Projekt Indywidualny na kierunku Informatyka

Politechnika Warszawska

Wydział Elektryczny



Zaawansowane sterowanie robotem LEGO NXT

Daniel Król, krold@ymail.com

Semestr V, Informatyka / Inżynieria Komputerowa

26.01.2009

Prowadzący : dr inż. Witold Czajewski

Spis treści

1 Wprowadzenie **3**

1.1 Tematyka projektu 3

1.2 Zdefiniowany cel projektu 3

1.3 Co to jest LEGO NXT ? 4

1.4 Co NXT oferuje ? 5

2 Przygotowania i możliwe rozwiązania **6**

2.1 Sposób sterowania 6

2.2 Języki programowania z podziałem na lokalne i zdalne 7

2.3 Sposób lokalizacji w labiryncie 8

2.4 Wybór rozwiązania 9

3 Realizacja **10**

3.1 Budowa robota 10

3.2 Implementacja 12

3.2.1 Kod dla robota 13

3.2.2 Kod PC 14

3.3 Protokół komunikacji 15

3.4 Opis uruchomieniowy 17

4 Podsumowanie, wnioski i możliwości rozwoju **21**

5 Kod źródłowy **23**

5.1 mazesolv.nxc 23

5.2 MazeSolver.cs 30

1 Wprowadzenie

1.1 Tematyka projektu

Przedmiotem projektu jest sterowanie robotem opartym na platformie LEGO NXT. Temat projektu nie obejmuje żadnych konkretnych działań, ani sposobów realizacji, tak więc dobór celu oraz sposobu wykonania również należy zaliczyć do tematyki projektu.

1.2 Zdefiniowany cel projektu

Celem projektu jest, by robot był w stanie przejechać z punktu A do punktu B w labiryncie. Przy czym położenie startowe i końcowe to jedyne konkretne informacje jakie robot otrzymuje z zewnątrz. Orientacja w labiryncie oraz badanie rozmieszczenia ścian labiryntu należą do zadań robota. W celu ułatwienia zadania przyjęte zostało, że labirynt będzie składał się z jednobarwnych, kwadratowych pól, z których każde zostanie obwiedzione kolorystycznie kontrastujacym z polami paskiem.

1.3 Co to jest LEGO NXT ?

NXT jest to zestaw pozwalający na tworzenie prostych robotów, oficjalną nazwą zestawu jest LEGO Mindstorms. De facto NXT stanowi rozwinięcie serii klocków LEGO Technics poprzez dodanie programowalnego klocka NXT Brick oraz calej gamy peryferiów współpracujących z nim.
Wsród peryferiów tych znajdują się : silniki z wbudowanymi enkoderami, czujniki natężenia światła, dotyku, dźwięku oraz ultradźwiękowe czujniki odległości, ponadto oprócz standardowych peryferiów LEGO do dyspozycji są również sensory innych producentów, m.in. : akcelerometry, żyroskopy i inne. Zastosowanie standardowej magistrali I2C oraz wejścia liniowego pozwala na tworzenie własnych autorskich czujników i aktorów.
NXT można nazwać platformą sprzętową do budowy robotów zarówno poprzez wzgląd na szeroki wachlarz peryferiów, jak i na możliwości samego NXT Brick. Głównym procesorem NXT jest 32 bitowy procesor z rodziny ARM7, pracujący z czestotliwością 48 MHz, co pozwala efektywnie wykonać kilka tysięcy instrukcji kodu NBC na sekundę.
NBC (NeXT Byte Code) jest assemblerem dla NXT, stanowi on bazowy zestaw instrukcji obsługiwanych przez NXT Brick.



1.4 Co NXT oferuje ?

Porty wejścia/wyjścia

NXT Brick posiada 3 porty wyjściowe pozwalające na zasilanie i sterowanie aktorami oraz 4 porty wejściowe pozwalające na podłączenie, zasilenie i odczytywanie stanu sensorów.

Silniki

W skład NXT wchodzą sliniki sterowane za pomocą PWM wraz z wbudowanymi przekładniami oraz optycznymi enkoderami pozwalającymi na dokładne sterowanie ich położeniem. NBC pozwala na synchroniczne sprzęganie silników, teoretycznie z dokładnością do 1 stopnia. Praktyka pokazuje, że dokładność standardowego sprzężenia jest w znacznym stopniu uzależniona od aktualnego naładowania akumulatorów.

Czujnik dotyku

Pozwala dwustanowo stwierdzić czy został on naciśnięty, czy też nie. Działa na podobnej zasadzie jak zwykłe przyciski, np w klawiaturze.

Sensor światła

Mierzy natężenie światła padającego na jego czoło w zakresie od 0(całkowita ciemnośc) do 1023(bardzo jasno), ponadto posiada własne sterowane źródło światła, pozwalające mierzyć jasność podłoża poprzez badanie intensywności światła odbitego.

Sensor dźwięku

Pozwala mierzyć natężenie dźwięku w zakresie od 0 do 100%, przy czym 100% można uzyskać poprzez klaśnięcie w pobliżu czujnika.

Ultradźwiękowy czujnik odległości(sonar)

Umożliwia pomiar odległości w zakresie od 5 do 230 cm z teoretyczną dokładnością +/- 3cm, przy czym badania zrealizowane przez mojego kolegę
Karola Skrzymowskiego wykazały, że odchyłki potrafią być większe oraz występują nieliniowo.



2 Przygotowania i możliwe rozwiązania

2.1 Sposób sterowania

W przypadku LEGO NXT istnieją 2 sposoby sterowania robotem, które wstępnie były przeze mnie brane pod uwagę : lokalne i zdalne. W trakcie rozwoju projektu powstał pomysł na 3 rozwiązanie, a mianowanie sterowanie hybrydowe.

- Sterowanie lokalne.
  Poprzez sterowanie lokalne rozumiem, że całość logiki sterującej znajduje się w robocie i jest on całkowicie autonomiczny. Wszystkie obliczenia wykonywane są bezpośrednio na urządzeniu docelowym i nie występuje komunikacja z innymi urządzeniami.

- Sterowanie zdalne.
  Sterowanie zdalne polega na tym, że robot przesyła surowe dane do komputera PC, tam są one obrabiane i do robota wysyłane są jedynie konkretne rozkazy typu: jedź prosto, skręć. Tak więc przy sterowaniu zdalnym logika sterująca znajduje się na urządzeniu zewnętrznym, robot jedynie informuje o aktualnym stanie sensorów i wykonuje rozkazy.

- Sterowanie hybrydowe.
  Jego powstanie było wymuszone niedoskonałością obu powyższych metod. W tym modelu robot odpowiada za poruszanie się oraz orientację w labiryncie, natmiast wszelkie złożone algorytmy znajdują się na komputerze PC, do którego dostarczane są jedynie wstępnie obrobione dane, które są następnie ostatecznie przetwarzane a wyniki obliczeń są zwracane do robota.

2.2 Języki programowania z podziałem na lokalne i zdalne

W przypadku języków programowania istnieje ich znaczna różnorodność, do opisu wybrałem jedynie te darmowe i najbardziej powszechnie stosowane. Języki dedykowane do sterowania hybrydowego jako takie nie istnieją, sterowanie takie musi zostać zrealizowane niezależnie w 2 różnych językach(częściowo z wyjątkiem leJOS, gdzie oba programy pisane są w tym samym środowisku, jednak nadal są to 2 niezależne twory). Szersze zestawienie dostępnych języków można znaleźć na stronie zespołu tworzącego roboty klasy LEGO Sumo. [1]

- **Sterowanie lokalne**

  NXT-G - Jest to graficzne środowisko dostarczarne razem z zestawem LEGO Mindstorms. Cechuje je łatowość obsługi, jednak tworzenie rozbudowanych programów jest znacznie utrudnione ze względu na graficzny system programowania i ogromne rozmiary programów.

  leJOS NXJ - Jest to pakiet pozwalający tworzyć oprogramowanie dla LEGO w języku JAVA, do jego działania niezbędne jest wymienienie firmware znajdującego się na NXT Brick.

  NXC - Rozwijając skrót Not eXactly C, jest to kompilator języka C dla LEGO NXT. NXC nie obsługuje pełnej składni języka C ze względu na ograniczone możliwości samego NXT Brick. Oferuje jednak największe możliwości spośród darmowo dostępnych środowisk dla LEGO NXT.

