

Politechnika Świętokrzyska
Wydział Mechatroniki i Budowy Maszyn
Katedra Automatyki i Robotyki

Kierunek: Automatyka i Robotyka
Specjalność: Automatyka przemysłowa

inż. Szczepan Kostecki

Praca dyplomowa magisterska

Temat pracy magisterskiej:
Projekt systemu mapowania otoczenia oraz planowania ruchu robota mobilnego.

Project of the environment mapping system and mobile robot motion planning.

opiekun pracy dyplomowej:
dr hab. inż. Paweł Łaski prof. PŚk

Kielce 2020r.

POLITECHNIKA ŚWIĘTOKRZYSKA w Kielcach
WYDZIAŁ MECHATRONIKI I BUDOWY MASZYN
KATEDRA AUTOMATYKI I ROBOTYKI

DZIEKAN
Zatwierdzam: Wydział Maszyn

Dziekan WMiBM
prof. dr hab. inż. Robert L. Stańczyk

Rok akademicki 2019/2020

**ZADANIE
NA PRACĘ DYPLOMOWĄ MAGISTERSKĄ**

Rodzaj studiów: stacjonarne

Kierunek: Automatyka i Robotyka

Specjalność: Automatyka Przemysłowa

Wydano studentowi: **Szczepan Kostecki**

I. Temat pracy: Projekt systemu mapowania otoczenia oraz planowania ruchu robota mobilnego.
Project of the environment mapping system and mobile robot motion planning.

II. Plan pracy:

1. Wstęp.
2. Dobór elementów systemu.
3. Badania modelowe i eksperymentalne systemu planowania ruchu.
4. Wnioski i podsumowanie.
5. Literatura

III. Cel pracy:

IV. Uwagi dotyczące pracy:

Celem pracy jest projekt systemu mapowania otoczenia oraz planowania ruchu robota mobilnego.

V. Termin oddania pracy: zgodnie z Regulaminem Studiów

VI. Konsultanci:

mgr inż. Gabriel Bracha, mgr inż. Dawid Pietrala, mgr inż. Krzysztof Borkowski

VII. Recenzent

dr hab. inż. Leszek Płonecki prof. PŚk

Opiekun pracy dyplomowej:

.....
dr inż. Paweł Łaski

Temat pracy dyplomowej magisterskiej celem jej wykonania otrzymałem:

Kielce, dnia/...../2019 r.

podpis:

SPIS TREŚCI

WPROWADZENIE	5
1. Wiadomości wstępne	7
1.1. Systemy detekcji oraz położenia obiektów i przeszkód	7
1.1.1. Systemy radarowe	7
1.1.2. LIDAR	8
1.1.3. Dalmierze ultradźwiękowe	10
1.1.4. Dalmierze odbiciowe IR	11
1.1.5. Systemy wizyjne	13
1.2. Samolokalizacja i mapowanie	14
2. Dobór elementów systemu	17
2.1. Platforma mobilna	17
2.2. Układ sterowania platformą mobilną	19
2.3. Skaner laserowy 2D.....	22
2.4. Komputer jednoukładowy	24
3. Projekt systemu	27
3.1. Koncepcja budowy i działania systemu	27
3.2. Oprogramowanie platformy mobilnej	28
3.2.1. Odbiór i nadawanie danych.....	29
3.2.2. Zmiana trybów pracy	34
3.2.3. Odometria.....	36
3.2.4. Tryb autonomiczny	36
3.3. Aplikacja mobilna do sterowania platformą mobilną	38
3.4. Aplikacja komputera jednoukładowego	43
3.4.1. Interfejs graficzny	43
3.4.2. Obsługa skanera laserowego RPLidar A2M8	45
3.4.3. Wyznaczenie pozycji przeszkoły	49
3.4.4. Generowanie mapy.....	50
3.4.5. Algorytm wyznaczania trasy	54
4. Badania symulacyjne i eksperymentalne systemu.....	57
4.1. Badanie poprawności działania algorytmu mapowania	57
4.1.1. Tworzenie mapy na podstawie punktów testowych.....	57
4.1.2. Przejazd testowy.....	59

4.2. Badanie poprawności działania algorytmu planowania ruchu.....	61
4.2.1. Wyznaczenie toru ruchu w losowo wygenerowanej mapie.....	61
4.2.2. Wyznaczenie ścieżki ruchu robota na torze testowym	63
5. Wnioski i podsumowanie.....	65
5.1. Wnioski	65
5.2. Nabyte umiejętności.....	65
5.3. Możliwości rozwoju systemu.....	66
LITERATURA	67
ZAŁĄCZNIKI.....	71

WPROWADZENIE

Początek XXI wieku to ciągły rozwój komputeryzacji zarówno po stronie sprzętowej jak i oprogramowania. Każdego roku na rynek trafiają kolejne generacje urządzeń wykorzystujących nowe wydajniejsze komponenty, co umożliwia implementację oprogramowania bazującego na bardziej rozbudowanych algorytmach. Obecny trend na „smart” urządzenia przyśpieszył rozwój systemów wbudowanych opartych o mikroprocesory. Oprogramowanie tych układów niejednokrotnie korzysta z rozwiązań uczenia maszynowego, logiki rozmytej czy też sieci neuronowych.

Jednym z elementarnych zadań ówczesnej automatyki jest zastąpienie ręcznego sterowania lub kierowania maszynami i urządzeniami poprzez zastosowanie wyspecjalizowanych sterowników i algorytmów sterujących. Ostatnie lata to również intensywne badania nad autonomicznymi pojazdami, gdzie system komputerowy jest w stanie zastąpić kierowcę. Największymi sukcesami w tej dziedzinie mogą pochwalić się Tesla Motors oraz Google. Pierwszy z producentów wyposaży swoje auta w autopilota, który przy odpowiednim oznakowaniu drogi ma możliwość w pełni przejąć zadania kierowcy. Jednak działanie systemu kontroluje człowiek i algorytm okresowo sprawdza jego czujność w razie wystąpienia sytuacji niebezpiecznej. Według klasyfikacji wprowadzonej przez SEA International autopilot Tesli możemy zaklasyfikować do 3 poziomu z 5 poziomowej skali autonomii pojazdów[1]. Natomiast systemy sterowania autonomicznego Google bazują na wcześniej utworzonych mapach przestrzennych tras po których porusza się samochód. Oba rozwiązania bazują na samochodach o napędzie elektrycznym i zastosowaniu czujników radarowych, dalmierzy laserowych oraz kamer[2]. Wykonywanie algorytmu sterującego przez komputer pokładowy jest dodatkowo wspomagane poprzez sieć komputerową do której jest on podłączony bezprzewodowo.

Pojazdy autonomiczne to nie tylko samochody, ale także mobilne roboty wykorzystywane przez służby ratownicze oraz przemysł m. in. do inspekcji przestrzeni trudno dostępnych czy miejsc w których występują zagrożenia dla zdrowia bądź życia człowieka. Dzięki zastosowaniu w nich układów detekcji i rozpoznawania przeszkoł co daje możliwość poruszania się w zmiennym nie znanym a priori otoczeniu[3]. Zadaniem operatora takiego robota jest jedynie wskazania miejsca docelowego oraz obserwowanie parametrów przesyłanych zwrotnie do terminala sterowniczego.

Celem tej pracy jest wykonanie projektu systemu tworzącego dwuwymiarową mapę otoczenia dla niewielkiego pojazdu autonomicznego. Zadaniem zaproponowanego algorytmu robota jest wyznaczenie trasy przejazdu na postawie wygenerowanej mapy do zadanego punktu docelowego z uwzględnieniem przeszkód występujących w jego pobliżu. Poprawność działania zaimplementowanych rozwiązań należy poprzeć wynikami badań przeprowadzanych w trybie symulacyjnym oraz eksperymentalnym.

W realizacji projektu systemu mapowania i planowania toru ruchu dla robota mobilnego przyjęto następujące założenia:

- samodzielne wykrywanie przeszkód na podstawie danych z systemu pomiarowego,
- tworzenie stref bezpieczeństwa w pobliżu obiektów w celu uniknięcia kolizji z nimi,
- planowanie ruchu poprzez wyznaczanie punktów pośrednich trasy,
- niezależność systemu tworzenia mapy od robota,
- dobór parametrów pracy algorytmu przez operatora,
- zdalny podgląd aktualnego stanu systemu,
- wykorzystanie zabezpieczeń poprawności komunikacji pomiędzy systemem mapującym a robotem.

W rozdziale pierwszym zawarto informacje wstępne związane z systemem mapowania i samolokalizacji oraz z najpopularniejszymi rodzajami używanych do tego celu sensorami. Przedstawiono zasadę wykonywania pomiarów, wady i zalety dla danego typu oraz ich zastosowanie. Rozdział drugi opisuje użyte w projekcie komponenty systemu. Wskazana została platforma mobilna, sensor skanujący otoczenie oraz elektronika odpowiedzialna za przetwarzanie informacji. Opisane zostały także najważniejsze parametry wybranych elementów. Projekt systemu został przedstawiony w rozdziale trzecim. Zawiera on m. in. koncepcje działania oraz opisuje utworzone na potrzeby projektu oprogramowanie. Kolejny rozdział zawiera wyniki przeprowadzonych badań symulacyjnych i eksperymentalnych zaproponowanego systemu. W podsumowaniu zawarto wnioski wyprowadzone podczas pracy nad projektem, wskazano zdobyte doświadczenie oraz zaproponowano możliwości rozwoju i modyfikacji zaproponowanych rozwiązań.

1. WIADOMOŚCI WSTĘPNE

1.1. SYSTEMY DETEKCJI ORAZ POŁOŻENIA OBIEKTÓW I PRZESZKÓD

Zdolność autonomicznego poruszania się robotów mobilnych opiera się na poznawaniu otoczenia przy pomocy zestawu czujników dostarczających informacji o pobliskiej przestrzeni. Sensoryka tych urządzeń korzysta głównie z:

- systemów radarowych,
- LIDAR,
- dalmierzy ultradźwiękowych,
- dalmierzy odbiciowych IR,
- dalmierzy laserowych ToF (*Time of Flight*),
- systemów wizyjnych.

1.1.1. Systemy radarowe

Radar, czyli *Radio Detecting and Ranging*, to określenie urządzenia posiadającego zdolność do wykrywania obecności obiektów w polu jego widzenia za pomocą odbitego od tego obiektu promieniowania elektromagnetycznego. Pomiar radarowy rozpoczyna się od wyemitowania w określonym kierunku przez antenę nadawczą impulsu wąskiej wiązki sygnału o wysokiej częstotliwości (od 3 Mhz do 110 Ghz), który porusza się w przestrzeni z prędkością światła. Jeżeli na swojej drodze napotka on na przeszkodę, jego energia zostanie rozproszona a jej część powróci po pewnym czasie do anteny odbiornika w postaci echa [15].

Wyznaczenie odległości pomiędzy radarem a obiektem dokonuje na podstawie czasu powrotu sygnału echa i określa się wzorem (1).

$$R = \frac{c_0 \cdot t_e}{2} \quad (1)$$

gdzie: R – rzeczywista odległość do przeszkody [m],

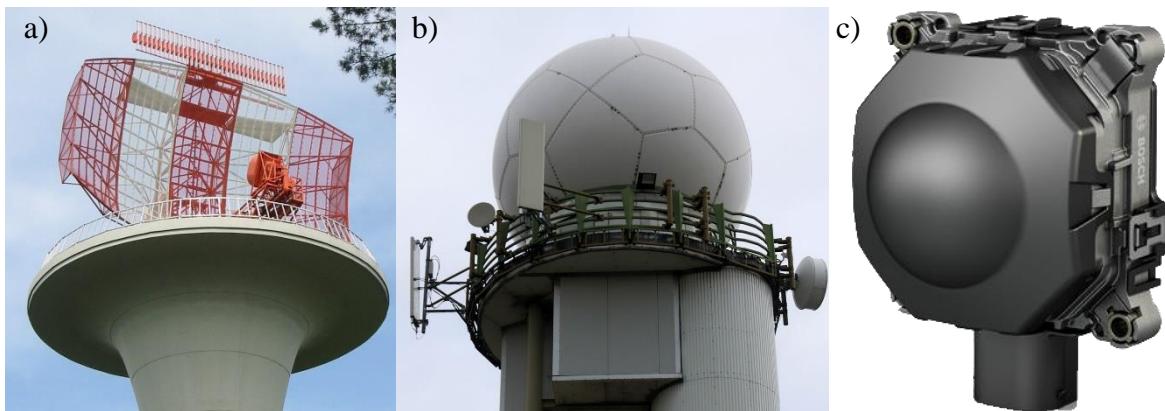
c_0 – prędkość światła: 299 792 458 m/s.

t_e – czas pomiędzy wyemitowaniem impulsu a powrotem echa [s]

Na działanie systemów radiolokacyjnych nie mają wpływu takie czynniki jak mgła, zachmurzenie, ciemność czy opady atmosferyczne co umożliwia wykrycie obiektów nie widocznych „gołym okiem”. Radary mają możliwość skanowania pewnego wycinka horyzontu bądź, jak jest w większości przypadków dzięki wirującym antenom nadawczym i odbiorczym,

są w stanie wykonywać pomiary w pełnym zakresie kątowym. Jednocześnie charakteryzują się one dużym zasięgiem (od kilkuset metrów do kilkuset kilometrów), precyzją pomiarów (poniżej 1 m) oraz możliwością detekcji pewnych określonych typów obiektów poprzez odpowiedni sygnał nadawany. Znalazły one zastosowanie w takich dziedzinach jak:

- lotnictwo – określanie położenia statków powietrznych (kontrola lotów) (rys. 1 a),
- meteorologia – generowanie map zachmurzenia i opadów (rys. 1 b),
- geologia – wykrywanie podziemnych instalacji, podziemnych przeszkód czy pustek, sprawdzanie struktur glebowych bądź infrastruktury drogowej,
- motoryzacja – sensory układów wspomagających kierowcę (np. aktywny tempomat, automatyczne awaryjne hamowanie) oraz w systemach autonomicznych (rys. 1 c).



