHandler RangefinderDataReceived;
13   /// Obsluga zdarzenia odebrania danych pomiarowych
14   /// z magnetometru
15   public event EventHandler MagnetometerDataReceived;
16   /// Obsluga zdarzenia odebrania danych
17   /// pomiarowych z zyroskopu
18   public event EventHandler GyroscopeDataReceived;
19   /// Obsluga zdarzenia odebrania rezultatow
20   /// przeprowadzonej analizy obrazu
21   public event EventHandler DetectionResultReceived;
22   /// Obsluga zdarzenia odebrania potwierdzenia realizacji komendy
23   public event EventHandler AcknowledgementReceived;
24   /// Obsluga zdarzenia odebrania danych przeslanych z robota
25   public event EventHandler DataPacketReceived;
26   /// Obsluga zdarzenia odebranie danych o baterii
27   public event EventHandler PowerLevelReceived;
28   /// Obsluga zdarzenia odebrania danych z kamery
29   public event EventHandler CameraDataReceived;
30   /// Obsluga zdarzenia otwarcia polaczenia
31   public event EventHandler ConnectionOpened;
32   /// Obsluga zdarzenia zamkniecia polaczenia
33   public event EventHandler ConnectionClosed;
34
35
36   /// Domyslny konstruktor
37   /// <param name="port">
38   ///   Identyfikator portu na ktorym odbywac sie bedzie komunikacja
39   /// </param>
40   public DarkExplorer(string port);
41
```

```
42    /// Metoda wysyla zadanie pobrania danych
43    /// z czujnika podanego jako parametr
44    /// <param name="type">
45    ///   Typ czujnika za pomoca ktorego
46    ///   ma zostac przeprowadzony pomiar
47    /// </param>
48    public void RequestSensorData(SensorType type);
49    /// Metoda wysyla zadanie przeprowadzenia
50    /// kalibracji czujnika podanego jako parametr
51    /// <param name="type">
52    ///   Typ czujnika dla ktorego ma zostac przeprowadzona kalibracja
53    /// </param>
54    public void RequestSensoreCalibartion(SensorType type);
55    /// Metoda wysylajaca do robota komende obslugi
56    /// trybu rekonstrukcji sciezki powrotnej
57    /// <param name="command">
58    ///   Komenda rekonstrukcji sciezki powrotnej
59    /// </param>
60    public void SendTrackReconstructionCommand(TrackReconstructionCommand c);
61    /// Metoda wysylajaca do robota komede
62    /// obslugi trybu autonomicznego
63    /// <param name="command">
64    ///   Komenda obslugi trybu autonomicznego
65    /// </param>
66    public void SendAutonomousModeCommand(AutonomousModeCommand command);
67    /// Metoda konfigurujaca dzialanie prawych silnikow
68    /// <param name="direction">Tryb pracy silnikow</param>
69    /// <param name="speed">Moc silnikow</param>
70    public void SetRightEnginesState(EnginesDirection direction, byte speed);
71    /// Metoda konfigurujaca dzialanie lewych silnikow
72    /// <param name="direction">Tryb pracy silnikow</param>
73    /// <param name="speed">Moc silnikow</param>
74    public void SetLeftEnginesState(EnginesDirection direction, byte speed);
75    /// Metoda zatrzymujaca silniki
76    public void StopEngines();
77    /// Metoda wlaczajaca i wylaczajaca diode oswietleniowa LED
78    /// <param name="state">
79    ///   stan diody (true - wlaczona, false - wylaczona)
80    /// </param>
81    public void SetDiodeState(bool state);
82    /// Metoda wylaczajaca serwomechanizm
83    public void DisableServomechanism();
84    /// Metoda wlaczajaca serwomechanizm i ustawiajaca
85    /// podana jako parametr pozycje
86    /// <param name="position">Pozycja serwomechanizmu</param>
```

```
87    public void SetSerwoPosition(byte position);
88    /// Metoda wysylajaca zadanie pobrania danych
89    /// o stanie naladownia baterii
90    public void SendPowerLevelRequest();
91    /// Metoda otwierajaca polaczenie z robotem
92    public void OpenConnection();
93    /// Metoda zamykajaca port na ktorym odbywa sie komunikacja
94    public void CloseConnection();
95    /// Metoda zwracajaca informacje ta temat
96    /// gotowosci portu do transmisji danych
97    public bool IsConnectionActive;
98 }
```

Listing C.19: Źródła klas obsługi protokołu komunikacji (DarkExplorerComm.cs)