- **Sterowanie zdalne**

  iCommand Jest to część pakietu leJOS udostępniająca komunikację poprzez Bluetooth, ze względu na znaczne rozbudowanie tej biblioteki jest ona stosunkowo wolna.

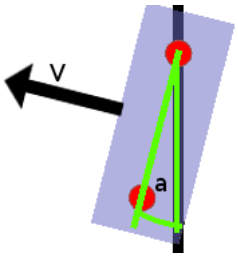
  Mindsqualls Jest to bilblioteka napisana w C# oparta na .NET Framework oferująca wyłącznie komunikację, tak więc niewielką część możliwości pakietu iCommand.

[1] `http://www.teamhassenplug.org/NXT/NXTSoftware.html`

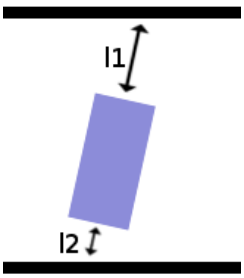
2.3 Sposób lokalizacji w labiryncie

LEGO NXT oferuje ograniczoną liczbę wejść sensorów tak więc ich dobór również nie był bez znaczenia.

- **Orientacja na podstawie 2 czujników linii** Do określania pozycji wykorzystywane są 2 czujniki lini oraz 1 lub 2 sonary. Wykorzystywana jest różnica w czasie pomiędzy przekroczeniem linii przez poszczególne czujniki. Na tej podstawie obliczany jest odchył robota od trasy wzorcowej. Na podstawie obliczonego kąta oraz odczytu z sonaru można skorygować pozycję robota. Wadą tego rozwiązania jest możliwość korekcji wyłącznie w momencie przekraczania lini. Ograniczeniem jest również dokładność samego pomiaru. Nie dysponując przerwaniami procesora nie jesteśmy w stanie uchwycić dokładnie momentu wjazdu na linię, w efekcie odczyty są zafałszowane(opóźnione). Ilustracja poniżej obrazuje podstawę do obliczenia kąta odchyłu.



- **Orientacja na podstawie 2 sonarów** W tym przypadku do pozycjonowania się w labiryncie wykorzystywane są 2 sonary. Robot próbuje pozycjonować się jak najbliżej środka przestrzeni pomiędzy bocznymi ścianami. Ten sposób pozycjonowania pozwala na ciągła korekcję toru jazdy, co nawet przy niedokładności pomiarów pozwala na dość dokładną korekcję. Różnica odległości od ścian definuje przesunięcie robota, natomiast tempo narastania różnicy pomiędzy tymi odległościami pozwala na wywnioskowanie kąta odchylenia od optymalnego toru jazdy.



2.4 Wybór rozwiązania

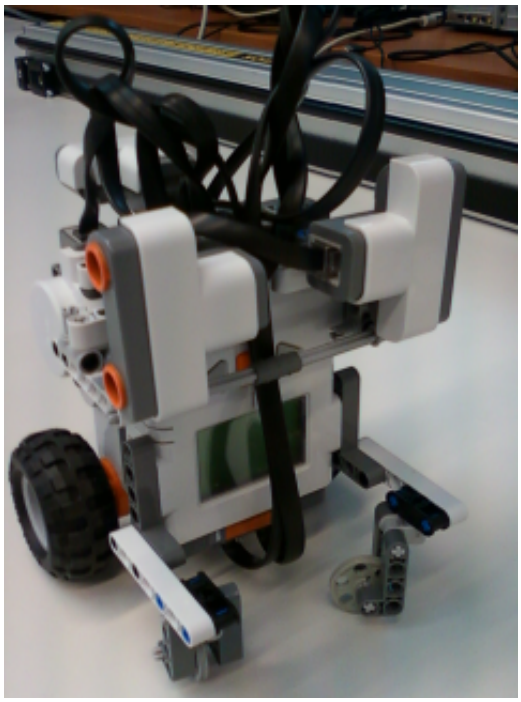
Zdecydowałem, że w labiryncie będę orientował się z wykorzystaniem 3 sonarów i 1 czujnika lini. Sposób ten jest znacznie bliższy ludzkiego postrzegania, a jednocześnie daje więcej możliwości poźniejszejszego poszerzania możliwości robota, ponieważ dostarcza znacznie więcej informacji o otoczeniu.
W przypadku sposobu sterowania następowała stopniowa ewolucja. Początkowo sterowanie robotem realizowane było wyłącznie lokalnie, jednak kontrola aktualnych parametrów robota była znacznie utrudniona i uciążliwa, co znacznie spowalniało pracę nad rozwojem projektu. Skłoniło mnie to do zastosowania sterowania zdalnego, które pozwalało na całkowitą kontrolę nad zachowaniem robota. Rozwiązanie to początkowo bardzo dobre, wraz z rozwojem projektu również stało się niewystarczające. Problemem w tym przypadku była szybkość komunikacji. Protokół LEGO oparty był o wiele krótkich telegramów, co sprawiało, że narzut na prawidłowe ich zaadresowanie i dekodowanie był znaczny, co istotnie zaniżało faktyczną przepustowość łącza. Efektem powyższych jest rozwiązanie hybrydowe w którym kontrola behawioralna robota jest realizowana bezpośrednio na nim - kontrola toru jazdy, obrotów, skrętów oraz mapowanie labiryntu na podstawie odczytów z sonarów. Komputer natomiast stanowi interface człowiek-maszyna, wykonuje bardziej złożone obliczenia i przesyła do robota ich wyniki. Komunikacja nadal odbywa się poprzez Bluetooth, jednak wykorzystałem opracowany przeze mnie protokół komunikacyjny. W ramach 1 pakietu wysyłana jest znaczna część jednoznacznie zakodowanych informacji posiadanych przez robota. Specjalizacja protokołu znacznie przyśpieszyła transmisję.
Wybór języka programowania nie stanowił większego problemu, dla LEGO wybrany został NXC ze względu na jego wydajność (najszybszy z dostępnych darmowo), dla PC została wybrana biblioteka Mindsqualls napisana w C# z podobnego względu, leJOS jest zbyt rozbudowany i przez to znacznie wolniejszy od specjalizowanego Mindsqualls. Dodatkowym czynnikiem był fakt, że są to języki ze wspólnej rodziny bazujące na C.
Jako środowisko programistyczne przy pisaniu oprogramowania dla robota posłużyły mi Bricx Command Center oraz LEMAS (autorstwa dr. Macieja Sławińskiego), oba dedykowane dla platformy LEGO NXT i języka NXC. Oprogramowanie dla komputera PC napisane zostało przy użyciu Microsoft Visual Studio 2008.

3 Realizacja

Początek projektu stanowiło zapoznanie się dokumentacją oraz możliwościami oferowanymi przez platforme LEGO NXT. Środowiska programistyczne i języki programowania dla tej platformy również były mi obce, więc im również należało poświęcic uwagę. Nie stanowiło to jednak bezpośredniego celu projektu, tak więc nie będę się na ten temat rozpisywał.

3.1 Budowa robota

Pierwszy realny etap stanowiło zbudowanie robota. Podstawowym założeniem było, aby rozmiary podstawy robota były jak najmniejsze, co pozwoliłoby na swobodne poruszanie się po korytarzach labiryntu. Efektem był robot o znacznej wysokości, posiadający zarówno silniki jak i LEGO Brick zamontowane pionowo. Znacznie podnosiło to środek ciężkości całej konstrukcji, jednocześnie umiejscawiając go na środku robota. Powodowało to problemy ze stabilnością przy gwałtownych manewrach zmiany prękości. Rozwiązaniem tego problemu było jednoznaczne przesunięcie środka ciężkości ku jednemu z końców oraz zastosowanie dodatkowego zestawu kół. Pierwsza wersją mająca realne szanse, by stać się finalną a jednocześnie posiadająca swe zdjęcia przedstawiała się następująco.



W późniejszym okresie okazało się, że minimalny zasięg sonarów wynosi ponad 5 cm, co przy 15 cm szerokości całego robota oraz założeniu, że ma on swobodnie poruszać się po labiryncie szerokości 30 cm stanowiło dość poważny problem. Skutkiem tego było odsunięcie sonarów od krawędzi robota w celu poprawienia jego minimalnego zasięgu. Po tym zabiegu robot przedstawia się następująco i nie był poddawany dalszym modyfikacjom.