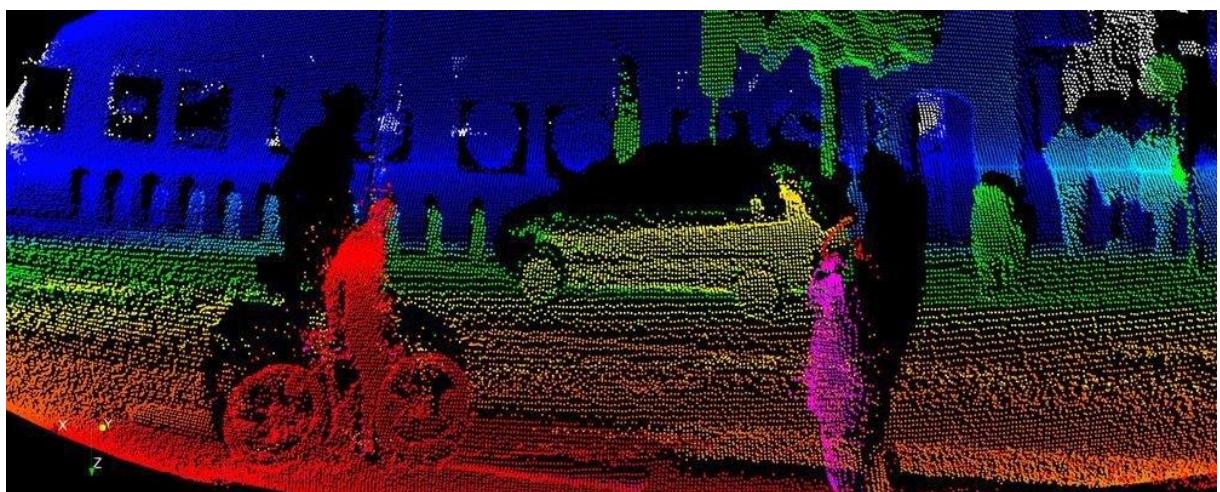
Rys. 1. Przykłady radarów a) lotniczy [16], b) meteorologiczny [17], c) sensor Bosch LRR4 [18].

1.1.2. LIDAR

LIDAR, czyli *Light Detection and Ranging* jest urządzeniem działającym tej samej zasadzie co radar. Tutaj również wysyłany jest krótki impuls promieniowania elektromagnetycznego i po odbiciu wiązki od obiektu trafia ona na detektor po pewnym upływie czasu, który jest proporcjonalny do odległości względem przeszkody. LIDAR wykorzystuje jednak do pomiaru światło laserowe zamiast najczęściej używanych w radarach mikrofal. W przypadku wykonywania pomiarów naziemnych, gdzie promienie świetlne poruszają się w powietrzu, długość fali generowanej przez laser mieści się w zakresie podczerwieni (głównie od 900 nm do 1064 nm). Natomiast w przypadku pomiarów pod wodą używany jest głównie laser zielony (o długości fali 532 nm)[19]. Użycie wiązki laserowej wymusiło zastosowanie w nich toru optycznego jak np. zestawy ruchomych luster. W związku z tym sensory te można zakwalifikować do grupy urządzeń optoelektromechanicznych.

Pierwotnym zastosowaniem LIDAR-ów było tworzenie map ukształtowania terenu, które zastosowanie mają w hydrologii, geologii oraz geomorfologii. Trójwymiarowe pomiary laserowe obecnie używane są m. in. w budowie infrastruktury drogowej i uzbrojenia terenu, leśnictwie i archeologii. Urządzenia tego typu tworzyły również mapy powierzchni Księżyca podczas misji Apollo 15.

Obecnie branża motoryzacyjna w projektowaniu nowych samochodów coraz częściej korzysta z skanerów laserowych, które wchodzą w skład systemów wspomagania kierowców ADAS (*Advanced Driver Assistance Systems*) oraz w systemów mapowania terenu pojazdów autonomicznych i bezzałogowych, gdzie ich zadaniem jest wykrywanie oraz lokalizowanie przeszkód na drodze (rys. 2).



Rys. 2. Wizualizacja danych z trójwymiarowego LIDAR-u zamontowanego w samochodzie [20].

Skanery laserowe występują w odmianach dwu i trójwymiarowych. W pierwszej z nich nadawana wiązka sterowana jest tylko w jednej osi w wyniku czego otrzymujemy pomiar w płaszczyźnie dwuwymiarowej. W odmianie 3D strumień świetlny odchylany jest zarówno horyzontalnie jak i wertykalnie dzięki czemu otrzymane dane można przedstawić w przestrzeni trójwymiarowej.

Do zalet LIDAR-ów możemy zaliczyć:

- wysoka częstotliwość próbkowania do kilkunastu kHz,
- zasięg pomiarowy od kilku metrów do kilkuset metrów,
- rozdzielcość sięgająca do 1mm,
- dokładność pomiarowa na poziomie kilkunastu milimetrów,
- brak wpływu warunków oświetleniowych na pomiar.

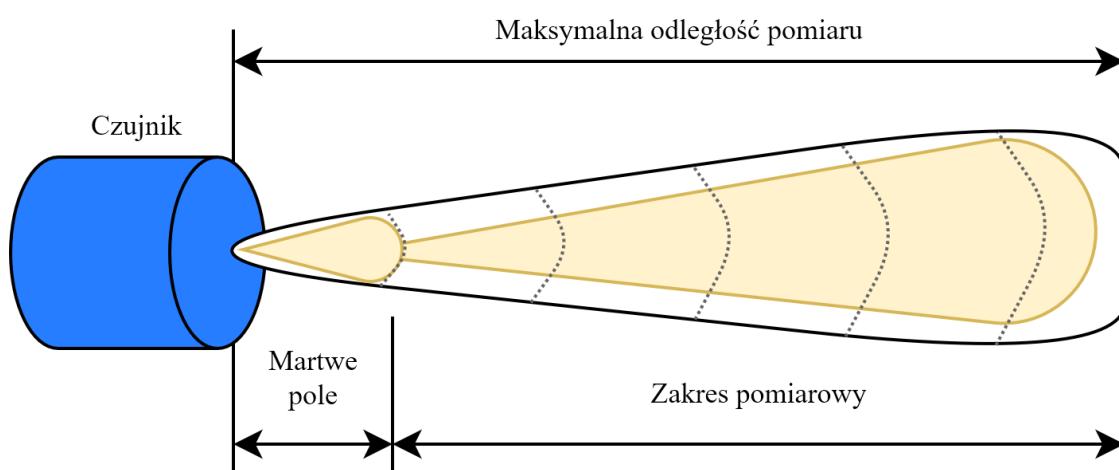
Posiadają one jednak też wady:

- brak dowolności w umiejscowieniu (zarówno źródło wiązki laserowej oraz powierzchnia detektora nie mogą zostać zakryte elementami konstrukcyjnymi urządzenia),
- skomplikowana budowa przez połączenie elementów elektronicznych, optycznych i mechanicznych,
- wysoka cena spowodowana złożonością konstrukcji i precyzją jej wykonania,
- interferencja światła na krawędzi obiektu co prowadzi do przekłamań w pomiarach,
- różna refleksyjność powierzchni co prowadzi do problemów z odbiciem wiązki laserowej np. od obiektów przezroczystych,
- duży wpływ na pomiar zjawisk takich jak mgła, duże zapylenie czy silne opady deszczu i śniegu.

Zastosowanie skanerów laserowych w autonomicznych pojazdach i robotach mobilnych do wykrywania i lokalizacji przeszkód a także mapowania otoczenia jest obiektem intensywnych badań [4][5] co owocuje kolejnymi udoskonaleniami w tej technice.

1.1.3. Dalmierze ultradźwiękowe

Dalmierze ultradźwiękowe i sonary do pomiaru odległości wykorzystują fale dźwiękowe o wysokiej częstotliwości wykraczające poza zakres słyszalny przez człowieka a zasada pomiaru jest analogiczna do pomiarów radarowych czy przy użyciu skanerów laserowych. Dzięki mniejszej prędkości rozchodzenia się dźwięku układ przetwarzający takiego czujnika jest mniej skomplikowany od wcześniej opisanych typów sensorów.



Rys. 3. Pole widzenia czujnika ultradźwiękowego.

Częstotliwość pracy czujników ultradźwiękowych jest odwrotnie proporcjonalna do ich zakresu pomiarowego co jest ich cechą charakterystyczną. Użycie fal o częstotliwości 50 kHz ogranicza ich zasięg do około 10 m, natomiast jeżeli częstotliwość wzrośnie do 200 kHz maksymalna mierzona odległość spada do 1 m [21]. Ich pole widzenia z kształtem zbliżonym do stożka (rys. 3) również jest ich rozpoznawalną cechą. Największą zaletą tego typu urządzeń jest możliwość pomiarów pod wodą, gdzie fale dźwiękowe nie są pochłaniane w takim stopniu jak fale elektromagnetyczne.

Do zalet dalmierzy wykorzystujących fale akustyczne można również zaliczyć:

- niewielką cenę z racji prostej budowy,
- brak wpływu na pomiar zjawisk takich jak mgła, duże zapylenie czy silne opady,
- bezproblemowe odbicie fali od większości materiałów.

Jako główne wady dalmierzy ultradźwiękowych można wymienić [6]:

- niewielki zasięg od kilkudziesięciu centymetrów do kilku metrów,
- mniejsza dokładność pomiarowa w porównaniu do innych typów sensorów,
- niewielka dokładność detekcji niewielkich obiektów,
- występowanie strefy martwej tuż przed czujnikiem,
- wpływ zmian temperatury, ciśnienia i wilgotności na propagację fali w ośrodku.

Sensory ultradźwiękowe znalazły szerokie zastosowanie w przemyśle, gdzie na liniach produkcyjnych m. in. czuwają nad poprawnym ustawieniem produktów względem maszyn czy do monitorowania poziomu płynu w urządzeniach napełniających. W motoryzacji są one szeroko stosowane jako czujniki parkowania oraz w instalacjach autoalarmu.

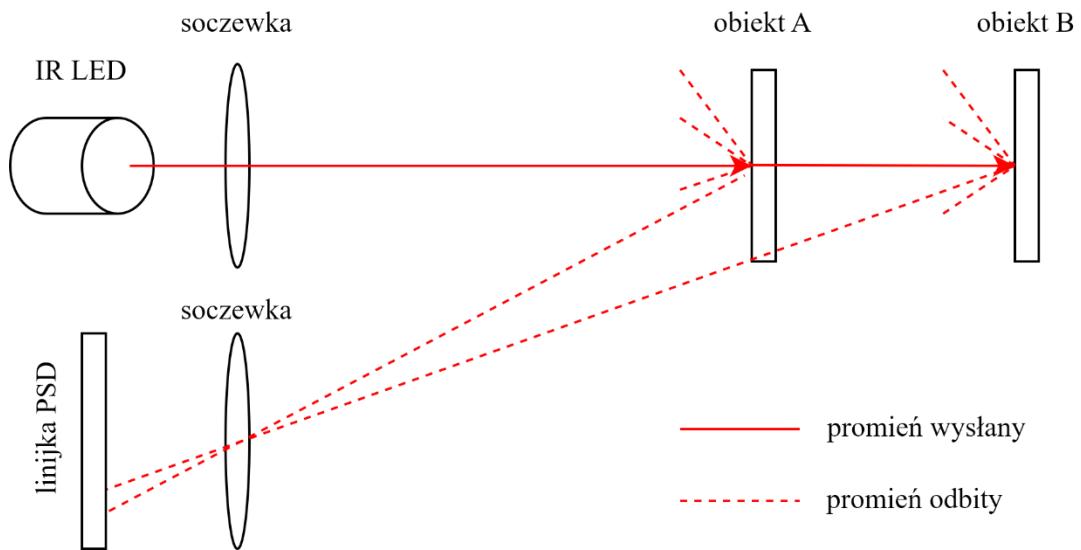
1.1.4. Dalmierze odbiciowe IR

Czujniki optyczne wykorzystujące światło podczerwone stanowią szeroką grupę sensorów wykorzystywanych w automatyce i robotyce. Jednym ich z głównych zastosowań są systemy bezpieczeństwa i zabezpieczeń np. kontrola zamknięcia osłony. Do tego celu wystarczają fotoprzekaźniki, które zwierają (NO) lub rozwierają (NC) styki po wykryciu elementu.