```
1  /// Klasa reprezentujaca polecenia wysylane
2  /// do robota za posrednictwem bluetooth
3  public class Request {
4    /// Unikalny typ polecenia zwiazanego z zadaniem
5    public RequestType CommandType { get; set; }
6    /// Flaga informujaca o koniecznosci potwierdzenia odebrania pakietu
7    public bool RequireAcknowledgement { get; set; }
8    /// Flaga informujaca o tym czy polecenie
9    /// posiada sekcje rozszerzajaca parametry podstawowe
10   public bool HasExtendedData { get; set; }
11   /// Maska parametrow polecenia
12   public byte ParamMask { get; set; }
13   /// Rozszerzone parametry polecenia
14   public byte[] ExtendedParamData { get; set; }
15 }
16
17 /// Klasa reprezntujaca odpowiedz wyslana przez robota
18 public class Response {
19   /// Naglowek odpowiedzi
20   public ResponseHeader Header { get; set; }
21   /// Dane przesylane w ramach sekcji DATA
22   public byte[] ResponseData { get; set; }
23 }
24 /// Klasa reprezentujaca naglowek odpowiedzi robota
25 public class ResponseHeader {
26   /// Unikalny typ polecenia zwiazanego z rzadaniem
27   public RequestType CommandType { get; set; }
28   /// Flaga informujaca o wystapieniu bledow
29   /// podczas realizacji polecenia
30   public bool HasErrorsOccured { get; set; }
```

```
31    /// Flaga informujaca o koncu transmisji
32    public bool IsEndOfTransmission { get; set; }
33    /// Ilosc danych przesylanych w sekcji DATA
34    public byte DataSize { get; set; }
35  }
36  /// Klasa umozliwiajaca konwersje pomiedzy instancjami
37  /// obiektow a strumieniem bajtow do przeslania
38  public class CommunicationProcessor {
39    /// Metoda kodujaca zadanie do postaci tablicy bajtow
40    /// ktora moze zostac przeslana bezposrednio do robota
41    /// <param name="request">Instancja obiektu zadnia</param>
42    /// <returns>
43    ///   Tablica bajtow z zakodowanym zadaniem do przeslania
44    /// </returns>
45    public static byte[] EncodeRequest(Request request);
46    /// Metoda dekodujaca strumien bajtow przeslanych
47    /// przez robota jako naglowek odpowiedzi
48    /// <param name="headerStream">
49    ///   Tablica bajtow z naglowkiem odpowiedzi
50    /// </param>
51    /// <returns>Naglowek odpowiedzi po zdekodowaniu</returns>
52    public static ResponseHeader DecodeResponseHeader(byte[] stream);
53  }
```

Listing C.20: Źródła klas typów pomocniczych (DarkExplorerTypes.cs)

```
1   /// Typ wyliczeniowy reprezentujacy rozmiar obrazu
2   /// jaki ma zostac pobrany z kamery
3   public enum ImageSize {
4     /// Obrazek maly rozmiar 160x100
5     Small = 0,
6     /// Obrazek sredni rozmiar 320x200
7     Medium = 1,
8     /// Obrazek duzy rozmiar 640x400
9     High = 2
10  }
11  /// Typ wyliczeniowy reprezentujacy tryb pracy silnikow
12  public enum EnginesDirection {
13    /// Tryb postoju
14    Stop = 0,
15    /// Tryb jazdy do przodu
16    Straight = 1,
17    /// Tryb jazdy wstecz
18    Back = 2
19  }
```

```
20 /// Typ wyliczeniowy reprezentujacy typy pomiaru
21 public enum SensorType {
22   /// Pomiar napiecia na baterii
23   Battery = 0,
24   /// Pomiar temperatury
25   Thermometer = 1,
26   /// Pomiar przyspieszenia
27   Accelerometer = 2,
28   /// Pomiar odleglosci
29   Rangefinder = 3,
30   /// Pomiar indukcji pola magnetycznego
31   Magnetometer = 4,
32   /// Pomiar kata obrotu
33   Gyroscope = 5
34 }
35 /// Typ wyliczeniowy reprezentujacy polecenia obslugi
36 /// trybu rekonstrukcji sciezki
37 public enum TrackReconstructionCommand {
38   /// Rozpoczecie nagrywania sciezki
39   StartRecordingTrack = 0,
40   /// Koniec nagrywania sciezki
41   StopRecordingTrack = 1,
42   /// Usuniecie nagranej sciezki
43   EraseRecordedTrack = 2,
44   /// Rozpoczecie rekonstrukcji sciezki
45   StartTrackReconstruction = 3,
46   /// Przerwanie rekonstrukcji sciezki
47   StopTrackReconstruction = 4,
48   /// Przeslanie informacji o sciezce
49   SendTrackData = 5,
50 }
51 /// Typ wyliczeniowy reprezentujacy polecenia
52 /// obslugi trybu autonomicznego
53 public enum AutonomousModeCommand {
54   /// Wylaczenie trybu autonomicznego
55   DisableAutonomousMode = 0,
56   /// Tryb autonomiczny - ksztalty
57   EnableShapeDetection = 1,
58   /// Tryb autonomiczny - twarz
59   EnableFaceDetection = 2,
60   /// Przeslanie wspolrzednych obiektu
61   SendDetectionResult = 3,
62   /// Przeslanie zbinaryzowanego obrazu
63   SendBinarizedImage = 4
64 }
```