3.2 Implementacja

Początkowo program został zrealizowany na kilku jednocześnie działających wątkach przy założeniu całkowitej autonomiczności robota. Każdy z nich testowany osobo sprawdzał się nadzyczaj dobrze w swoim zadaniu. Jednak w momencie, gdy wszystkie wątki zostały uruchomione jedocześnie działanie robota stało się bardzo niestabilne. Wynikało to z mało wydajnego mechanizmu wielowątkowości zaimplementowanego w NXT. Ponadto obliczenia związane z wyznaczaniem najkrótszej drogi okazały się zbyt złożone i wbrew moim oczekiwaniom odnośnie robota(powinien działać w czasie rzeczywistym, bez wyraźnych przestojów) trwały znacznie zbyt długo.
Kolejne rozwiązanie zakładało sterowanie zdalne z przeniesieniem całości logiki na komputer PC. Wynikiem było poprawne choć również wolne działanie robota, wynikające z małej przepustowości łącza Bluetooth oraz architektury telegramów standardowo oferowanych przez NXT.
Program i projekt został po raz kolejny przeprojektowany. Obecnie składa sie on z 2 części : kodu odpowiedzialnego za sterowanie behawioralne robotem znajdujące się bezpośrednio na robocie, oraz programu dla komputera PC który pozwala śledzić aktualne parametry robota oraz odpowiada za obliczanie drogi pomiędzy dwoma punktami zorientowanymi na mapie.
Starałem się maksymalnie zredukować liczbę równoległych wątków i obecnie program działający na NXT jest strukturą niemal całkowicie proceduralną(jednocześnie działają 2 wątki), wraz z wykorzystaniem skoncentrowanej komunikacji z komputerem PC, na którym, w celu przyśpieszenia działania(oraz większej kontroli nad robotem) wykonywane są bardziej złożone obliczenia odpowiedzialne za kroki podejmowane przez robota.

3.2.1 Kod dla robota

Nie udało mi się całkowicie uproceduralnić kodu robota toteż występują w nim 2 równolegle działające wątki, nie stanowi to jednak problemu i program działa płynnie

Wątek main to de facto serce programu, zawiera główną pętlę w której sekwencyjnie wykonywane są mniejsze części programu. Przed pętlą inicjowane są parametry robota oraz uruchamiany jest wątek LineScanning. W pętli sekwencyjnie sprawdzana jest aktualna pozycja względem pozycji docelowej, następnie analizowane są odczyty z sonarów, wykonywana jest komunikacja z PC i jeżeli komputer odpowiedział pozycją do której robot ma się udać wykonywany jest krok.

Wątek Line Scanning odpowiada wyłącznie za analizę koloru podłoża oraz aktualizowanie na tej podstawie pozycji robota. Konieczność zastosowania w tym celu osobnego wątku wynikała z zastosowania standardowych procedur synchronizacji silnika, które blokują wątek w których zostały uruchomione. Nie oddzielenie zadania skanowania podłoża do osobnego wątku skutkowałoby zatrzymaniem skanowania w trakcie ruchu robota, czyli dokładnie w momencie, gdy skanowanie ma szanse wykryć zmianę pozycji.

Funkcje wykorzystywane w wątku main

MinorCommunication - funkcja odpowiadająca za wymianę informacji mniej istotych, wysyłanych jedynie w celach kontroli parametrów przez użytkownika, nie stanowi ona o wynikach działania programu

MessageExchange - funkcja wymienia z komputerem informacje na temat mapy, oraz innych informacji niezbędnych do obliczenia następnego kroku robota oraz odbiera wyniki obliczeń komputera (następny krok).

MakeStep - funkcja odpowiedzialna za jazdę, obecnie bez korekcji pozycji robota względem ścian.

TurnFunction - funkcja realizująca prostopadłe skręty na podstawie informacji o aktualnej pozycji i orientacji oraz pozycji docelowej następnego kroku.

AnalyzeSonars, UpdateMap - funkcje odpowiedzialne za analizę sonarów oraz tworzenie mapy labiryntu. Sprawdzają w jakiej odległości znajduje się przeszkoda i odnotowują ten fakt na mapie labiryntu.

3.2.2 Kod PC

Kod dla komputera PC składa się z 3 klas podstawowych : MazeSolver, Map oraz MazeSolverView.

Klasa MazeSolver stanowi serce programu dla komputera PC. Wykorzystuje ona funkcje biblioteki Mindsqualls. Głównym składnikiem tej klasy jest wątek utworzony z metody GetMessage. Odpowiada on za utrzymanie komunikacji z robotem, wymianę telegramów, buforowanie aktualnych danych na jego temat oraz odpowiadanie na zapytania robota. Telegramy w tej metodzie są odbierane, następnie dekodowane a ich wyniki zapisywane są do zmiennych udostępnianych wyższym warstwom programu, jeżeli telegram zawierał prośbę o następny krok do wykonania, krok ten jest obliczany i wysyłany w odpowiedzi. Całość komunikacji odbywa się poprzez system mailboxów oferowany przez bilbiotekę Mindsqualls.
Klasa Map, podrzędna względem MazeSolver, odpowiada za przechowywanie akutualnej mapy posiadanej przez robota oraz wyliczanie najkrotszej trasy pomiedzy 2 punktami znajdujacymi się na tej mapie. Klasa Map udostępnia całość posiadanych informacji do warstw wyższych co pozwala m.in. na wyświetlenie mapy. Kroki, które mają zostać wykonane przez robota obliczane są w wewnętrznej metodzie klasy Map, ShortestStep. Jest ona oparta o algorytm wyszukiwania najkrótszej drogi pomiędzy dwoma punktami w grafie acyklicznym (Algorytm A*). Informacje na temat wykorzystanego algorytmu są ogólnie dostępne, toteż nie będę go tu opisywał, a jedynie podam odnośnik do strony zawierającej dokładniejsze informacje.[2]

[2] `http://pl.wikipedia.org/wiki/Algorytm_A*`

3.3 Protokół komunikacji

Komunikacja z NXT korzysta z modelu klient-serwer, gdzie w przypadku protokołu LEGO rolę serwera najczęściej pełni NXT Brick, a klientem jest urządzenie zewnętrzne, w przypadku protokołu zrealizowanego przeze mnie role urządzeń są odwrócone.
Standardowy protokół komunikacji LEGO pozwala na zdalną kontrolę nad wszystkimi parametrami robota. Oparty jest on o krótkie telegramy zawierające informacje na jeden konkretny temat, często jednak zawierające dane nadmiarowe. W efekcie informacja użyteczna stanowi niewielką część komunikatów przesyłanych pomiędzy urządzeniami.
Protokół napisany przeze mnie, opiera się na przesyle skoncentrowanych wiadmości zawierających jedynie informacje użyteczną dla komputera PC. Z danych mających zostać przesłane formowana jest pojedyncza wiadomość(do wysłania całej wiadomości wystarczy 1 nagłówek). Format komunikatu oraz jego treści są zdefiniowane, w efekcie przesyłanie opisu zawartości telegramu stało się zbędne i pozwoliło zmiejszyć rozmiar przesyłanych danych

Porównanie komunikacji na przykładzie funkcji MinorCommunication
Funkcja ta przesyła do komputera odczyty z 3 sonarów, 1 czujnika natężenia światła oraz dane na temat aktualnej pozycji i orientacji względem labiryntu.
Protokół LEGO
Każdy sensor należy odpytać indywidualnie, toteż należy wykonać 4 zapytania w celu uzyskania danych na temat czujników, ponadto trzeba wykonać zapytanie o aktualną pozycję i orientację.
Odczyt czujnika składa się z 3 bajtów zapytania : 2 bajty nagłowka i 1 bajt zawierający numer portu. Odpowiedź stanowi pakiet 16 bajtów : 3 bajty nagłówka oraz 13 bajtów danych na temat sensora, przy czym informację użyteczna stanowią 2 bajty (10 i 11) zawierające znormalizowany odczyt z sensora.
Przesłanie pozycji i orientacji musiałoby zostać zrealizowane jako zapytanie o długości 5 bajtów, a odpowiedź miałaby długość 7 bajtów.
Całkowita ilość przesłanych danych wynosiłaby 88 bajtów.