W kontekście pojazdów autonomicznych skupić się należy na dalmierzach wykorzystujących odbicie światła podczerwonego. Ich zasada działania różni się od wcześniej opisanych typów. Nie jest tu mierzony czas do powrotu wiązki a ilość lub położenie wiązki powracającego światła. Podstawowa budowa takiego urządzenia to dioda LED emitująca

światło podczerwone oraz fototranzystor, którego częściowe otwarcie jest sterowane bazą, na którą pada światło powracające po odbiciu od obiektu. Natężenie wiązki odbitej od przedmiotu jest odwrotnie proporcjonalne do kwadratu odległości, co przekłada się na napięcie występujące pomiędzy kolektorem a emiterem fototranzystora. Wartość ta niestety również zależy od wartości współczynnika refleksyjności materiału, od którego odbija się podczerwień. Niedogodność ta dyskwalifikuje użycie tego typu czujników do pomiaru dystansu względem przedmiotów z różnych materiałów i kolorów. Ogranicza to również zasięg tego typu czujników, który nie przekracza kilkudziesięciu centymetrów. Ich bardzo prosta budowa sprawia, że mieścią się w niewielkich obudowach o powierzchni nawet poniżej 1 mm^2 . Dzięki temu znalazły one zastosowanie m. in. jako czujniki zbliżeniowe w smartfonach.

Drugi rodzaj dalmierzy wykorzystujących podczerwień opiera pomiar o triangulację co modyfikuje ich budowę. Tuż za diodą LED znajduje się soczewka skupiąca, a odbiornikiem jest tu czuła na padające światło linijka PSD (Position Sensitive Device)[22], która pozwala zmierzyć miejsce padania wiązki odbitej (rys. 4).



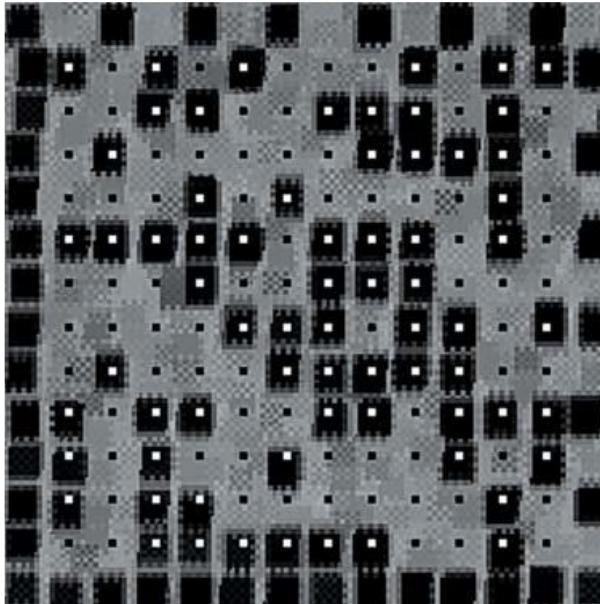
Rys. 4. Zasada działania dalmierzy wykorzystujących triangulację.

Zastosowanie tej metody minimalizuje wpływ współczynnika refleksyjności materiału obiektu, na który pada promień, oraz jednocześnie zwiększy zasięg nawet do kilku metrów. Czujniki te pracują w sposób ciągły co sprawia, że częstotliwość pomiarowa zależna jest od zastosowanego w systemie przetwornika analogowo-cyfrowego. Posiadają one jednak martwe pole zależne od odległości pomiędzy diodą nadawczą a detektorem PSD. Im większa jest ta odległość tym zwiększa się strefa martwa. Są one również wrażliwe na zapylenie rozpraszające nadawaną wiązkę podczerwieni.

1.1.5. Systemy wizyjne

Zwiększenie mocy obliczeniowej procesorów wykorzystywanych w systemach wbudowanych pozwala na zwiększenie ilości danych do przetworzenia w jednostce czasu. Dzięki temu w urządzeniach przemysłowych oraz w systemach autonomicznych coraz częściej stosuje się kamery[7], które dostarczają pełną informację o otoczeniu mieszkającym się w kadrze. Ogromna ilość danych generowanych przez kamery (jeden piksel obrazu reprezentuje nawet kilka bajtów) jest bez odpowiedniego algorytmu ich przetwarzania bezużyteczna. Z tego powodu każdy system wizyjny składa się z dwóch części: sprzętowej i oprogramowania. Programy „widzenia maszynowego” realizują m. in.[8]:

- odczyt i weryfikację napisów oraz oznaczeń (rys. 5),
- pomiary wielkości geometrycznych,
- weryfikacja kolorów, kształtów oraz struktury powierzchni przedmiotów,
- wykrywanie, zliczanie oraz określanie pozycji obiektów.



Rys. 5. Dekodowanie znacznika Data Matrix[8].

Głównymi zadaniami algorytmów przetwarzania obrazu projektowanych na potrzeby robotów autonomicznych, jak i robotów przemysłowych wyposażonych w systemy wizyjne[9], jest rozpoznanie obiektów znajdujących się w widzianej przez kamerę scenie, wyznaczenie ich pozycji oraz orientacji w przestrzeni. Poprawne wykonanie tych czynności pozwala na wyznaczenie prawidłowego toru ruchu robota. Możliwość wykrywania i dekodowania charakterystycznych obiektów jak znaczniki QR Code, ARTag czy Data Matrix pozwala na

zwiększenie precyzji ruchu pojazdu autonomicznego w przypadku, gdy znaczniki niosą dodatkową informację np. o ich lokalizacji.

Dzięki wydajnym modułom obliczeniowym nowoczesne systemy wizyjne są zdolne do wykrywania i śledzenia obiektów znajdujących się w ruchu a także określania ich prędkości oraz kierunku ruchu[10].

Do zalet tego typu rozwiązań zaliczyć można:

- analogiczne działanie do zmysłu wzroku,
- elastyczność systemu – detekcja innego rodzaju obiektów bez zmian sprzętowych,
- duża efektywność – każda klatka zawiera całą scenę,
- brak sygnału nadawanego – kamera rejestruje światło padające od obiektów.

Natomiast wadami są:

- duża ilość danych do przetworzenia przez jednostkę centralną,
- zależność działania od warunków oświetleniowych,
- wrażliwość na zapylenie, opady atmosferyczne czy mgłę,
- skomplikowanie algorytmów przetwarzających.

Główymi obszarami zastosowania „widzenia maszynowego” są:

- systemy kontroli jakości na liniach produkcyjnych,
- sterowanie robotami przemysłowymi np. lakierniczymi,
- montaż i pakowanie,
- segregacja obiektów,
- analiza otoczenia w systemach pojazdów autonomicznych i wspomagania kierowców.

1.2. SAMOLOKALIZACJA I MAPOWANIE

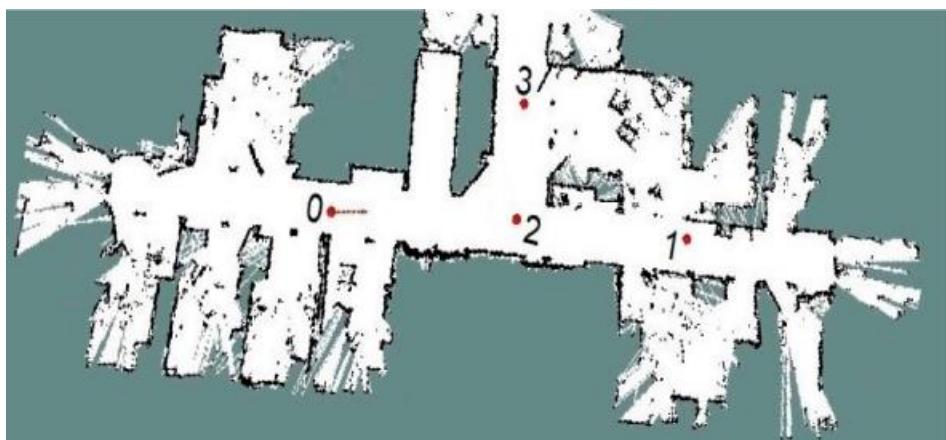
Nawigacja w środowisku trójwymiarowym robotów jest zagadnieniem, nad którym skupia się wiele ośrodków badawczych. Percepcja i interpretacja otoczenia wymagają obróbki dużej ilości danych oraz skomplikowanych programów przetwarzających. Algorytmy jednocośnej samolokalizacji i tworzenia mapy SLAM (*Simultaneous Localization and Mapping*) oraz ich rozwinięcie VSLAM (wykorzystujące system wizyjny) są intensywnie rozwijane. Ich działanie opiera się na zbieraniu danych z czujników rozpoznających odległość do przeszkód (np. LIDAR, zestaw kamer) oraz informacji o estymowanym położeniu robota na podstawie

odometrii[11]. Tworzenie statycznych map lokalizacji przeszkód i stref niedostępnych nie stwarza już obecnie problemów używając precyzyjnych systemów pomiarowych. Problematycznym zagadnieniem jest generowanie map dynamicznie zmieniającego się otoczenia np. w implementacji robota poruszającego się w tłumie.

SLAM nie definiuje jednoznacznie algorytmu a jedynie przedstawia ogólne zasady budowy mapy nieznanego środowiska[12]. Technika ta stosowana jest w:

- urządzeniach poszukiwawczych i ratowniczych USAR (*Urban Search and Rescue*),
- podczas planowania trajektorii dla pojazdów bezzałogowych w kopalniach i magazynach,
- tworzeniu map podczas inspekcji pomieszczeń.

Przykładem algorytmu jednoczesnej samolokalizacji i mapowania jest Hector SLAM będący modułem systemu ROS (*Robot operating system*). Nazwa pochodzi od zespołu Hector (*Heterogeneous Cooperating Team of Robots*) grupy badawczej pracującej na uniwersytecie technicznym w Darmstadt[23]. Rozwiązanie to wykorzystując estymację obserwacyjną, opartą o rozszerzony filtr Kalmana (EKF), jedynie na podstawie danych z LIDAR-u określa położenie urządzenia ma mapie[13] oraz lokalizację przeszkód (rys. 6).



Rys. 6. Przykład wizualizacji mapy wygenerowanej przez Hector SLAM[14].

Obecnie rozwiązania oparte o VSLAM wkroczyły do popularnych autonomicznych robotów sprzątających. Pierwsze generacje tych urządzeń posiadały proste czujniki oraz nieskomplikowany algorytm. Kolejne generacje wyposażano w skanery laserowe, które wykonując pomiary dostarczały danych do generowania mapy pomieszczenia w pamięci urządzenia. Najnowsze generacje są wyposażone w kamerę oraz system przetwarzania obrazu

(rys. 7). Algorytm przetwarzania obrazu bazuje na wykrywaniu krawędzi, a następnie po ich interpretacji do określenia położenia przeszkody, która następnie umieszczana jest w wygenerowanej mapie. Producent dostarczając aplikację na smartfony umożliwił podgląd utworzonej przez urządzenie mapy oraz wskazanie aktualnej jego pozycji.



Rys. 7. Robot sprzątający wykorzystujący technikę VSLAM[24].

2. DOBÓR ELEMENTÓW SYSTEMU

System mapowania otoczenia i planowania trasy ruchu, którego projekt opisuje ta praca, jest elementem współpracującym z kołowym robotem mobilnym. Współpracę tych dwóch urządzeń możemy nazwać pojazdem autonomicznym. W kwestii doboru podzespołów budujących maszynę skupić należy się na:

- platformie mobilnej i jej układu sterowania,
- czujniku detekcji i pomiaru odległości do przeszkód,
- jednostce przetwarzania danych.

2.1. PLATFORMA MOBILNA

Platforma mobilna jest elementem umożliwiającym przemieszczanie się w przestrzeni oraz jednocześnie jest głównym elementem konstrukcyjnym robota. Powinna ona spełniać wymagania co do wytrzymałości, wielkości oraz możliwościach ruchu w zróżnicowanym terenie. Składają się na nią następujące elementy:

- szkielet konstrukcyjny pełniący funkcje nośne oraz elementy obudowy,
- napędy (silniki wraz z przekładniami),
- koła przenoszące moment obrotowy generowany przez napędy,



Rys. 8. Podwozie Mobot Explorer A0 [25].

W projekcie jako platformę wykorzystano podwozie robota Mobilnego Mobot Explorer A0 (rys. 8). Podstawowe dane techniczne zostały przedstawione poniżej (tabela 1). Producent, polska firma Wobit, zajmuje się produkcją i wdrażaniem rozwiązań oraz komponentów dla automatyki.