```
65 /// Typ wyliczeniowy reprezentujacy
66 /// dostepne rodzaje polecen sterujacych
67 public enum RequestType {
68   /// Test polaczenia
69   Welcome = 0,
70   /// Sterowanie silnikami
71   Engines = 1,
72   /// Sterowanie serwomechanizmem
73   Servomechanism = 2,
74   /// Sterowanie kamera
75   Camera = 3,
76   /// Sterowanie czujnikami
77   Sensors = 4,
78   /// Sterowanie rekonstrukcja trasy
79   TrackReconstruction = 5,
80   /// Sterowanie trybem autonomicznym
81   AutonomousMode = 6
82 }
```

Listing C.21: Źródła klas zdarzeń (DarkExplorerEvents.cs)

```
1  /// Zdarzenie otrzymania danych prostokata ograniczajacego
2  /// wyznaczonego w procesie analizy obrazu
3  public class DetectionResultReceivedEventArgs : EventArgs {
4    /// Minimalna wspolrzedna X prostokata ograniczajacego
5    public int MinX { get; private set; }
6    /// Minimalna wspolrzedna Y prostokata ograniczajacego
7    public int MinY { get; private set; }
8    /// Maksymalna wspolrzedna X prostokata ograniczajacego
9    public int MaxX { get; private set; }
10   /// Maksymalna wspolrzedna Y prostokata ograniczajacego
11   public int MaxY { get; private set; }
12 }
13 /// Zdarzenie otrzymania danych o nagranej sciezce
14 public class TrackDataReceivedEventArgs : EventArgs {
15   /// Dane nagranej sciezki
16   public int[] PathData { get; private set; }
17 }
18
19 /// Zdarzenie otrzymania danych o przyspieszeniu
20 public class AccelerationDataReceivedEventArgs : EventArgs {
21   /// Wartosc przyspieszenia odczytana na osi X
22   public double AxisX { get; private set; }
23   /// Wartosc przyspieszenia odczytana na osi Y
24   public double AxisY { get; private set; }
```

```
25    /// Wartosc przyspieszenia odczytana na osi Z
26    public double AxisZ { get; private set; }
27  }
28
29  /// Zdarzenie otrzymania danych o odleglosci
30  /// od przeszkody
31  public class RangefinderDataReceivedEventArgs : EventArgs {
32    /// Dane pomiarowe z lewego czujnika odleglosci
33    public int LeftSensorData { get; private set; }
34    /// Dane pomiarowe z prawego czujnika odleglosci
35    public int RightSensorData { get; private set; }
36  }
37
38  /// Zdarzenie otrzymania danych pomiarowych z magnetometru
39  public class MagnetometerDataReceivedEventArgs : EventArgs {
40    /// Skladowa X wektora indukcji pola magnetycznego
41    public int ValueX { get; private set; }
42    /// Skladowa Y wektora indukcji pola magnetycznego
43    public int ValueY { get; private set; }
44  }
45
46  /// Zdarzenie otrzymania danych pomiarowych z zyroskopu
47  public class GyroscopeDataReceivedEventArgs : EventArgs {
48    /// Kat obrotu wykryty na osi X
49    public int AngleX { get; private set; }
50    /// Kat obrotu wykryty na osi Y
51    public int AngleY { get; private set; }
52    /// Kat obrotu wykryty na osi Z
53    public int AngleZ { get; private set; }
54  }
55
56  /// Zdarzenie otrzymania danych o temperaturze
57  public class TemperatureDataReceivedEventArgs : EventArgs {
58    /// Wartosc temperatury odczytanej za pomoca robota
59    public int Temperature { get; private set; }
60  }
61
62  /// Zdarzenie otrzymania potwierdzenia realizacji wyslanej komendy
63  public class AcknowledgementReceivedEventArgs : EventArgs {
64    /// Rodzaj komendy ktorej dotyczy powtwierdzenie
65    public RequestType Type { get; private set; }
66    /// Rezultat zakonczenia akcji
67    public Boolean State { get; private set; }
68  }
69
```