Protokół własny

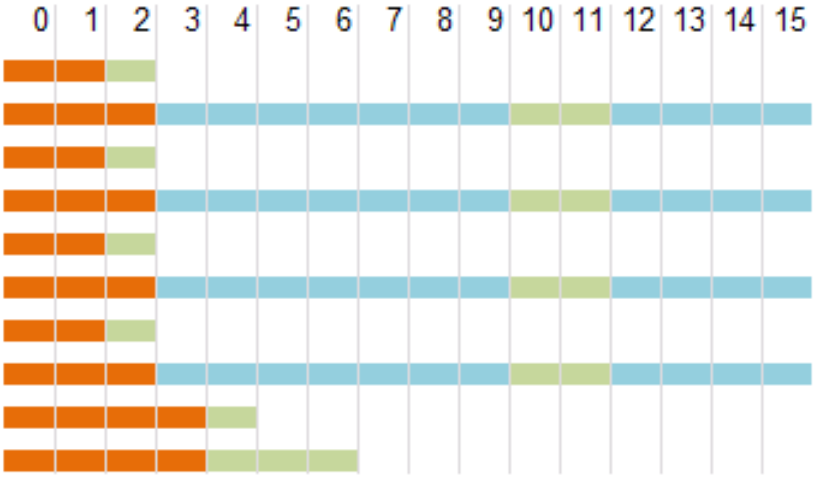
Komunikat wysyłany do komputera składa się z : 4 bajtów nagłówka, 4 bajtów danych z sensorów, 2 bajtów na temat aktualnej pozycji oraz 1 bajtu informacji o aktualnej orientacji robota. Odpowiedź komputera stanowią 3 bajty : 2 nagłówka oraz 1 informujący czy dane zostały odebrane.

Daje to sumaryczną ilość 14 bajtów wymienionych pomiędzy urządzeniami.

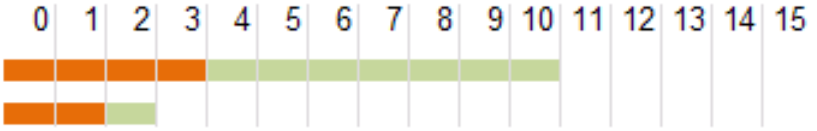
Porównanie

Z powyższego zestawienia wynika, że protokól LEGO do wysłania tej samej informacji użytecznej wymaga przesłania ponad 6 razy więcej danych niż specjalizowany protokół mojego autorstwa.

Ponadto komunikacja protokołem LEGO wymaga wysłania 5 komunikatów podczas, gdy w przypadku protokołu autorskiego wystarcza 1 wiadomość.



Schemat przesyłanych danych przy użyciu protokołu LEGO.
Kolory oznaczają typ danych
czerwony - nagłówek, niebieski - informacja nadmiarowa
zielony - informacja użyteczna



Schemat przesyłanych danych przy użyciu protokołu autorskiego.
Kolory oznaczają typ danych
czerwony - nagłówek, zielony - informacja użyteczna

3.4 Opis uruchomieniowy

Zewnętrzne biblioteki (kod dla PC)

Aby wykorzystywać zewnętrzne biblioteki należy je dołączyć do projektu. By tego dokonać, należy dodać referencje do bibliotek. W MS Visual Studio wykonuje się to następująco. W drzewie projektu wyszukujemy gałąź References, następnie z menu kontektstowego wybieramy Add Reference, gdzie w zakładce Browse wskazujemy plik .dll zawierający porządaną bibliotekę.

W przypadku programu dla PC wykorzystywane są 2 zewnętrzne biblioteki:

MindSqualls - odpowiedzialna za komunikację z robotem

SpPerfChart - wykorzystywana do rysowania diagramów

Dodawanie kontrolek do toolbox'a

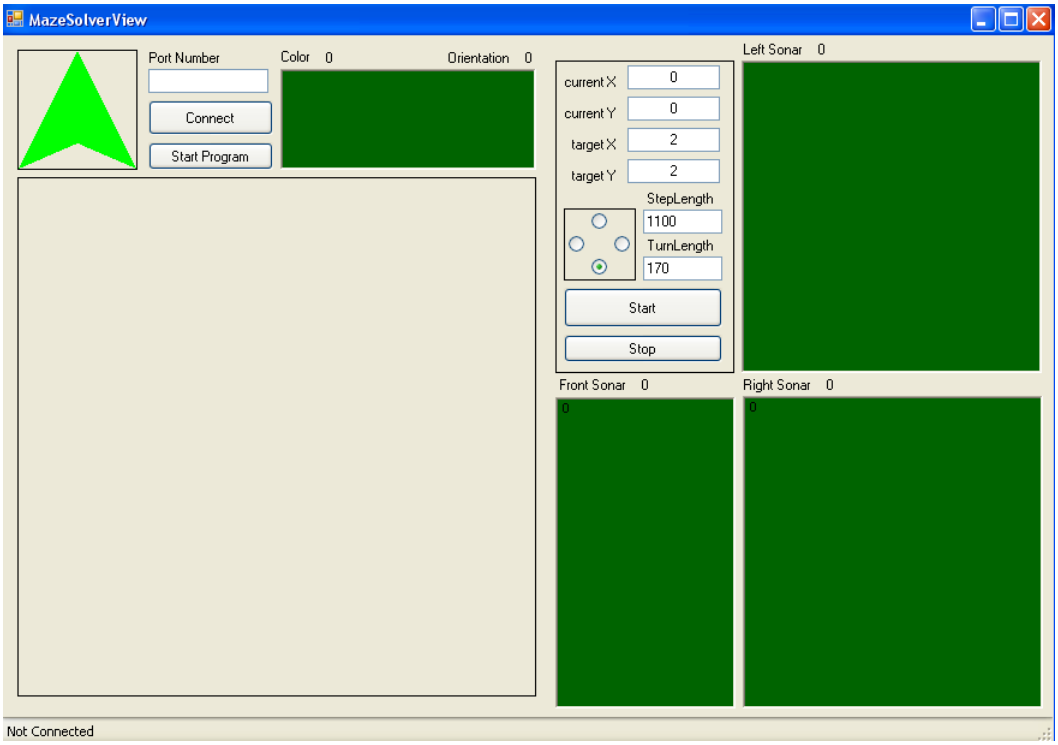
Aby dodać SpPerfChart do formatki programu należy najpierw dodać go do toolbox'a skąd może zostać przeniesiony do okna programu. W tym celu należy, kliknąć prawym klawiszem na toolbox i z menu kontekstowego wybrać Choose Items, w zakładce .NET Framework Components kliknąć Browse i wskazać ścieżkę do kontrolki.

Kompilowanie i wgrywanie programu dla robota

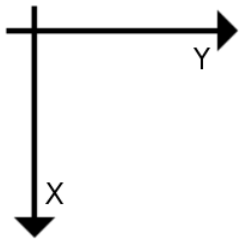
BricxCC nie oferuje struktury projektów toteż, by skompilować kod wystarczy utworzyć nowy plik, wprowadzić kod oraz klikając na Compile skompilować go. Aby wgrać program do NXT Bricka należy połączyć się z NXT korzystając z opcji Find Brick, następnie skompilować program i kliknąć Download, program zostanie przesłany do NXT, a pomyślne zakończenie zostanie obwieszczone piknięciem NXT Bricka.

Uruchamianie programu MazeSolver

W chwili obecnej kolejność uruchamiania programów jest bez znaczenia. Istotne jest jedynie, by przed próbą połączenia się z robotem został on włączony. Nie jest konieczne uruchamianie programu na robocie, może on zostać uruchomiony zdalnie i ta metoda jest zalecana ze względu na wygodę. Połączenie przez Bluetooth jest symulowane jako połączenie przez port COM, toteż na formatce programu znajduje się pole Port Number gdzie należy wprowadzić numer portu na które został zamapowany modem Bluetooth, jeżeli numer ten jest nieznany należy pole zostawić puste i kliknąć Connect. Program spróbuje wyszukać połączenie Bluetooth na wszystkich dostępnych portach, jest to rozwiązanie wolne toteż nie jest zalecane. Następnie należy podać aktualną i docelową pozycję robota, wybrać jego aktualną orientację względem labiryntu oraz kliknąć Start.