Tabela 1

Dane techniczne podwozia Mobot Explorer A0

Materiał konstrukcji	stal
Wymiary	115 mm × 361 mm × 350 mm
Liczba silników	4
Model silników	Buechler Motor 1.61.077.714
Średnica kół	120 mm
Prześwit	30 mm
Prędkość maksymalna	0,38 m/s

Konstrukcja podwozia wymagała modyfikacji. Zastosowano w niej silniki prądu stałego z przekładnią. Jednak nie posiadają one enkoderów inkrementalnych pozwalających na pomiar przemieszczenia kątowego koła co utrudnia zastosowanie odometrii.

Problem rozwiązyano poprzez usunięcie niewielkiego fragmentu obudowy osłaniającej oś silnika po przeciwniejszej stronie przekładni oraz zamontowaniu na nim magnetycznego enkodera inkrementalnego CPR64, który następnie został zabezpieczony obudową wykonaną za pomocą drukarki 3D (rys. 9). Opisaną modyfikację zastosowano dla dwóch z czterech silników.

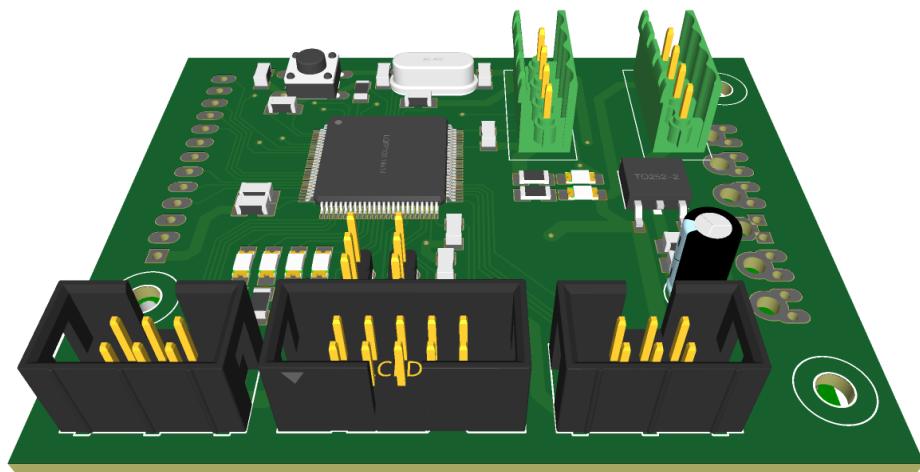
\



Rys. 9. Zmodyfikowany silnik Buechler Motor.

2.2. UKŁAD STEROWANIA PLATFORMĄ MOBILNĄ

Kontrolowany ruch kołowego robota mobilnego uzyskiwany jest dzięki odpowiedniemu sterowaniu jego napędami. Zadanie jest to realizowane przez układ sterowania, który na podstawie zadanych sygnałów np. o prędkości ruchu lub punkcie trasy generuje, przy pomocy algorytmu przetwarzającego, odpowiednie sygnały determinujące pracę efektorów urządzenia.



Rys. 10. Wizualizacja 3D obwodu drukowanego z mikrokontrolerem STM32.

Do sterowania platformą mobilną opisaną w rozdziale 2.1. użyto układu opartego o mikrokontroler STM32F407VGT7 (rys. 10). Obwód skonstruowano na potrzeby robota IMPULS II. Posiada on:

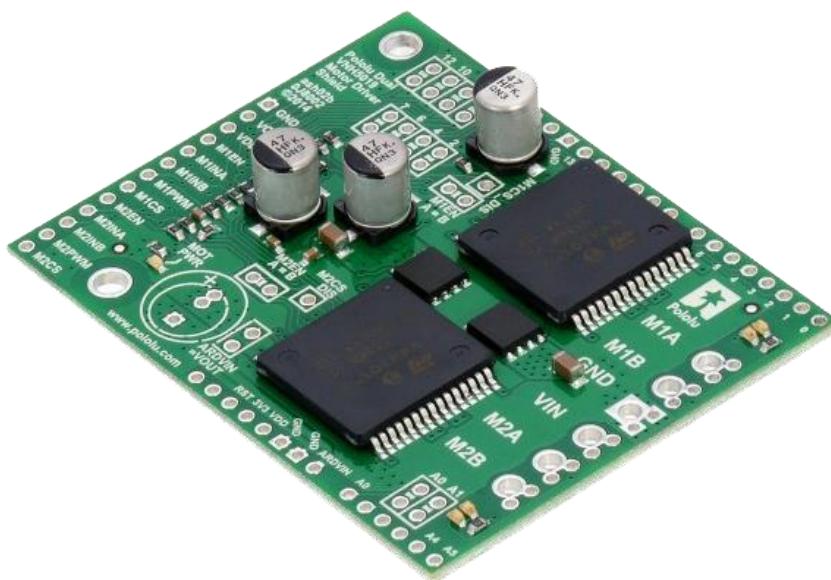
- dwa przyłącza enkoderów inkrementalnych,
- dwa złącza dla interfejsów komunikacyjnych UART oraz SPI,
- złącze programatora ST-Link,
- wyprowadzenia dla modułu sterownika silników prądu stałego,
- przycisk resetowania mikrokontrolera,
- zestaw diod LED połączonych z wyjściami cyfrowymi.

Płytkę z mikrokontrolerem współpracuje z modułem dwukanałowego mostka H VNH5019 (rys. 11), który zamontowany jest do niego za pomocą listwy kołkowej po stronie, na której nie znajdują się elementy elektroniczne. Jego parametry zawarta tabela 2. Możliwość bezprzewodowego zdalnego sterowania robotem uzyskano dzięki modułowi WIFI-RS232 (rys. 12). Jego specyfikacje przedstawia tabela 3. Jako źródło zasilania został użyty pakiet akumulatorów litowo-polimerowych o pojemności 2 Ah i znamionowym napięciu pracy 11,1 V.

Tabela 2

Specyfikacja dwukanałowego mostka H VNH5019

Napięcie zasilania	od 5,5 V do 24 V
Prąd wyjściowy ciągły	12 A
Maksymalny chwilowy prąd	30 A
Napięcie pracy części logicznej	od 3 V do 5 V
Maksymalna częstotliwość PWM	20 kHz
Pomiar prądu	Analogowy 140 mV/A
Zabezpieczenia	Przed odwrotną polaryzacją zasilania
	Zwarcia do masy i zasilania
	Termiczne przed przeciążeniem



Rys. 11. Moduł dwukanałowego mostka H VNH5019 [26]



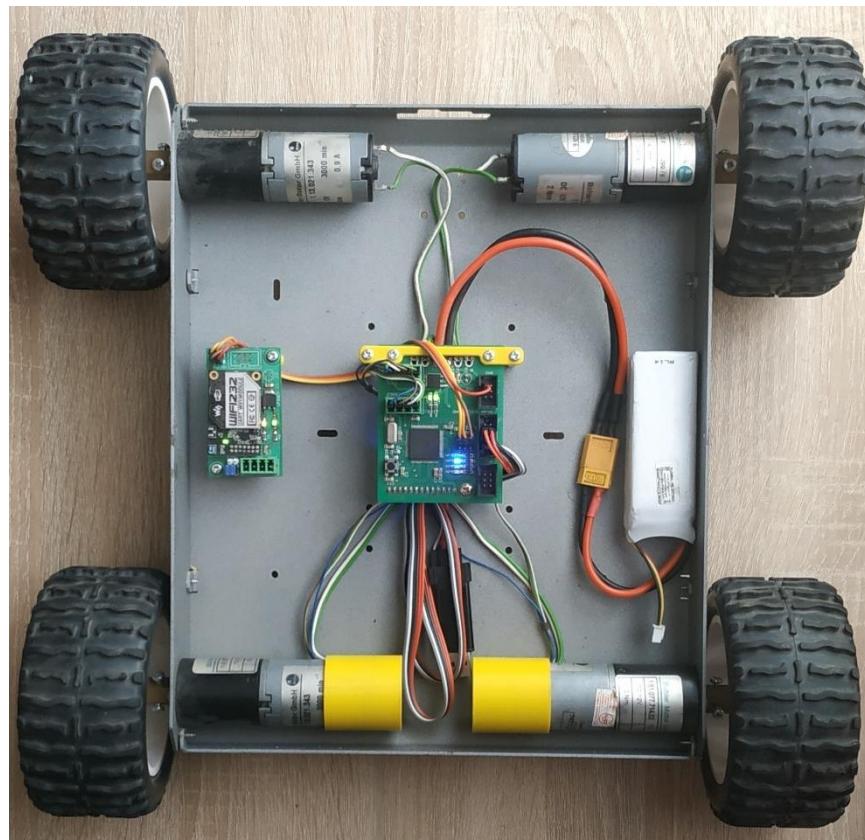
Rys. 12. Moduł komunikacji bezprzewodowej WiFi [27]

Tabela 3

Parametry modułu komunikacyjnego WiFi232

Napięcie zasilania	3,3 V
Maksymalny pobór prądu	0,3 A
Maksymalna prędkość transmisji	450 kb/s
Obsługa szyfrowania	Tak (WEP, WPA2-PSK)
Protokół sieci bezprzewodowej	IEEE 802.11 b/g/n
Wymiary modułu	40 mm × 25 mm
Obsługiwane protokoły	TCP, UDP, ARP, HTTP, DNS, DHCP, ICMP

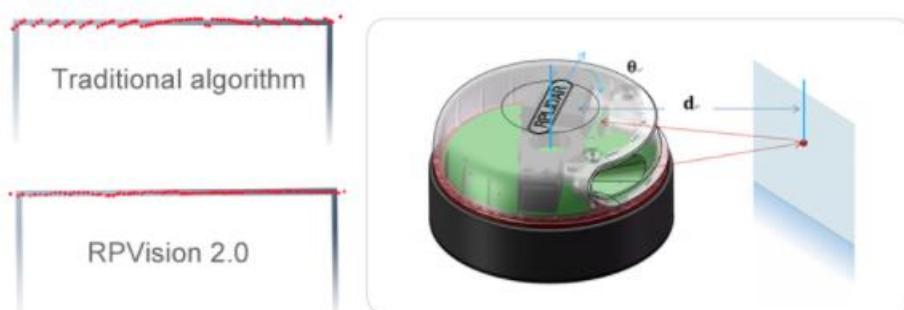
Końcowe przygotowanie platformy mobilnej polegało na kompletacji elementów podwozia Mobot Explorer A0 oraz odpowiednim połączeniom elektrycznym pomiędzy modułami oraz silnikami.



Rys. 13. Przygotowana platforma mobilna.

2.3. SKANER LASEROWY 2D

Dobranym w projekcie elementem dostarczającym informacji o przestrzeni w otoczeniu robota został skaner laserowy RPLidar A2M8. Dzięki obrotowej głowicy, napędzanej silnikiem BLDC, zapewnia on pomiar w pełnym zakresie kątowym. Kompaktowa obudowa jest zaletą w przypadku użycia LIDAR-u w budowie mobilnych pojazdów autonomicznych. Urządzenie cechuje się znacznym poprawieniem odwzorowywania konturów względem innych skanerów dzięki zastosowaniu algorytmu kompensacyjnego RPVision 2.0 (rys. 14)



Rys. 14. Porównanie działania skanera RPLidar z innymi rozwiązaniami [28].

W trakcie wykonywania pomiaru urządzenie wysyła bardzo krótkie impulsy światła podczerwonego o długości fali 785 nm i czasie trwania poniżej 90 μ s. Modulacja wiązki laserowej minimalizuje wpływ oświetlenia zewnętrznego na wynik pomiaru dzięki czemu możliwe jest użycie zarówno wewnętrz pomieszczeń oraz na zewnątrz. Zastosowany laser pracuje z mocą 3 mV. Spełnia on certyfikację Class 1 przez co emitowane promieniowanie jest w pełni bezpieczne dla wzroku [28]. Dane pomiarowe są przesyłane w postaci pakietów przez interfejs komunikacyjnego UART do układu nadzawanego, który jednocześnie przesyła tym samym portem komendy sterujące pracą LIDAR-u.

Główne parametry pomiarów skanera RPLidar A2M8:

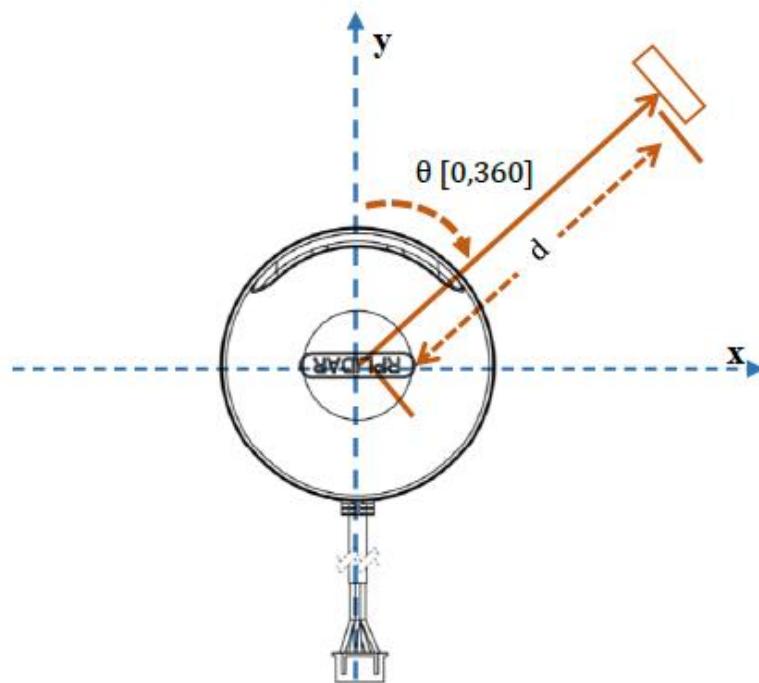
- zasięg pomiarowy od 0,2 m do 12 m,
- zakres kątowy od 0 do 360 stopni,
- czas pojedynczego pomiaru 125 μ s,
- liczba pomiarów w ciągu sekundy 8000,
- rozdzielcość poniżej 2 mm,
- rozdzielcość kątowa od 0,45 do 1,35 stopnia (zależna od częstotliwości skanowania),
- częstotliwość skanowania od 5 Hz do 15 Hz.