```
70 /// Zdarzenie otrzymania danych z kamery
71 public class CameraDataEventArgs : EventArgs {
72   /// Strumien danych zdjecia wykonanego za pomoca robota
73   public byte[] RawInputData { get; private set; }
74   /// Obraz w postaci mapy bitowej
75   public Bitmap BmpImage { get; private set; }
76   /// Rozmiar obrazu
77   public ImageSize Size { get; private set; }
78 }
79
80 /// Zdarzenie otrzymania danych o baterii
81 public class PowerLevelEventArgs : EventArgs {
82   /// Wartosc napiecia zmierzona na bateriach
83   public double Voltage { get; private set; }
84   /// Poziom naladowania baterii
85   public double PowerLevel { get; private set; }
86 }
87
88 /// Zdarzenie otwarcia polaczenia
89 public class ConnectionOpenedEventArgs : EventArgs {
90   /// Port na ktorym polaczenie zostalo otwarte
91   public SerialPort Port { get; private set; }
92 }
93
94 /// Zdarzenie zamkniecia polaczenia
95 public class ConnectionClosedEventArgs : EventArgs {
96   /// Port dla ktorego polaczenie zostalo zamkniete
97   public SerialPort Port { get; private set; }
98 }
```

Spis rysunków

Spis tabel

Bibliografia

[1] Paweł Kmak. Rozwój systemu sterowania dla robota mobilnego. Praca magisterska, Wydział Fizyki i Informatyki Stosowanej, 2009.

[2] Isaac Assimov. *Runaround*. Astounding Science Fiction, 1942.

[3] dr inż. Tomasz Buratowski. Teoria robotyki. `http://www.robotyka.com/teoria\_spis.php`, 2010.

[4] Wortal robotyki. `http://www.asimo.pl/teoria/robotyka.php`, 2010.

[5] Monika Jaworowska. Żyroskopy i akcelerometry mems w elektronice użytkowej. `http://elektronikab2b.pl/technika/`, 2010.

[6] Marcin Pomianowski. Kieszonkowy akcelerometr - miernik przyśpieszenia xyz. Elektronika praktyczna nr 2/2010, 2010.

[7] Przemysław Szulim Piotr Kleczyński. Praktyczne zastosowanie mems-ów w platformach bezzałogowych. `http://elektronikab2b.pl/technika/`, 2010.

[8] Hokuriku Electric Industry. Piezoresistive type 3-axis acceleration sensor application note. http://www.hdk.co.jp, 2007.

[9] Freescale Semiconductor. *Technical Data of Three Axis Low-g Micromachined Accelerometer MMA7260QT*. Freescale Semiconductor, 2008.

[10] Monika Jaworowska. Żyroskopy i akcelerometry mems w elektronice użytkowej. `http://elektronikab2b.pl/technika/12098-yroskopy-i-akcelerometry-mems-w-elektronice-uytkowej`, 2010.

[11] STMicroelectronics. Mems motion sensor:ultra-stable three-axis digital output gyroscope datasheet. `http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00265057.pdf`, 2010.

[12] Dora Sumisławska Adrian Ciz, Marek Gulanowski. Wstepny projekt modułu imu. `http://konar.ict.pwr.wroc.pl/uploads/download/raporty/IMU.pdf`, 2011.

[13] Inc. MEMSIC. *Dual-axis Magnetic Sensor with I2C Interface.* MEMSIC, Inc., 2008.

[14] A. Gacsadi L. Tepelea I. Gavrilut, V. Tiponut. *Obstacles Avoidance Method for an Autonomous Mobile Robot using Two IR Sensors.* University of Oradea, Politehnica University of Timisoara, 2008.

[15] Acroname robotics. `http://www.acroname.com/robotics/info/articles/sharp/sharp.html`, 2010.

[16] Jacek Augustyn. *Projektowanie systemów wbudowanych na przykładzie rodziny SAM7S z rdzeniem ARM7TDMI.* Wydawnictwo IGSMiE PAN, 2007.

[17] Sharp Corporation. *GP2D12 Optoelectronic Device Datasheet.* Sharp Corporation, 2006.

[18] Atmel Corporation. Biblioteka at91lib. `http://www.atmel.com/dyn/resources/prod_documents/at91lib_20100901_softpack_1_5_svn_v8476.zip`, 2011.

[19] Atmel Corporation. *Using Atmel's DataFlash.* Atmel Corporation, 2002.

[20] Atmel Corporation. *32-megabit 2.7-volt Only DataFlash.* Atmel Corporation, 2003.

[21] Yen-Chun Lin Yao-Jiunn Chen. *Simple Face-detection Algorithm Based on Minimum Facial Features.* The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON), 2007.

[22] Java bluetooth.com - development kits. `http://www.javabluetooth.com`, 2011.

[23] Bluecove jsr-82 project. `http://bluecove.org`, 2011.

[24] Eugene Kordin. Windows mobile application development. `http://www.mono-project.com/HowToSystemIOPorts`, 2010.