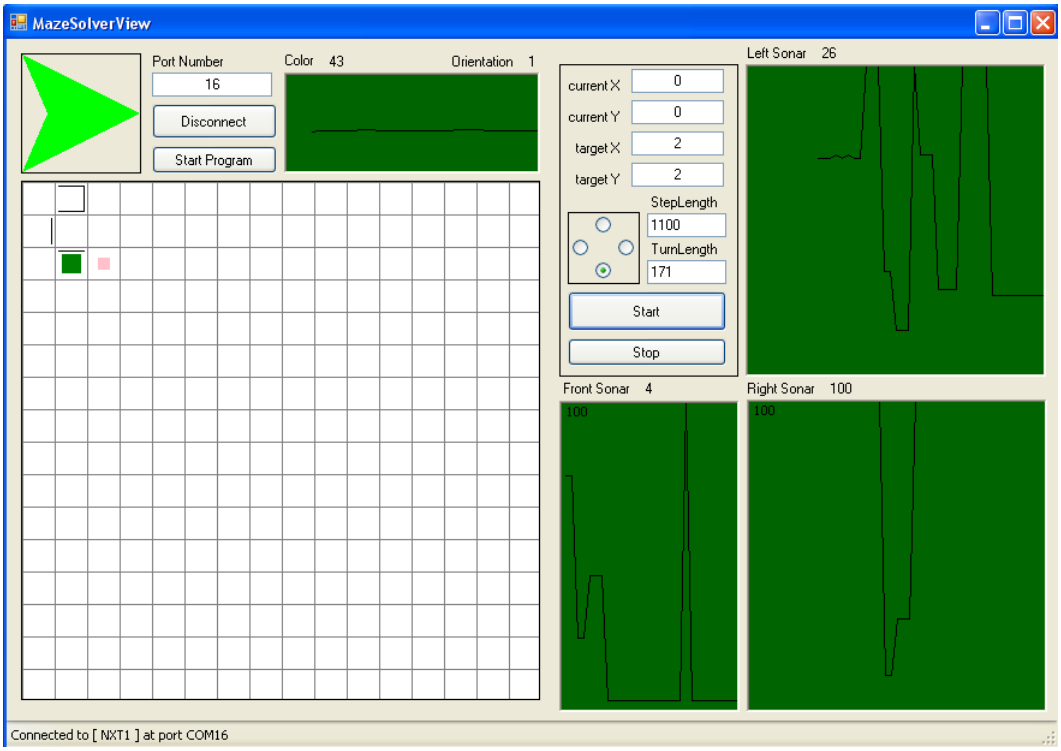


Wygląd programu dla PC po uruchomieniu

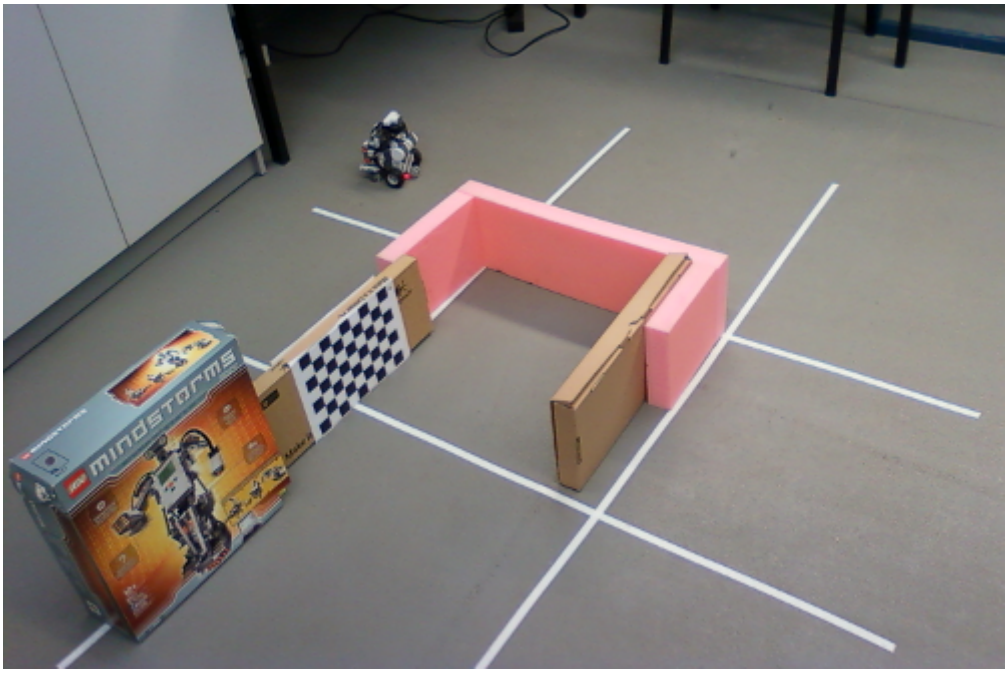


Zorientowanie układu współrzędnych

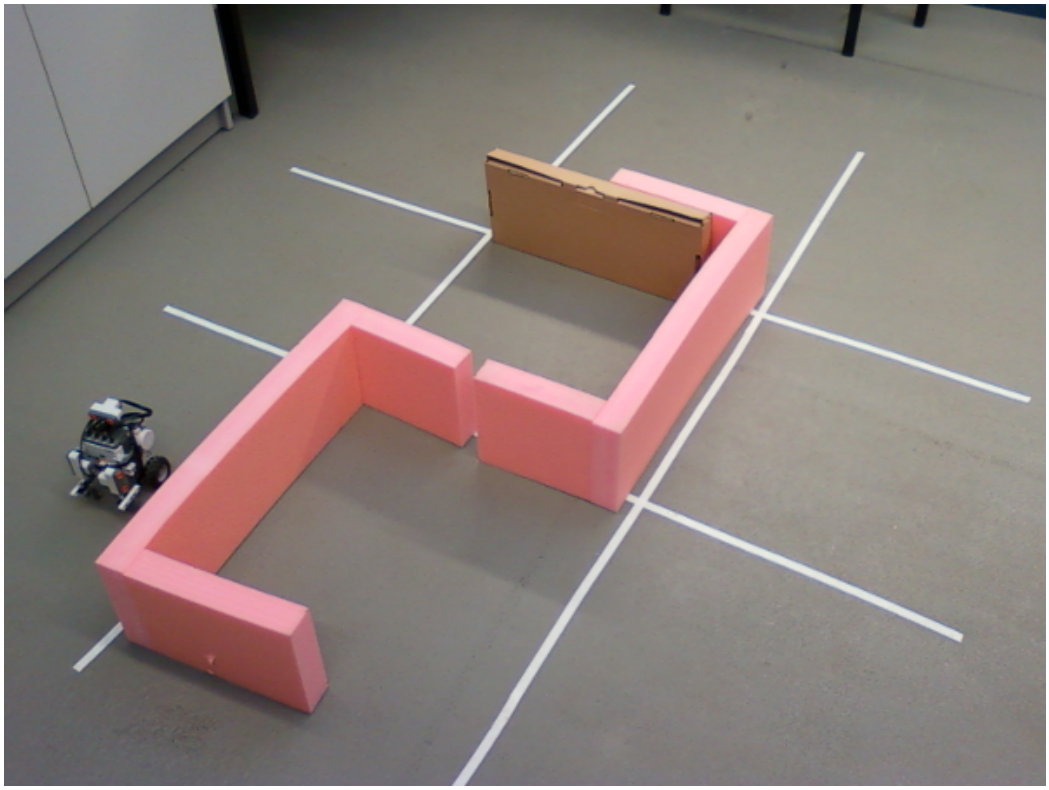
Aktualną pozycję wpisuję się w pola currentX oraz currentY (aktualne współrzędne), natomiast współrzędne docelowe podaje się w polach targetX i currentY, orientację robota względem labiryntu wybiera się na panelu 4 przycisków poniżej.
Pola StepLength i TurnLength służą do kalibracji długości pól labiryntu oraz wymaganego kąta obrotu kół, by robot wykonał obrót o 90 stopni.
Panel po lewej stronie ukazuje aktualny stan mapy którą posiada robot. Kompas natomiast jego aktualną orientację względem labiryntu.