Znamionowe wartości elektryczne:

- napięcie zasilania od 4,9 V do 5,5 V,
- maksymalny chwilowy pobór prądu 2,5 A,
- natężenie prądu podczas pomiaru 0,5 A,
- natężenie prądu podczas stanu bezczynności 0,2 A,
- napięcie pracy części logicznej 3,3 V.

Warto zwrócić uwagę na wysoką wartość maksymalnego chwilowego natężenia prądu. Występuje ona jedynie w bardzo niewielkim ułamku sekundy podczas startu silnika napędzającego obrotową głowicę, determinuje to użycie źródła zasilania o odpowiedniej wydajności prądowej. Podczas ustabilizowanej ciągłej pracy natężenie prądu nie przekracza 0,5 A.

Wynikiem pomiarów wykonywanych przez LIDAR jest odległość d do obiektu znajdującego się pod danym kątem θ (rys. 15). Ramki danych przesyłane przez port szeregowy składają się z kilkunastu pomiarów oraz sumy kontrolnej całego pakietu.



Rys. 15. Zasada określania położenia przeszkody w układzie współrzędnych [28].

2.4. KOMPUTER JEDNOUKŁADOWY

Głównym elementem systemu mapowania i systemu mapowania otoczenia oraz planowania ruchu robota mobilnego jest jednostka centralna. Jej algorytm odbiera dane wysyłane w trakcie wykonywania pomiarów przez LIDAR. Na ich podstawie w trakcie ruchu platformy mobilnej generowana jest w pamięci dwuwymiarowa mapa przeszkód, która jest podstawą do wyznaczenia trasy przejazdu do punktu docelowego.

W projekcie może zostać użyty dowolny komputer jednoukładowy bądź laptop z systemem operacyjnym Windows lub Linux o minimalnych parametrach:

- dwu rdzeniowy procesor o architekturze ARM lub x86,
- taktowanie procesora 1 GHz,
- pamięć operacyjna 1 GB,
- złącze USB 2.0.

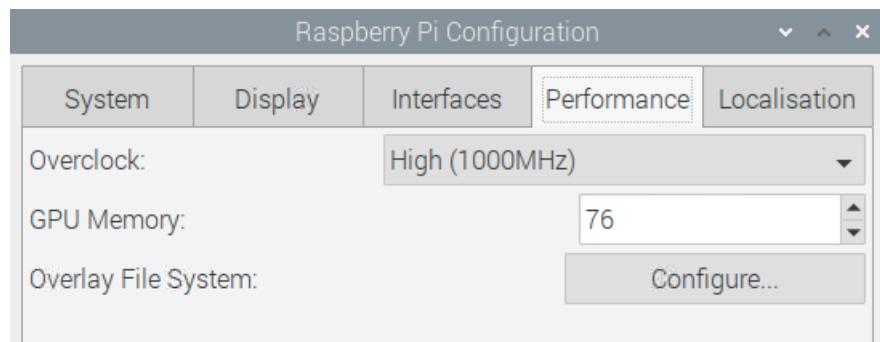


Rys. 16. Mikrokomputer Raspberry Pi 2 Model B [29].

W budowanym systemie została użyta platforma rozwojowa Raspberry Pi 2 Model B (rys. 16). Popularna „malina” posiada cztero-rdzeniowy procesor Broadcom BCM2836 o architekturze ARM Cortex A7 taktowany częstotliwością 900 MHz. Możliwe jest jednak zwiększenie częstotliwości pracy procesora do 1 GHz (rys. 17) dzięki czemu mikrokomputer spełnia wymagane parametry. Zainstalowana pamięć operacyjna to LPDDR2 o pojemności 1 GB. Na wyposażeniu znajduje się również:

- cztery porty USB 2.0,
- gniazdo Ethernet RJ45,
- wyjście sygnału obrazowego HDMI 1.4,

- gniazdo jack 3,5 mm jako wyjście sygnału analogowego audio,
- złącze uniwersalnych wejść wyjść cyfrowych procesora,
- złącze kamery oraz złącze wyświetlacza,
- gniazdo micro USB do zasilania,
- gniazdo karty micro SD będącej pamięcią nieulotną.



Rys. 17. Okno konfiguracji zwiększonego taktowania procesora BCM2836.

Mikrokomputer pracuje pod kontrolą systemu operacyjnego Raspbian Buster opartego na dystrybucji Linuxa Debian (rys. 18). W projekcie komunikację bezprzewodową WiFi zapewniono poprzez zastosowanie niewielkiej karty sieciowej Mediatec MT7601E podłączonej poprzez złącze USB.

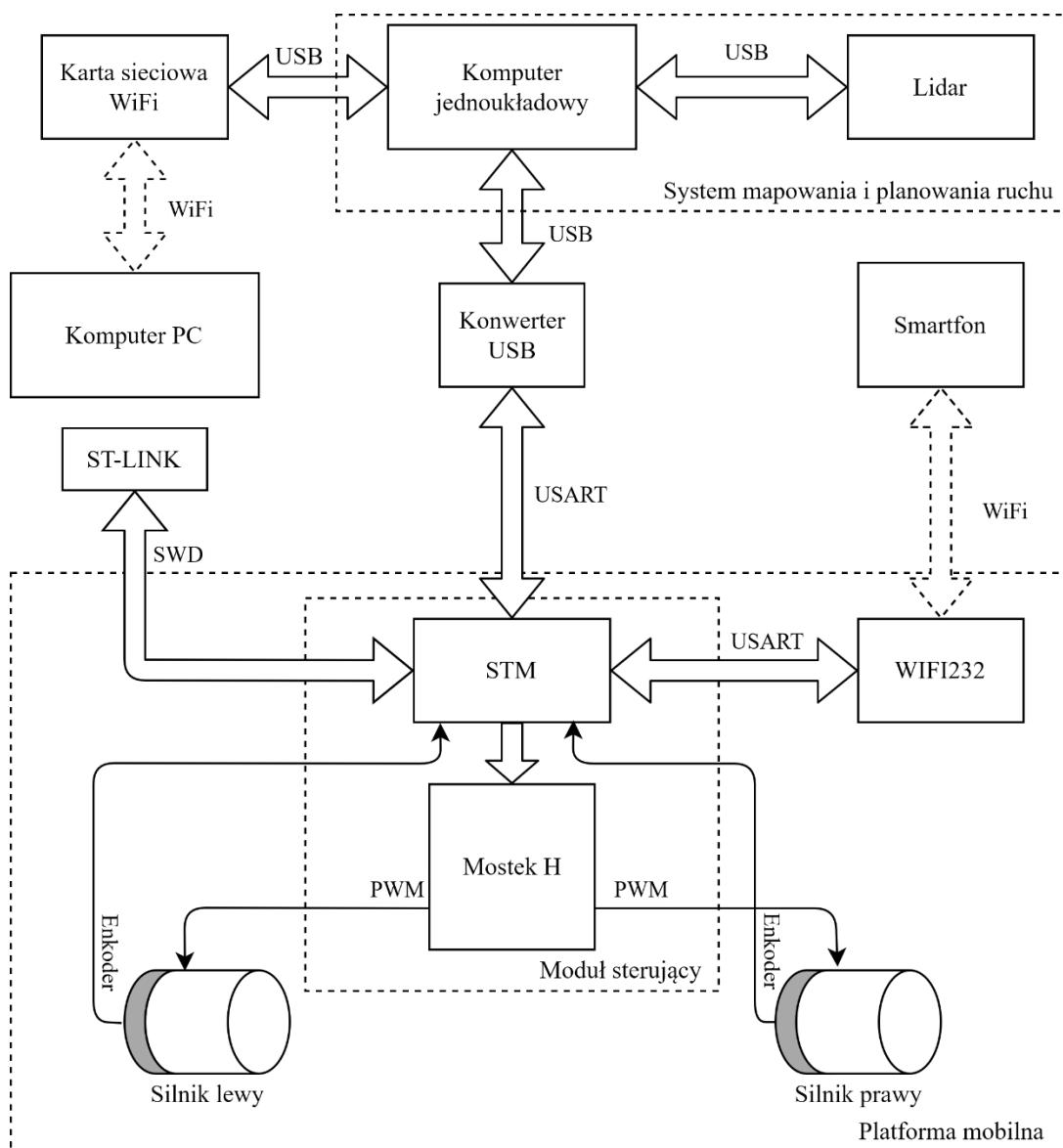
```
pi@raspberrypi:~ $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"
NAME="Raspbian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
pi@raspberrypi:~ $
```

Rys. 18. Zrzut ekranu z szczegółowymi informacjami na temat systemu operacyjnego.

3. PROJEKT SYSTEMU

3.1. KONCEPCJA BUDOWY I DZIAŁANIA SYSTEMU

Projektowany system mapowania otoczenia i planowania ruchu robota w połączeniu z platformą mobilną tworzy niewielki pojazd autonomiczny. Oprogramowanie podzespołów wchodzących w jego skład powinno pozwalać na poruszanie się w nieznanym a priori otoczeniu do wyznaczonego punktu docelowego uwzględniając występujące przeszkody. Zakłada się również możliwość sterowania urządzeniem przy użyciu teleoperatora na odległość do kilkunastu metrów.



Rys. 19. Schemat ideowy projektu.

Koncepcję budowy robota wraz z zaznaczonymi przepływami sygnałów pomiędzy modułami przedstawiono powyżej (rys. 19). Działanie platformy mobilnej jest niezależne od systemu mapowania i planowania ruchu. Dzięki zastosowaniu modułu komunikacji bezprzewodowej, możliwe jest sterowanie nią w trybie teleoperatora przy użyciu smartfona z przygotowaną aplikacją mobilną, łączącą się przy pomocy sieci WiFi. Komputer jednoukładowy otrzymując dane z skanera laserowego, tworzy w swej pamięci dwuwymiarową mapę otoczenia. Poprzez połączenie pulpu zdalnego, przez sieć bezprzewodową, możliwa jest kontrola nad pracą systemu. Z komputera PC operator ma możliwość wprowadzenia parametrów programu działającego na mikrokomputerze. Ustala on m. in. wielkość mapy, rozmiar jej siatki oraz ustala punkt docelowy w przypadku planowania trasy ruchu pojazdu autonomicznego. Wygenerowane punkty trasy są przesyłane do platformy przy użyciu portu szeregowego. Z poziomu aplikacji możliwe jest także zatrzymanie pracy autonomicznej i przejście do trybu teleoperatora. Podgląd aktualnego stanu układu sterującego podwoziem możliwe jest poprzez komputer z aplikacją STMStudio łączący się z mikrokontrolerem przy użyciu programatora ST-Link.

3.2. OPROGRAMOWANIE PLATFORMY MOBILNEJ

Platforma mobilna w projektowanym systemie jest niezależnym urządzeniem. Na podstawie otrzymanych danych od modułu komunikacyjnego WiFi232 (w przypadku trybu teleoperatora) oraz komputera jednoukładowego (tryb autonomiczny) moduł sterujący jej generuje sygnały sterujące pracę silników. Informacja zwrotna o wykonanej przez nie liczbie obrotów jest przetwarzana na prędkość oraz położenie robota.

Program realizujący wyżej wymienione zadania wykonywany jest w mikrokontrolerze STM32F407VGT6 umieszczonym na module sterującym. Oparty jest on o 32-bitowy rdzeń ARM Cortex M4 o maksymalnej częstotliwości pracy 168 Mhz. Na wyposażeniu mikrokontrolera znajdziemy również:

- 1 MB pamięci programu, 192 kB pamięci operacyjnej,
- 2 liczniki 32-bitowe oraz 12 liczników 16-bitowych,
- 3 x SPI, 3 x I2C, 4 x USART, 2 x USB, 2 x CAN,
- 82 wejścia-wyjścia cyfrowe ogólnego przeznaczenia, z czego 16 może służyć jako wejścia analogowe a 2 jako wyjścia analogowe,
- 3 przetworniki analogowo-cyfrowe o rozdzielczości 12-bitów.

Do programowania mikrokontrolera wybrano środowisko programistyczne Keil µVision 5. Umożliwia ono pisanie kodu w języku C przy użyciu wygodnego edytora kodu wykrywającego na bieżąco błędy składni. Zawiera ono również dodatkowe narzędzie debugger ułatwiający wykrywanie i eliminowanie błędów algorytmów.

Poniżej przedstawione zostaną najważniejsze fragmenty oprogramowania platformy mobilnej umożliwiające jej ruch przy sterowaniu w trybie teleoperatora oraz poruszanie się w trybie autonomicznym. Algorytm bazuje na rozwiązaniach robota mobilnego IMPULS II budowanego przez studentów i doktorantów w ramach koła naukowego Impuls działającego na Politechnice Świętokrzyskiej.

3.2.1. Odbiór i nadawanie danych

Dane do sterowania platformą mobilną są przekazywane przez dwa kanały komunikacyjne wykorzystujące transmisję szeregową USART (*universal synchronous asynchronous receiver-transmitter*). Jeden z nich (USART1) realizuje połączenie z modułem bezprzewodowym, który przesyła dane odebrane z smartfona. Drugi (USART6) podłączony jest do konwertera portu szeregowego USB, który to używany jest przez komputer jednoukładowy. Przez oba kanały wysyłana jest telemetria informująca o położeniu robota wyznaczonym przez odometrię.

```
void COM_Conf(void) {
    // ----- USART 1 ----- WIFI -----
    DMA2_Stream2->PAR = (uint32_t) &USART1->DR;
    DMA2_Stream2->M0AR = (uint32_t) pC->Com.wifi_bufread;
    DMA2_Stream2->NDTR = (uint16_t) BUF_LEN;
    DMA2_Stream2->CR |= DMA_SxCR_PL | DMA_SxCR_MINC | DMA_SxCR_CHSEL_2
        | DMA_SxCR_EN;
    DMA2_Stream7->PAR = (uint32_t) &USART1->DR;
    DMA2_Stream7->M0AR = (uint32_t) pC->Com.wifi_bufwrite;
    DMA2_Stream7->NDTR = (uint16_t) BUF_LEN;
    DMA2_Stream7->CR |= DMA_SxCR_PL | DMA_SxCR_MINC | DMA_SxCR_DIR_0
        | DMA_SxCR_CHSEL_2;
    GPIOA->MODER |= GPIO_MODE9_1 | GPIO_MODE10_1;
    GPIOA->PUPDR |= GPIO_PUPDR_PUPD9_0 | GPIO_PUPDR_PUPD10_0;
    GPIOA->AFR[1] |= 0x00000770;
    USART1->BRR = 84000000 / 115200;
    USART1->CR3 |= USART_CR3_DMAR | USART_CR3_DMAT;
    USART1->CR1 |= USART_CR1_RE | USART_CR1_TE | USART_CR1_IDLEIE
        | USART_CR1_UE;
    NVIC_EnableIRQ(USART1_IRQn);

    // ----- USART 6 ----- SBC -----
    DMA2_Stream1->PAR = (uint32_t) &USART6->DR;
    DMA2_Stream1->M0AR = (uint32_t) pC->Com.sbc_bufread;
    DMA2_Stream1->NDTR = (uint16_t) BUF_LEN;
```

```

DMA2_Stream1->CR |= DMA_SxCR_PL | DMA_SxCR_MINC | DMA_SxCR_CIRC
    | DMA_SxCR_CHSEL_2 | DMA_SxCR_CHSEL_0 | DMA_SxCR_EN;
DMA2_Stream6->PAR = (uint32_t) &USART6->DR;
DMA2_Stream6->M0AR = (uint32_t) pC->Com.sbc_bufwrite;
DMA2_Stream6->NDTR = (uint16_t) BUF_LEN;
DMA2_Stream6->CR |= DMA_SxCR_MINC | DMA_SxCR_CHSEL_2 | DMA_SxCR_CHSEL_0
    | DMA_SxCR_DIR_0;
GPIOC->MODER |= GPIO_MODER_MODER6_1 | GPIO_MODER_MODER7_1;
GPIOC->PUPDR |= GPIO_PUPDR_PUPDR6_0 | GPIO_PUPDR_PUPDR7_0;
GPIOC->AFR[0] |= 0x88000000;
USART6->BRR = 84000000 / 115200;
USART6->CR3 |= USART_CR3_DMAR | USART_CR3_DMAT;
USART6->CR1 |= USART_CR1_TE | USART_CR1_RE | USART_CR1 UE
    | USART_CR1_IDLEIE;
NVIC_EnableIRQ(USART6 IRQn);

TIM6->PSC = 168 - 1;
TIM6->ARR = 60000 - 1;
TIM6->DIER |= TIM_DIER_UIE;
TIM6->CR1 |= TIM_CR1_CEN;
NVIC_EnableIRQ(TIM6_DAC IRQn);
}

```

Lis. 1. Konfiguracja interfejsów komunikacyjnych USART.

Powyższy fragment kodu (lis. 1) odpowiada za konfigurację kanałów komunikacyjnych poprzez odpowiednie ustawienie rejestrów interfejsów USART. Prędkość transmisji danych została w obu przypadkach ustalona na 115200 bitów na sekundę. W konfiguracji wykorzystano też mechanizm DMA (*Direct Memory Access*) odpowiedzialny za przesył danych wewnątrz mikroprocesora bez użycia do tego celu rdzenia. Zadaniem DMA jest tutaj przeniesienie danych z rejestrów odbiorczych interfejsów USART do wskazanych buforów w pamięci RAM. Skonfigurowano także licznik TIM6 do generowania przerwania co 60 ms w trakcie obsługi którego nadawane są dane telemetryczne. Dane przesyłane są w postaci ustalonych pakietów a ich analizowanie rozpoczyna się po wykryciu bezczynności kanału komunikacyjnego co oznacza zakończoną transmisję (lis. 2).

```

void USART1_IRQHandler(void)
{
    if((USART1->SR & USART_SR_IDLE) != RESET){
        char c = USART1->DR;
        COM_ReadFromWifi();
    }
}
void USART6_IRQHandler(void){
    if ((USART6->SR & USART_SR_IDLE) != RESET){
        char c = USART6->DR;
        COM_ReadFromSBC();
    }
}

```

```

static void COM_ReadFromWifi(void) {
    uint8_t *buf = pC->Com.wifi_bufread;
    if (buf[0] == FrameType_Header) {
        if (buf[1] == FrameType_Drive)
            COM_GetDriveFrame(buf);
        else if (buf[1] == FrameType_CmdOnly)
            COM_GetCmdOnly(buf);
    }
    DMA2_Stream2->CR &= ~DMA_SxCR_EN;
    DMA2->LIFCR |= DMA_LIFCR_CTCIF2;
    DMA2_Stream2->CR |= DMA_SxCR_EN;
}

static void COM_ReadFromSBC(void){
    uint8_t *buf = pC->Com.sbc_bufread;
    if (buf[0] == FrameType_Header) {
        if (buf[1] == FrameType_NewPathPoints)
            COM_GetNewPathPoints(buf);
        else if (buf[1] == FrameType_CmdOnly)
            COM_GetCmdOnly(buf);
        else if (buf[1] == FrameType_SetPosition)
            COM_SetPosition(buf);
    }
    DMA2_Stream1->CR &= ~DMA_SxCR_EN;
    DMA2->LIFCR |= DMA_LIFCR_CTCIF1;
    DMA2_Stream1->CR |= DMA_SxCR_EN;
}

```

Lis. 2. Odbiór danych i interpretacja typu odebranego pakietu danych.

Pakiety przesyłane do robota jak i te przez niego wysyłane są budowane według określonego schematu (tabela 4) oraz zawierają dwa bajty sumy kontrolnej obliczanej algorytmem CRC wykorzystującym wielomian 16-bitowy. Podczas analizy odebranych danych w pierwszej kolejności sprawdzany jest znak początku ramki (0x9B). Jeżeli znak ten jest poprawny interpretowany jest typ ramki oraz wywoływana jest odpowiednia dla danego pakietu funkcja zapisująca odebrane dane do odpowiednich miejsc w pamięci operacyjnej. W przypadku gdy suma kontrolna obliczona z otrzymanego pakietu nie zgadza się z tą w nim zawartą dane zostają odrzucone.

Tabela 4

Schemat tworzenia pakietów danych

Nr bajtu	1	2	3 – n-2	n-1	n
Zawartość	Znak początku pakietu	Numer typu pakietu	Dane	Straszy bajt sumy kontrolnej	Młodszy bajt sumy kontrolnej

W trybie teleoperatora sterowanie odbywa się za pomocą dwóch 8-bitowych wartości. Prędkości liniowej oraz prędkości obrotu wokół środka robota. Są one wskalowane w zakresie od -100 do 100 co można interpretować jako procent prędkości maksymalnej robota. Przesłanie wartości 50 w przypadku gdy maksymalna prędkość robota to 0,3 m/s oznacza, że robot będzie poruszał się z prędkością 0,15 m/s. Dodatkowo przesyłany jest bajt z informacją o trybie pracy robota lub o wykonaniu zaprogramowanej komendy. Pakiet danych z tymi wartościami składa się z 11 bajtów a funkcję interpretującą zamieszczono poniżej (lis. 3).

```
static void COM_GetDriveFrame(uint8_t *buf) {
    uint16_t crc1 = crc16(buf, 9);
    uint16_t crc2 = ((uint16_t)(buf[9] << 8) + (uint16_t)buf[10]);
    if (crc1 == crc2) {
        int8_t frontspeed = (int8_t)buf[2];
        int8_t dirspeed = (int8_t)buf[3];
        pC->Com.cmdIn = (eCmdName)buf[8];
        COM_CommandIn();
        frontspeed = frontspeed * pC->Mot.speedrefmax[0] / 100.0f;
        dirspeed = dirspeed * pC->Mot.speedrefmax[0] / 100.0f;
        float frontcoef = (-0.005) * abs(dirspeed) + 1;
        float dircoef = (-0.005) * abs(frontspeed) + 1;
        pC->Mot.frontspeed = frontspeed * frontcoef;
        pC->Mot.dirspeed = dirspeed * dircoef;
        LED2_TOG;
        pC->Status.hostcomtick = 0;
    }
}
```

Lis. 3. Funkcja interpretująca pakiet danych dla trybu teleoperatora.

W trybie autonomicznym robot porusza się po wskazanych punktach znajdujących się na trasie przejazdu. Ostatni z tych punktów jest punktem docelowym ruchu robota. Rozmiar pakietu danych zależy od ilości przesyłanych punktów. Po otrzymaniu poprawnego pakietu następuje wyczyszczenie punktów trasy znajdujących się już w pamięci robota. Następnie w pętli wykonywane jest zapisanie współrzędnych punktów nowej trasy aż do ostatniego przesłanego punktu, który staje się punktem docelowym trasy. Współrzędne punktów są podzielone na 4 bajty przesyłane od najstarszego do najmłodszego (lis. 4).

```
static void COM_GetNewPathPoints(uint8_t* buf)
{
    uint8_t numOfPoints = buf[2];
    uint16_t numOfBytes = 8*numOfPoints+3;
    uint16_t crc1 = crc16(buf,numOfBytes);
    uint16_t crc2 = ((uint16_t)buf[numOfBytes]<<8)
                    +((uint16_t)buf[numOfBytes+1]);
    if (crc1 == crc2){
        for(uint8_t i=0;i<POINTS_MAX;i++){
            pC->Auto.PathPoints[i].x_pos = 0;
```

```

        pC->Auto.PathPoints[i].y_pos = 0;
        pC->Auto.PathPoints[i].disttopoint = 0;
        pC->Auto.PathPoints[i].yaw_diff = 0;
        pC->Auto.PathPoints[i].yaw_ref = 0;
        pC->Auto.PathPoints[i].type = Temppoint;
        pC->Auto.PathPoints[i].status = Waitpoint;
        pC->Auto.PathPoints[i].statustemp = Waitpoint;
    }
    pC->Auto.numOfPoints = 0;
    for(uint8_t i=0; i<numOfPoints; i++){
        double x_pos = (double)((int32_t)(((int32_t)buf[8*i+3+0]<<24)
+ ((int32_t)buf[8*i+3+1]<<16) + ((int32_t)buf[8*i+3+2]<<8) +
((int32_t)buf[8*i+3+3]<<0)) / 1000.0;
        double y_pos = (double)((int32_t)(((int32_t)buf[8*i+3+4]<<24)
+ ((int32_t)buf[8*i+3+5]<<16) + ((int32_t)buf[8*i+3+6]<<8) +
((int32_t)buf[8*i+3+7]<<0)) / 1000.0;

        pC->Auto.PathPoints[i].x_pos = x_pos;
        pC->Auto.PathPoints[i].y_pos = y_pos;
        pC->Auto.PathPoints[i].type = Temppoint;
        pC->Auto.PathPoints[i].status = DriveToPoint;
        pC->Auto.PathPoints[i].statustemp =DriveToPoint;
    }
    pC->Auto.numCurrentPoint = 0;
    pC->Auto.CurrentPoint =
        pC->Auto.PathPoints[pC->Auto.numCurrentPoint];
    pC->Auto.numOfPoints = numOfPoints;
    pC->Auto.PathPoints[numOfPoints-1].type = EndPoint;
}
}

```

Lis. 4. Funkcja interpretująca pakiet z danymi punktów trasy.