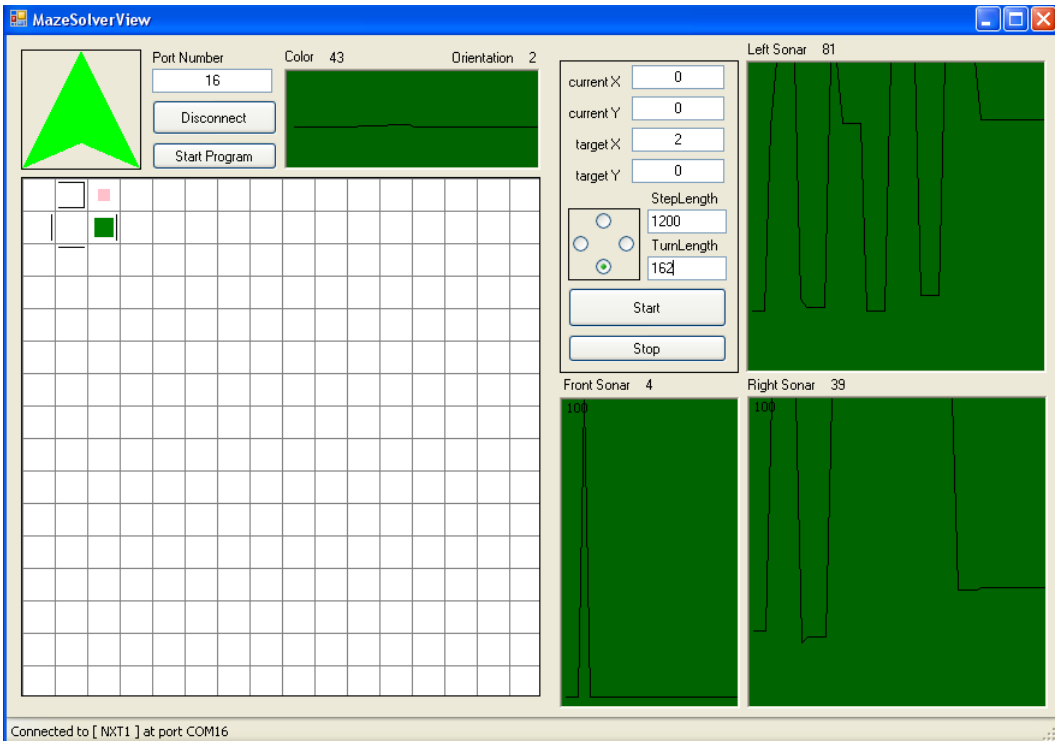


Wygląd programu w trakcie działania robota.



Robot w trakcie pokonywania przykładowego labiryntu.





Przykład labiryntu i wygląd utworzonej przez robota mapy po jego przejechaniu.

4 Podsumowanie, wnioski i możliwości rozwoju

Projekt dotyczył budowy i oprogramowania robota opartego o platformę LEGO NXT. Moim celem natomiast było stworzenie robota potrafiącego orientować się w labiryncie, znajdować w nim trasę pomiędzy dwoma zdefiniowanymi punktami oraz poruszać się po niej.
Projekt napisany w dwóch językach z rodziny C tj. C# oraz NXC, realizuje wstępne założenia, trudności sprawia jedynie poruszanie się po labiryncie. Poprawnie skalibrowany robot bez trudu pokonuje labirynty. Jednak brak algorytmu korekcji pozycji sprawia, że w przypadku błędów kalibracji robot nie jest w stanie utrzymyać pozycji centralnie pomiędzy ścianami co ostatecznie prowadzi do zboczenia z trasy i uderzenia w przeszkodę lub błędnego zamapowania labiryntu i nie dotarcia do celu.
Problemy z poruszaniem się po wyliczonej trasie wynikają z kilku czynników. Fakt, że robot budowany był z myślą o minimalizacji jego wymiarów sprawił, iż stabilność konstrukcji ucierpiała, co skutkuje w przekrzywianiu osi podczas jazdy i skrętów, a w efekcie odchylanie się od założonej trasy przejazdu. Kolejnym czynnikiem powodującym zbaczanie z trasy jest niedokładność synchronizacji silników oraz niedokładność enkoderów zainstalowanych w silnikach. Oba te czynniki sprawiają, że przed uruchomieniem robota należy skalibrować, doświadczalnie dobierając parametry jego jazdy dostosowując je do podłoża i aktualnego stanu naładowania baterii.
Wystąpienie problemów w realizacji sterowania sprawiło, że dokładnie zapoznałem się z 3 różnymi sposobami jego realizacji, co ostateczie również można zaliczyć do sukcesów projektu.
Platforma LEGO okazała się dośc przyjazna w użytkowaniu. Jednak możliwość budowy w oparciu o nią złożonych projektów jest ograniczona, zarówno przez moc obliczeniową, która w przypadku programów wielowątkowych znacznie spada, jak i małej dokładności sensorów oferowanych dla tej platformy.
Jako kierunek rozwoju wskazałbym poprawę synchronizacji silników(napisanie własnej funkcji za to odpowiedzialnej), scalenie funkcjonalności obu wątków w jeden oraz napisanie procedury korekcji pozycji względem ścian labiryntu.

Bibliografia

1. http://mindstorms.lego.com/en-us/support/files/default.aspx
2. http://pl.wikipedia.org/wiki/Algorytm_A*
3. http://bricxcc.sourceforge.net/
4. http://www.teamhassenplug.org/NXT/NXTSoftware.html

5 Kod źródłowy

5.1 mazesolv.nxc

```
#define Color IN_4
#define Sonar_F IN_1
#define Sonar_L IN_2
#define Sonar_R IN_3

#define Motor_Left OUT_A
#define Motor_Right OUT_C
#define Motor_Both OUT_AC

#define Dir_Yp 0
#define Dir_Xp 1
#define Dir_Ym 2
#define Dir_Xm 3

#define Dist_0 40
#define Dist_1 50
#define Dist_2 80

#define MESS_STEP_INFO 0x01
#define MESS_START 0x02
#define MESS_STOP 0x04
#define MESS_BLANK 0x00

int CellSize=30;
int StepLength=1000;
int TurnLength=165;

byte Map[256];              /* Bit data 4-ClosedSetMarker 3-0 = WallInfo [ 0 -
    Probably passable, 1 - Impassable ][X-,Y-,X+,Y+]*/

byte CurrentPosition=0;
byte TargetPosition=255;
byte Orientation;
byte NextStep;

long LastRotoLeft,LastRotoRight;

long LineRotoLeft,LineRotoRight;
int Turn=1;

unsigned int OnLine=0;

/* Controll Flags */

byte SolveFlag;


/*******************************************************/
byte MergePoint(byte x, byte y)
```

```
{
  return (x%16)*16+(y%16);
}
byte SplitX(byte p)
{
  return p/16;
}
byte SplitY(byte p)
{
  return p%16;
}

void Success(void)
{
  PlayTone(740,100);
  Wait(100);
  PlayTone(940,100);
  Wait(100);
  PlayTone(1140,100);
  Wait(100);
  PlayTone(940,100);
  Wait(100);
  PlayTone(1140,100);
  Wait(100);
  StopAllTasks();
}

void Fail(void)
{
  PlayTone(440,1000);
  Wait(500);
  StopAllTasks();
}

void UpdateMap(byte Dist, byte Dir)
{
  byte X = SplitX(CurrentPosition);
  byte Y = SplitY(CurrentPosition);

  if(Dir==Dir_Yp)
  {
    if(Y+Dist <= 15)
    {
      Map[MergePoint(X,Y+Dist)] |= 0x01 ;
    }
  }
  else if(Dir==Dir_Ym)
  {
    if(Y-Dist >= 0)
    {
      Map[MergePoint(X,Y-Dist)] |= 0x04;
    }
  }
  else if(Dir==Dir_Xm)
```

```
  {
    if(X-Dist >= 0)
    {
      Map[MergePoint(X-Dist,Y)] |= 0x08;
    }
  }
  else if(Dir==Dir_Xp)
  {
    if(X+Dist<=15)
    {
      Map[MergePoint(X+Dist,Y)] |= 0x02;
    }
  }
}

void AnalyzeSonars()
{
  byte D=Orientation;

  /* Front */
  if((SensorUS(Sonar_F)+5)<Dist_0)UpdateMap(0,(D)%4);
//  else if(SensorUS(Sonar_F)<Dist_1)UpdateMap(1,(D)%4);
//  else if(SensorUS(Sonar_F)<Dist_2)UpdateMap(2,(D)%4);

  /* Right */
  if(SensorUS(Sonar_R)<Dist_0)UpdateMap(0,(D+3)%4);
//  else if(SensorUS(Sonar_R)<Dist_1)UpdateMap(1,(D+1)%4);
//  else if(SensorUS(Sonar_R)<Dist_2)UpdateMap(2,(D+1)%4);

  /* Left */
  if(SensorUS(Sonar_L)<Dist_0)UpdateMap(0,(D+1)%4);
//  else if(SensorUS(Sonar_L)<Dist_1)UpdateMap(1,(D+3)%4);
//  else if(SensorUS(Sonar_L)<Dist_2)UpdateMap(2,(D+3)%4);

}

void Init()
{
  SetSensorLowspeed(Sonar_F); /* Sonar*/
  SetSensorLowspeed(Sonar_L); /* Sonar*/
  SetSensorLowspeed(Sonar_R); /* Sonar*/
  SetSensorLight(Color); /* Line Detector*/
  LastRotoLeft=MotorRotationCount(Motor_Left);
  LastRotoRight=MotorRotationCount(Motor_Right);
};

void Turn_Function()
{
 byte TurnAngle;
  if(SplitX(NextStep)>SplitX(CurrentPosition))
  {
    /* X+ */
    TurnAngle=1;
  }
```

```
  else if(SplitX(NextStep)<SplitX(CurrentPosition))
  {
    /* X- */
    TurnAngle=3;
  }
  else if(SplitY(NextStep)>SplitY(CurrentPosition))
  {
    /* Y+ */
    TurnAngle=0;
  }
  else if(SplitY(NextStep)<SplitY(CurrentPosition))
  {
    /* Y- */
    TurnAngle=2;
  }
  else
  {
  //StopAllTasks();
  }
  Turn = ( (4 + TurnAngle-Orientation) %4 );
    NumOut(0,LCD_LINE4, Orientation);
    NumOut(0,LCD_LINE5, TurnAngle);
  NumOut(0,LCD_LINE6, Turn);
  Orientation = TurnAngle;


  if(Turn==3)
  {
    RotateMotorEx(Motor_Both,50,TurnLength,100,true,true);
  }else if(Turn==2)
  {
   RotateMotorEx(Motor_Both,50,TurnLength*2,-100,true,true);
  }else if(Turn==1)
  {
    RotateMotorEx(Motor_Both,50,TurnLength,-100,true,true);
  }else
  {
  }

}

void MakeStep(void)
{

  if(CurrentPosition != NextStep)RotateMotorEx(Motor_Both,40,-StepLength,0,true,
      true);
}

byte MessageExchange()
{
  byte MessageBuffer[58];
  MessageBuffer[0]=SplitX(CurrentPosition);
  MessageBuffer[1]=SplitY(CurrentPosition);
  MessageBuffer[2]=SplitX(TargetPosition);
```

```
  MessageBuffer[3]=SplitY(TargetPosition);

  for(byte x=0;x<7;x++)
  {
    for(byte y=0;y<7;y++)
    {
      byte ax = SplitX(CurrentPosition) - 3 + x;
      byte ay = SplitY(CurrentPosition) - 3 + y;
      if (ax >= 0 && ax < 16 && ay >= 0 && ay < 16)
      {
        MessageBuffer[ 4 + 7 * x + y ] = Map[ MergePoint( ax, ay ) ];
      }
    }
  }

  if( SolveFlag == 1 ) MessageBuffer[54] = 'R'; /* Request for next step */
  else MessageBuffer[54] = 'U'; /* Just info update */
  MessageBuffer[55] = Orientation;
  MessageBuffer[57] = 0x54;              /* End of Data Marker */
  SendMessage(1,MessageBuffer);


  ReceiveMessage(2,true,MessageBuffer);
  if(MessageBuffer[0] == 'X'){
    Fail();
  }else if(MessageBuffer[0] == 'B')
  {
    CurrentPosition = MergePoint(MessageBuffer[1],MessageBuffer[2]);
    TargetPosition = MergePoint(MessageBuffer[3],MessageBuffer[4]);
    Orientation = MessageBuffer[5];
    StepLength = MessageBuffer[6]*100 + MessageBuffer[7];
    TurnLength = MessageBuffer[8]*100 + MessageBuffer[9];
    for(int i=0;i<256;i++)Map[i]=0;
    return MESS_START;
  }else if(MessageBuffer[0] == 'N')
  {
    NextStep = MergePoint(MessageBuffer[1],MessageBuffer[2]);
    return MESS_STEP_INFO;
  }
  return MESS_BLANK;
}

void MinorCommunication()
{
  byte Buffer[58];
  byte temp;
  {
    NumOut(0,LCD_LINE3,CurrentTick());

    temp = SensorUS(Sonar_F)+5;
    Buffer[0]=(temp < 101 ? temp : 100);
    temp = SensorUS(Sonar_L);
    Buffer[1]=(temp < 101 ? temp : 100);
    temp = SensorUS(Sonar_R);
```

```
    Buffer[2]=(temp < 101 ? temp : 100);

    Buffer[3]=SplitX(CurrentPosition);
    Buffer[4]=SplitY(CurrentPosition);

    Buffer[5] = Sensor(Color);
    Buffer[6] = Orientation;
    Buffer[7] = 55; /* End of data marker */

    SendMessage(3,Buffer);
  }
}

task LineScanning()
{
  while(1)
  {
    NumOut(0,LCD_LINE2,CurrentTick());
    if(Sensor(Color)>50)OnLine++;
    else if(OnLine>0){
      if(Orientation==Dir_Yp)
      {
        CurrentPosition = MergePoint(SplitX(CurrentPosition),SplitY(CurrentPosition
            )+1);
      }
      else if(Orientation==Dir_Ym)
      {
        CurrentPosition = MergePoint(SplitX(CurrentPosition),SplitY(CurrentPosition
            )-1);
      }
      else if(Orientation==Dir_Xm)
      {
        CurrentPosition = MergePoint(SplitX(CurrentPosition)-1,SplitY(
            CurrentPosition));
      }
      else
      {
        CurrentPosition = MergePoint(SplitX(CurrentPosition)+1,SplitY(
            CurrentPosition));
      }
      OnLine=0;
      if(Turn==0)
      {
       /* StepLength correction */
        //StepLength=(StepLength+((abs(LineRotoLeft-MotorRotationCount(Motor_Left))
            +abs(LineRotoRight-MotorRotationCount(Motor_Right)))/2))/2;
      }
      PlayTone(1500,20);
      }
    Wait(50);
  }
}

task main()
```

```
{
  Init();
  start LineScanning;

  byte MessageType;

  while(true)
  {
    NumOut(0,LCD_LINE1,CurrentTick());
    if( CurrentPosition == TargetPosition )
    {
      Success();
    }
    AnalyzeSonars();
    MinorCommunication();
    MessageType = MessageExchange();
    NumOut(0,LCD_LINE8,MessageType);
    if( SolveFlag == 1 && MessageType == MESS_STEP_INFO )
    {
      PlayTone(700,50);
      Turn_Function();
      MinorCommunication();
      MakeStep();
    }
    if( MessageType == MESS_STOP )
    {
      Fail();
    }
    else if ( MessageType == MESS_START )
    {
      SolveFlag=1;
      PlayTone(700,100);
    }
  }
}
```

5.2 MazeSolver.cs

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
using NKH.MindSqualls;
using System.IO;
using LegoNXTMazeSolver;
using System.Threading;
using System.Drawing;

namespace LegoNXTMazeSolver
{
    public class MazeSolver
    {
        private NxtBrick brick;
        public int sonarFront = 0;
        public int sonarLeft = 0;
        public int sonarRight = 0;
        public int color = 0;

        public Thread communicationThread;

        public Map map;
        public int orientation;
        public Point currentPosition;
        public Point targetPosition;

        public bool isConnected
        {
            get
            {
                if (brick == null) return false;
                return brick.IsConnected;
            }
        }

        public String name { get { return brick.Name; } }


        public MazeSolver(byte portNo)
        {
            brick = new NxtBrick(portNo);

            brick.Connect();

            map = new Map();

            currentPosition = new Point();
            targetPosition = new Point();

            communicationThread = new Thread(new ThreadStart(GetMessage));
            communicationThread.Start();
```

```
        StartProgram();
    }


    public void Dispose()
    {
        communicationThread.Abort();
        if (brick != null) brick.Disconnect();
        brick = null;
    }

    public void StartExploration(Point curr, Point targ, int orient, int
        stepLength, int turnLength)
    {
        StartProgram();
        brick.CommLink.MessageWrite(NxtMailbox.Box2, "B" + (char)(curr.X) + (
            char)(curr.Y) + (char)(targ.X) + (char)(targ.Y) + (char)orient +(
            char)(stepLength/100) + (char)(stepLength%100) + (char)(turnLength
            /100) + (char)(turnLength%100));
        currentPosition.X = curr.X;
        currentPosition.Y = curr.Y;
    }

    public void StopExploration()
    {
        brick.CommLink.StopProgram();
    }

    private void GetMessage()
    {
        string receivedData;
        char[] dataArray;
        while (true)
        {
            try /* Minor Communication */
            {   /* Data exchange */
                if (brick.IsConnected == false) this.Dispose();
                receivedData = brick.CommLink.MessageRead(NxtMailbox2.Box3,
                    NxtMailbox.Box3, true);
                dataArray = receivedData.ToCharArray();