Robot w trakcie pracy wysyła zarówno do mikrokomputera jak i smartfona informację o swoim aktualnym przemieszczaniu i kącie względem pozycji od uruchomienia oraz aktualnej prędkości z jaką się porusza. Podczas przygotowywania pakietu danych w pierwszej kolejności zostaje wyczyszczony bufor nadawczy a następnie do niego kopowane są aktualne wartości. Po przygotowaniu zostaje uruchomiony mechanizm DMA, który podczas tworzenia transmisji przesyła dane z buforu do interfejsu komunikacyjnego (lis. 5).

```

static void COM_PrepateTelemetryFrame(uint8_t* buf){
    ClearStr(buf, BUF_LEN);
    int32_t speed = pC->Mot.roverspeed * 1000.0;
    int32_t angle = pC->Mot.azimuth * 1000.0;
    int32_t posx = pC->Mot.posx * 1000.0;
    int32_t posy = pC->Mot.posy * 1000.0;
    buf[0] = 155;
    buf[1] = 156;
    buf[2] = speed>>24;
    buf[3] = speed>>16;
    buf[4] = speed>>8;
}

```

```

        buf[5] = speed>>0;
        buf[6] = angle>>24;
        buf[7] = angle>>16;
        buf[8] = angle>>8;
        buf[9] = angle>>0;
        buf[10] = posx>>24;
        buf[11] = posx>>16;
        buf[12] = posx>>8;
        buf[13] = posx>>0;
        buf[14] = posy>>24;
        buf[15] = posy>>16;
        buf[16] = posy>>8;
        buf[17] = posy>>0;
        uint16_t crc = crc16(buf, 18);
        buf[18] = crc>>8;
        buf[19] = crc>>0;
    }

    static void COM_SendToWifi(void){
        COM_PrepareTelemetryFrame(pC->Com.wifi_bufwrite);
        DMA2_Stream7->CR &= ~DMA_SxCR_EN;
        DMA2->HIFCR |= DMA_HIFCR_CTCIF7;
        DMA2_Stream7->NDTR = (uint16_t)20;
        DMA2_Stream7->CR |= DMA_SxCR_EN;
    }

    static void COM_SendToSBC(void){
        COM_PrepareTelemetryFrame(pC->Com.sbc_bufwrite);
        DMA2_Stream6->CR &= ~DMA_SxCR_EN;
        DMA2->HIFCR |= DMA_HIFCR_CTCIF6;
        DMA2_Stream6->NDTR = (uint16_t)20;
        DMA2_Stream6->CR |= DMA_SxCR_EN;
    }

    void TIM6_DAC_IRQHandler(void){
        if((TIM6->SR & TIM_SR_UIF) != RESET){
            LED3_ON;
            COM_SendToWifi();
            COM_SendToSBC();
            LED3_OFF;
            TIM6->SR &= ~TIM_SR_UIF;
        }
    }
}

```

Lis. 5. Wysyłanie danych telemetrycznych.

3.2.2. Zmiana trybów pracy

Zmiana trybu pracy możliwa jest dzięki przesyłaniu odpowiednich komend wraz z pakietami danych. Umożliwiają one oprócz wyboru trybu teleoperatora lub autonomicznego również możliwość zatrzymania jazdy automatycznej oraz jej wznowienie. Umożliwiają także przejście do kolejnego punktu trasy oraz wyzerowanie pozycji robota (lis. 6). Główna funkcja kontrolująca pracę robota (Mot_Act) wykonywana jest w funkcji obsługi przerwania licznika TIM7, który przepelnia się co 10 ms. W zależności od aktualnie wybranego trybu wywoływane

są odpowiednie funkcje. Podczas sterowania ręcznego wartość prędkości zadanej znajduje się po odbiorze danych w pamięci robota. Wartości zadane porównywane są z aktualnymi i wyznaczony zostaje uchyb sterowania. Na jego podstawie cyfrowy regulator PID wyznacza wartość wypełnienia sygnału PWM trafiającego do silników.

```

static void COM_CommandIn(void){
    switch(pC->Com.cmdIn){
        case Cmd_Wifi: pC->Auto.WorkMode = Mode_Wifi; break;
        case Cmd_AutoStart: pC->Auto.WorkMode = Mode_Auto; break;
        case Cmd_AutoPause: Mot_AutoPause(); break;
        case Cmd_AutoContinue: Mot_AutoContinue(); break;
        case Cmd_AutoNextPoint: Mot_AutoNextPoint(); break;
        case Cmd_ResetPos: COM_ResetPos(); break;
        default: break;
    }
    pC->Com.cmdIn = Cmd_Null;
}

static void Mot_Act(void){
    Mot_ReadPos();
    Mot_ReadStatus();
    if (pC->Auto.WorkMode == Mode_Wifi){
        if(pC->Status.work == On){
            Mot_SpeedRef();
            Mot_PidSpeedOn();
        }
        else{
            Mot_Stop();
        }
    }
    else if (pC->Auto.WorkMode == Mode_Auto){
        if(pC->Auto.CurrentPoint.status == DriveToPoint){
            Mot_AutoDrive(pC->Auto.CurrentPoint, 40.0, 30.0);
            Mot_SpeedRef();
            Mot_PidSpeedOn();
        }
        else {
            Mot_Stop();
        }
    }
    Mot_DirPwmAct();
}

void TIM7_IRQHandler(void){
    if((TIM7->SR & TIM_SR UIF) != RESET){
        Mot_Act();
        TIM7->SR &= ~TIM_SR UIF;
    }
}

```

Lis. 6. Zmiana trybów pracy robota.

3.2.3. Odometria

Realizacja trybu jazdy autonomicznej opiera się na znajomości położenia i orientacji robota w lokalnym układzie współrzędnych. Wyznaczenie tych wartości realizowane jest na podstawie odczytów liczby impulsów z enkoderów inkrementalnych zamocowanych na silnikach robota. W tym przypadku interpretacja sygnałów kwadraturowych generowanych przez czujniki odbywa się w sposób sprzętowy dzięki konfiguracji liczników mikrokontrolera do pracy w wyspecjalizowanym do tego celu trybie. Wyznaczając liczbę impulsów na obrót koła, oraz wyznaczając jego obwód, jesteśmy w stanie wyznaczyć przebytą drogę. Mechanizm ten wymagał kalibracji w celu otrzymania prawidłowych wartości przemieszczenia, kąta oraz prędkości (lis. 7).

```
static void Mot_ReadPos(void){
    int32_t r=0;
    for(uint8_t i=0;i<MOTMAX;i++){
        pC->Mot.poslast[i] = pC->Mot.pos[i];
        pC->Mot.pos[i] = (int16_t)pC->Mot.tims[i]->CNT;
        r = pC->Mot.pos[i] - pC->Mot.poslast[i];
        if(r < (-TIMMAX/2))
            pC->Mot.fullturn[i]++;
        else if(r > (TIMMAX/2))
            pC->Mot.fullturn[i]--;
        pC->Mot.postotallast[i] = pC->Mot.postotal[i];
        pC->Mot.postotal[i] = pC->Mot.fullturn[i] * TIMMAX +
            pC->Mot.pos[i];
        pC->Mot.speed[i] = (double)(pC->Mot.postotal[i] -
            pC->Mot.postotallast[i]) / (double)PIDTIME / 133.14;
        pC->Mot.distance[i] = (double)pC->Mot.postotal[i] / 133.14;
    }
    pC->Mot.roverspeed = (pC->Mot.speed[0] + pC->Mot.speed[1]) / 2.0;
    pC->Mot.azimuth = (pC->Mot.distance[0] - pC->Mot.distance[1]) / -1.2 *
        3600/2673;
    pC->Mot.posx += sin(pC->Mot.azimuth * TO_RADIANS) * pC->Mot.roverspeed *
        (double)PIDTIME;
    pC->Mot.posy += cos(pC->Mot.azimuth * TO_RADIANS) * pC->Mot.roverspeed *
        (double)PIDTIME;
}
```

Lis. 7. Funkcja wyznaczająca położenie i orientację robota.

3.2.4. Tryb autonomiczny

Tryb jazdy autonomicznej opiera się na tablicy punktów trasy robota załadowanych do pamięci oraz na podstawie aktualnego położenia i orientacji robota w przyjętym układzie współrzędnych. Gdy robot jest w trybie jazdy autonomicznej pobiera on pierwszy punkt z trasy i przypisuje do aktualnego celu, do którego zmierza. W każdym wywołaniu funkcji jazdy

automatycznej algorytm wyznacza aktualną odległość do punktu oraz kąt względem kierunku ruchu robota. Wartości te są przetwarzane przez dwa regulatory proporcjonalne na wartości prędkości ruchu liniowego oraz obrotu robota. W przypadku wystąpienia dużej różnicy kątowej generowany jest sygnał wyłącznie do obrotu. Jeżeli różnica ta zmniejszy się poniżej wyznaczonego progu algorytm wyznacza również prędkość jazdy do przodu. Wyznaczone wartości są następnie przetwarzane w ten sam sposób jak w trybie teleoperatora.

Jeżeli odległość do aktualnego punktu będzie mniejsza od ustalonego progu, algorytm pobierze z pamięci kolejny punkt trasy (funkcja Mot_AutoNextPoint). W przypadku dotarcia do punktu docelowego robot zatrzymuje się kończąc sekwencję autonomiczną (lis. 8).

Poruszanie się w trybie automatycznym może zostać zastopowane i ponownie wznowione dzięki komendom pauzy i kontynuacji (funkcja Mot_AutoPause i Mot_AutoContinue)

```

void Mot_AutoNextPoint(void){
    if (pC->Auto.CurrentPoint.type == Tempoint){
        pC->Auto.numCurrentPoint++;
        pC->Auto.CurrentPoint =
            pC->Auto.PathPoints[pC->Auto.numCurrentPoint];
        pC->Auto.CurrentPoint.status = DriveToPoint;
    }
}
void Mot_AutoPause(void){
    if (pC->Auto.CurrentPoint.status != Waitpoint){
        pC->Auto.CurrentPoint.statustemp = pC->Auto.CurrentPoint.status;
        pC->Auto.CurrentPoint.status = Waitpoint;
    }
}
void Mot_AutoContinue(void){
    pC->Auto.CurrentPoint.status = pC->Auto.CurrentPoint.statustemp;
}

static void Mot_AutoDrive(sPathPoint to_point, double speedmax_front, double
speedmax_dir){
    LED2_TOG;
    double yawkp = 2.0, yawhyst = 10.0, distkp = 50.0, disthyst = 0.06;
    double x_pos_act = pC->Mot.posx/100.0;
    double y_pos_act = pC->Mot.posy/100.0;
    double x_pos_ref = to_point.x_pos;
    double y_pos_ref = to_point.y_pos;
    double dx = x_pos_ref-x_pos_act;
    double dy = y_pos_ref-y_pos_act;
    double dist = sqrt(pow(dx,2) + pow(dy,2));
    double yact = pC->Mot.azimuth;
    double yref=0, yer=0;
    pC->Auto.CurrentPoint.disttopoint = dist;
    if(dist > 0.04)
        yref = atan2(dx, dy);
    yref = yref * (180.0 / M_PI);
    double er = yref - yact;
    yer = er;
    if(fabs(er) <= 180.0)

```

```

        yer = er;
    else if(er > 180.0)
        yer = er - 360.0;
    else if(er < -180.0)
        yer = 360.0 - fabs(er);
    pC->Auto.CurrentPoint.yaw_diff = yer;
    if(to_point.type == Temppoint){
        if(dist > disthyst*2)
            pC->Mot.frontspeed= dist * distkp;
        else
            Mot_AutoNextPoint();
    }
    else if(to_point.type == EndPoint){
        if(dist > disthyst)
            pC->Mot.frontspeed= dist * distkp;
        else{
            pC->Auto.CurrentPoint.status = DestinationPoint;
            Mot_Stop();
        }
    }
    if(pC->Mot.frontspeed > speedmax_front)
        pC->Mot.frontspeed = speedmax_front;
    else if(pC->Mot.frontspeed < -speedmax_front)
        pC->Mot.frontspeed = -speedmax_front;
    float dirspeed = yer * yawkp;
    if(dirspeed > speedmax_dir)
        dirspeed = speedmax_dir;
    else if(dirspeed < -speedmax_dir)
        dirspeed = -speedmax_dir;
    pC->Mot.dirspeed = (int8_t)dirspeed;
    if(fabs(yer) > yahyst)
        pC->Mot.frontspeed = 0.0;
    pC->Auto.CurrentPoint.yaw_ref = pC->Mot.dirspeed;
}