                /* Data update */
                sonarFront = dataArray[0];
                sonarLeft = dataArray[1];
                sonarRight = dataArray[2];
                currentPosition.X = dataArray[3]; currentPosition.Y = dataArray
                    [4];
                color = dataArray[5];
                orientation = dataArray[6];
            }
            catch (NxtCommunicationProtocolException exception)
            {

            }
```

```
            try
            {   /* Next step request, data update management */
                if (brick.IsConnected == false) this.Dispose();
                receivedData = brick.CommLink.MessageRead(NxtMailbox2.Box1,
                    NxtMailbox.Box1, true);
                dataArray = receivedData.ToCharArray();

                /* Update Data */
                currentPosition.X = dataArray[0]; currentPosition.Y = dataArray
                    [1];
                targetPosition.X = dataArray[2]; targetPosition.Y = dataArray
                    [3];

                for (int x = 0; x < 7; x++)
                {
                    for (int y = 0; y < 7; y++)
                    {
                        int ax = currentPosition.X - 3 + x;
                        int ay = currentPosition.Y - 3 + y;
                        if (ax >= 0 && ax < 16 && ay >= 0 && ay < 16)
                        {
                            map.fields[ax][ay].Parse((byte)(dataArray[x * 7 + y
                                + 4] & 0x0f));
                        }
                    }
                }
                orientation = dataArray[55];
                if (dataArray[54] == 'R') /* Request for next step */
                {/* Process Map */
                    MapField next = map.ShortestStep(currentPosition,
                        targetPosition);

                    /* Send Reply */
                    if (next == null)
                        brick.CommLink.MessageWrite(NxtMailbox.Box2, "X");
                    else
                    {
                        brick.CommLink.MessageWrite(NxtMailbox.Box2, "N" + (
                            char)(next.Position.X) + (char)(next.Position.Y));
                    }
                }
            }
            catch (NxtCommunicationProtocolException exception)
            {

            }
            Thread.Sleep(50);
        }
    }
    /*Not used anymore, not finished error code parsing
     *
    static public string TranslateException(NxtCommunicationProtocolException
        exception)
    {
```

```
            string[] split = exception.Message.Split(' ');
            string ErrorCode;
            if (split.Length > 15) ErrorCode = split[15];
            else return "Other Exception";
            switch (ErrorCode)
            {
                case "0x20": return "Pending communication transaction in progress
                    ";
                case "0x40": return "Specified mailbox queue is empty";
                case "0xBD": return "Request failed (i.e. specified file not found)
                    ";
                case "0xBE": return "Unknown command opcode";
                case "0xBF": return "Insane packet";
                case "0xC0": return "Data contains out-of-range values";
                case "0xDD": return "Communication bus error";
                case "0xDE": return "No free memory in communication buffer";
                case "0xDF": return "Specified channel/connection is not valid";
                case "0xE0": return "Specified channel/connection not configured or
                     busy";
                case "0xEC": return "No active program";
                case "0xED": return "Illegal size specified";
                case "0xEE": return "Illegal mailbox queue ID specified";
                case "0xEF": return "Attempted to access invalid field of a
                    structure";
                case "0xF0": return "Bad input or output specified";
                case "0xFB": return "Insufficient memory available";
                case "0xFF": return "Bad arguments";
                default: return "Other exception";
            }
        }*/

        public void StartProgram()
        {
            brick.CommLink.StartProgram("mazesolv.rxe");
        }
    }
}

public class MapField
{
    public bool Xm { get; set; }
    public bool Xp { get; set; }
    public bool Yp { get; set; }
    public bool Ym { get; set; }
    public Point Position;

    public MapField(Point p)
    {
        Xp = Xm = Yp = Ym = false;
        Position = p;
    }

   /* Not used anymore
    *
```

```
    public override Boolean Equals(object o)
    {
        if (o == null) return false;
        if (o.GetType() != this.GetType()) return false;
        MapField ob = (MapField)o;
        if (!ob.Position.Equals(this.Position)) return false;
        if (ob.Xm == Xm && ob.Xp == Xp && Ym == ob.Ym && ob.Yp == Yp) return true;
        return false;
    }*/

    public void Parse(byte nibbleToParse)
    {
        Yp = ((nibbleToParse & 0x01) == 0 ? false : true);
        Xp = ((nibbleToParse & 0x02) == 0 ? false : true);
        Ym = ((nibbleToParse & 0x04) == 0 ? false : true);
        Xm = ((nibbleToParse & 0x08) == 0 ? false : true);
    }
}

public class Map
{
    public MapField[][] fields;

    public Map()
    {
        fields = new MapField[16][];
        for (int i = 0; i < 16; i++)
        {
            fields[i] = new MapField[16];
            for (int j = 0; j < 16; j++) fields[i][j] = new MapField(new Point(i, j
                ));
        }
    }

    public MapField GetFieldNo(int no)
    {
        no %= 256;
        return fields[no / 16][no % 16];
    }

    internal MapField ShortestStep(Point currentPosition, Point targetPosition)
    {
        /* A* shortest way algorithm */
        double[] f_score = new double[256];
        double[] g_score = new double[256];
        double[] h_score = new double[256];
        MapField[] came_from = new MapField[256];
        ArrayList closedSet = new ArrayList();
        ArrayList openSet = new ArrayList();
        g_score[pti(currentPosition)] = 0;
        h_score[pti(currentPosition)] = Distance(currentPosition, targetPosition);
        f_score[pti(currentPosition)] = h_score[pti(currentPosition)];
        came_from[pti(currentPosition)] = null;
        openSet.Add(fields[currentPosition.X][currentPosition.Y]);
```

```
while (openSet.Count > 0)
{
    MapField lowest = (MapField)(openSet.ToArray())[0];
    foreach (MapField i in openSet.ToArray())
    {
        if (f_score[pti(i.Position)] > f_score[pti(i.Position)]) lowest = i
            ;
    }
    if (lowest.Position.Equals(targetPosition))
    {
        while (!came_from[pti(lowest.Position)].Position.Equals(
            currentPosition))
        {
            lowest = came_from[pti(lowest.Position)];
        }
        return lowest;
    }
    openSet.Remove(lowest);
    closedSet.Add(lowest);
    ArrayList neighbours = new ArrayList();
    if (lowest.Position.X > 0 && lowest.Xm == false)
        neighbours.Add(fields[lowest.Position.X - 1][lowest.Position.Y]);
    if (lowest.Position.X < 15 && lowest.Xp == false)
        neighbours.Add(fields[lowest.Position.X + 1][lowest.Position.Y]);
    if (lowest.Position.Y > 0 && lowest.Ym == false)
        neighbours.Add(fields[lowest.Position.X][lowest.Position.Y - 1]);
    if (lowest.Position.Y < 15 && lowest.Yp == false)
        neighbours.Add(fields[lowest.Position.X][lowest.Position.Y + 1]);
    foreach (MapField field in neighbours.ToArray())
    {
        if (closedSet.Contains(field)) continue;
        double tentative_g = g_score[pti(lowest.Position)];
        bool better = false;
        if (!openSet.Contains(field))
        {
            openSet.Add(field);
            better = true;
        }
        else if (tentative_g < g_score[pti(field.Position)])
        {
            better = false;
        }
        else better = false;

        if (better)
        {
            came_from[pti(field.Position)] = lowest;
            g_score[pti(field.Position)] = tentative_g;
            h_score[pti(field.Position)] = Distance(field.Position,
                targetPosition);
            f_score[pti(field.Position)] = g_score[pti(field.Position)] +
                h_score[pti(field.Position)];
        }
    }
```

```
        }
        return null;
    }

    private double Distance(Point a, Point b)
    {
        return abs(a.X - b.X) + abs(a.Y - b.Y);
    }

    private int abs(int p)
    {
        return (p > 0 ? p : -p);
    }

    internal int pti(Point x) /* Point to integer, used with 1-dimensional arrays
        */
    {
        return x.X * 16 + x.Y;
    }
}

public enum Orientation
{
    Yp = 0, Xp = 1, Ym = 2, Xm = 3
}
```