```

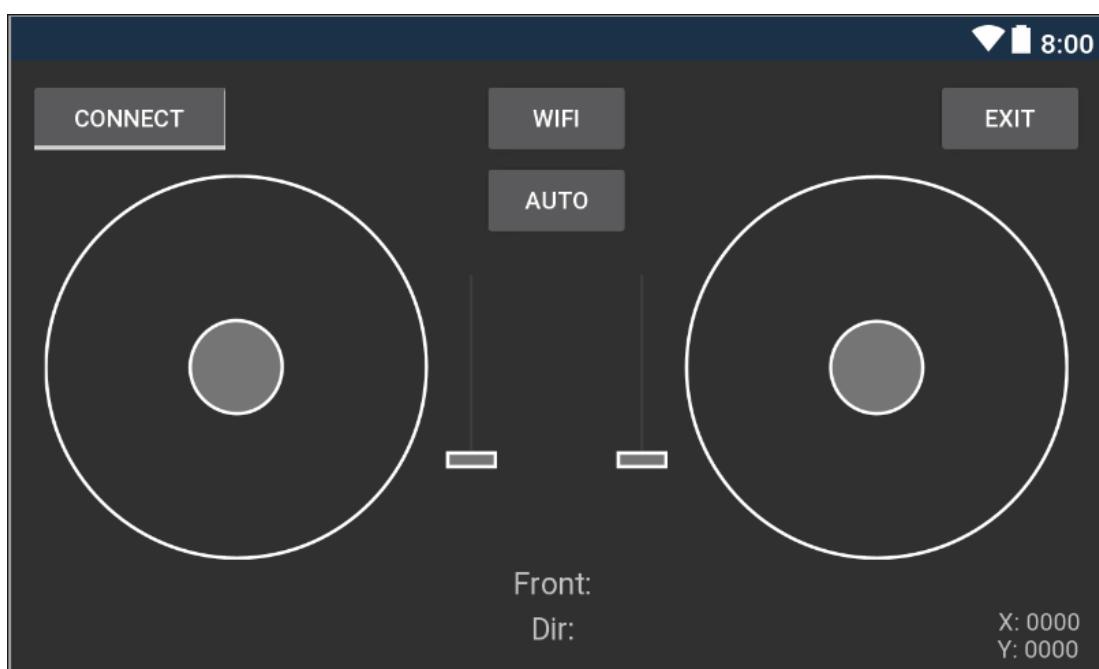
Lis. 8. Funkcje realizujące jazdę autonomiczną robota.

3.3. APLIKACJA MOBILNA DO STEROWANIA PLATFORMĄ MOBILNĄ

Sterowanie bezprzewodowe robotem w trybie teleoperatora odbywa się za pomocą smartfona z systemem Android wraz z przygotowaną aplikacją mobilną. Aplikacja ta została napisana przy użyciu platformy Xamarin wykorzystującej pakiet .Net i wchodzącej w skład narzędzi programu Visual Studio Community 2019. Środowisko to umożliwia projektowanie i wdrażanie programów przy użyciu języka C# i opisu graficznego interfejsu użytkownika językiem XAML (*Extensible Application Markup Language*). Aplikacje są kompilowane do języka pośredniego i uruchamiane przy pomocy maszyny wirtualnej Mono obsługującej natywne biblioteki systemu Android [30].

Przygotowana aplikacja posiada prosty interfejs graficzny (rys. 20). Składa się on z dwóch dotykowych joysticków, dwóch suwaków, czterech przycisków oraz czterech etykiet tekstowych.

Joysticki odpowiedzialne są za wyznaczenie prędkości jazdy robota, gdzie lewy odpowiada prędkości liniowej robota a prawy prędkości obrotu robota wokół własnej osi. Znajdujące się pomiędzy nimi suwaki ustalają maksymalne wartości jakie mogą zostać wysłane do robota. Znajdujące się tuż nad nimi przyciski umożliwiają wybór trybu pracy. Guzik w lewym górnym rogu odpowiada za połączenie się z platformą mobilną. W prawym górnym rogu znajduje się przycisk umożliwiający zamknięcie aplikacji.



Rys. 20. Interfejs graficzny aplikacji do sterowania robotem.

Połączenie z modułem komunikacyjnym WiFi232 zamocowanym na podwoziu robota odbywa się przy udziale protokołu TCP. Przycisk CONNECT dodatkowo wskazuje stan logiczny włączony bądź wyłączony. Właściwość ta jest wykorzystywana do informowania użytkownika o stanie połączenia z robotem. Po jego wcisnięciu zmieniany jest jego stan logiczny i w zależności od nowego stanu realizowana jest próba połączenia bądź następuje zerwanie połączenia z robotem mobilnym. W przypadku nieudanej próby otwarcia komunikacji TCP na ekranie zostaje wyświetlona informacja o błędzie (lis. 9).

```

WifiManager wifi = (WifiManager)GetSystemService(Context.WifiService);

btnConn.Click += async (s, e) => {
    if (btnConn.Checked == true) {
        if (!(wifi.IsEnabled)) {
            dialogWifi.Show();
            while (dialogWifi.IsShowing == true)
                await Task.Delay(100);
        }
        if (wifi.IsEnabled) {
            if (!(connected)) {
                DirText.Text = string.Format(wifi.ConnectionInfo.SSID.ToString());
                try {
                    tcpClient = new TcpClient();
                    tcpClient.ConnectAsync("10.10.100.254", 8899);
                    int dur = 0;
                    while (!(tcpClient.Connected)) {
                        if (dur % 2 == 1) {
                            btnConn.Checked = false;
                            btnConn.Text = "Connecting";
                        } else if (dur % 2 == 0) {
                            btnConn.Checked = true;
                            btnConn.Text = "Connecting";
                        }
                        if (dur >= 50) {
                            connected = false;
                            throw new Exception("No connect");
                        }
                    }
                    else
                        await Task.Delay(200);
                    dur++;
                }
                message = "Connect Succesful!";
                toast = Toast.MakeText(this, message, ToastLength.Long);
                toast.Show();
                tcpClient.NoDelay = true;
                networkStream = tcpClient.GetStream();
                connected = true;
                btnConn.Checked = true;
            }
            catch (Exception) {
                connected = false;
                dialogNoConn.Show();
                btnConn.Checked = false;
            }
        }
    }
}
if (btnConn.Checked == false) {
    if (connected) {
        networkStream.Dispose();
        networkStream.Close();
        tcpClient.Dispose();
        tcpClient.Close();
    }
    connected = false;
}};


```

Lis. 9. Realizacja połączenia TCP z modułem komunikacyjnym WiFi232.

Przesuwając joysticki następuje aktualizacja wartości w buforze nadawczym. Zakres wartości ich pracy to od -100 do 100. Jest ona skalowana do maksymalnej prędkości jaka została w danej chwili ustalona przy pomocy suwaków. Zmiana maksymalnej prędkości jest informowana na ekranie w postaci komunikatu oraz zostaje zapisana na stałe w pamięci urządzenia co umożliwia jej odczytanie przy ponownym uruchomieniu aplikacji (lis. 10).

```

FrontSeek.StopTrackingTouch += (s, e) =>{
    FrontSeek.Progress = 100;
};

FrontSeek.ProgressChanged += (s, e) =>{
byte speed_tmp = (byte)((int)((e.Progress - 100) * ((float)(MagFront / 100.0))));
FrontText.Text = string.Format("Front: {0}", speed_tmp);
connectivity.drive_data_struct[(int)Connectivity.drive_data.SPEED] = speed_tmp;
};

DirSeek.StopTrackingTouch += (s, e) =>{
    DirSeek.Progress = 100;
};

DirSeek.ProgressChanged += (s, e) =>{
byte dir_tmp = (byte)((int)((e.Progress - 100) * ((float)(MagDir / 100.0))));
DirText.Text = string.Format("Dir: {0}", dir_tmp);
connectivity.drive_data_struct[(int)Connectivity.drive_data.DIR] = dir_tmp;
};

MagDirSeek.StopTrackingTouch += (s, e) =>{
    MagDir = MagDirSeek.Progress;
    prefEditor.PutInt("MagDir", MagDir);
    prefEditor.Commit();
    message = "Prędkość skrętów: " + MagDir.ToString() + "%";
    toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
    toast.show();
};

MagFrontSeek.StopTrackingTouch += (s, e) =>{
    MagFront = MagFrontSeek.Progress;
    prefEditor.PutInt("MagFront", MagFront);
    prefEditor.Commit();
    message = "Prędkość do przodu: " + MagFront.ToString() + "%";
    toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
    toast.show();
};

```

Lis. 10. Obsługa joysticków oraz suwaków maksymalnej prędkości.

Pakiet danych wysyłany jest do robota co 100 ms. Czas ten jest odliczany przez licznik programowy. Po jego przepełnieniu przygotowywana jest ramka danych a następnie wysłana. Po zakończonej transmisji nadawczej odczytywany jest bufor odbiorczy i realizowana jest funkcja interpretująca dane telemetryczne, które następnie są aktualizowane na interfejsie graficznym (lis. 11).

```

Timer100ms = new SysTimer();
Timer100ms.Interval = 100;
Timer100ms.Enabled = true;
Timer100ms.Elapsed += delegate{
    connectivity.prepare_data();
    if (connected){
        networkStream.Write(connectivity.data_to_send, 0, 11);
        networkStream.Read(connectivity.read_buff);
        connectivity.telemetry();
        RunOnUiThread(() => {
            PosxText.Text = "X: " + connectivity.posx.ToString();
            PosyText.Text = "Y: " + connectivity.posy.ToString();
        });
    }
};

public void prepare_data() {
    drive_data_struct[(int) drive_data.HEADER] = 155;
    drive_data_struct[(int) drive_data.FRAMETYPE] = 1;
    Array.Copy(drive_data_struct, data_to_send, 11);
    UInt16 x = crc16(data_to_send, 9);
    data_to_send[(int) drive_data.CRC] = (byte)(x >> 8);
    data_to_send[(int) drive_data.CRC2] = (byte)(x);
    cmd_zero();
}

public void telemetry() {
    if (read_buff[0] == 155) {
        UInt16 crc1 = crc16(read_buff, 18);
        UInt16 crc2 = (UInt16)((read_buff[18] << 8) + read_buff[19]);
        if (crc1 == crc2) {
            speed = (((Int32)((UInt32) read_buff[2] << 24)
                + ((UInt32) read_buff[3] << 16)
                + ((UInt32) read_buff[4] << 8)
                + ((UInt32) read_buff[5] << 0))) / 1000.0;
            yaw = (((Int32)((UInt32) read_buff[6] << 24)
                + ((UInt32) read_buff[7] << 16)
                + ((UInt32) read_buff[8] << 8)
                + ((UInt32) read_buff[9] << 0))) / 1000.0;
            posx = (((Int32)((UInt32) read_buff[10] << 24)
                + ((UInt32) read_buff[11] << 16)
                + ((UInt32) read_buff[12] << 8)
                + ((UInt32) read_buff[13] << 0))) / 1000.0;
            posy = (((Int32)((UInt32) read_buff[14] << 24)
                + ((UInt32) read_buff[15] << 16)
                + ((UInt32) read_buff[16] << 8)
                + ((UInt32) read_buff[17] << 0))) / 1000.0;
        }
    }
}

```

Lis. 11. Wymiana danych z robotem mobilnym.

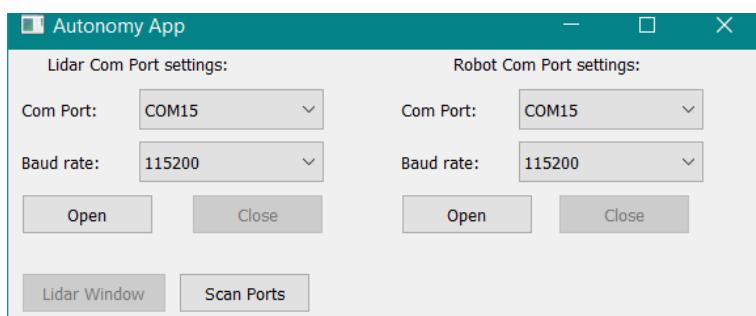
3.4. APLIKACJA KOMPUTERA JEDNOUKŁADOWEGO

Główny algorytm systemu mapowania otoczenia i planowania ruchu robota mobilnego jest uruchamiany na komputerze jednoukładowym Raspberry Pi 2 model B. Pracuje on pod kontrolą systemu operacyjnego Raspian, który jest specjalnie przygotowaną dystrybucją Linuxa. Procesor mikrokomputera zbudowany jest w architekturze RISC (zredukowana liczba instrukcji) różniącej się w znacznym stopniu od CISC (kompletna liczba instrukcji). Skompilowany program na komputerze PC nie zostanie uruchomiony na procesorze wykorzystującym rdzenie ARM. Według założeń projektu możliwe jest użycie dowolnego komputera więc użyte narzędzia programistyczne powinny pozwalać komplikację zarówno dla systemów Linux i Windows, ale również dla architektur ARM i x86-64. Na takie rozwiązanie pozwala pakiet Qt wchodzących w skład środowiska programistycznego Qt Creator. Umożliwia ono programowanie w języku C++. Program napisany przy użyciu tych narzędzi może być skompilowany bez zmian w kodzie dla obu systemów operacyjnych i architektur. Pełen kod programu został zawarty w załącznikach, w dalszej części zostały opisane najważniejsze jego funkcje.

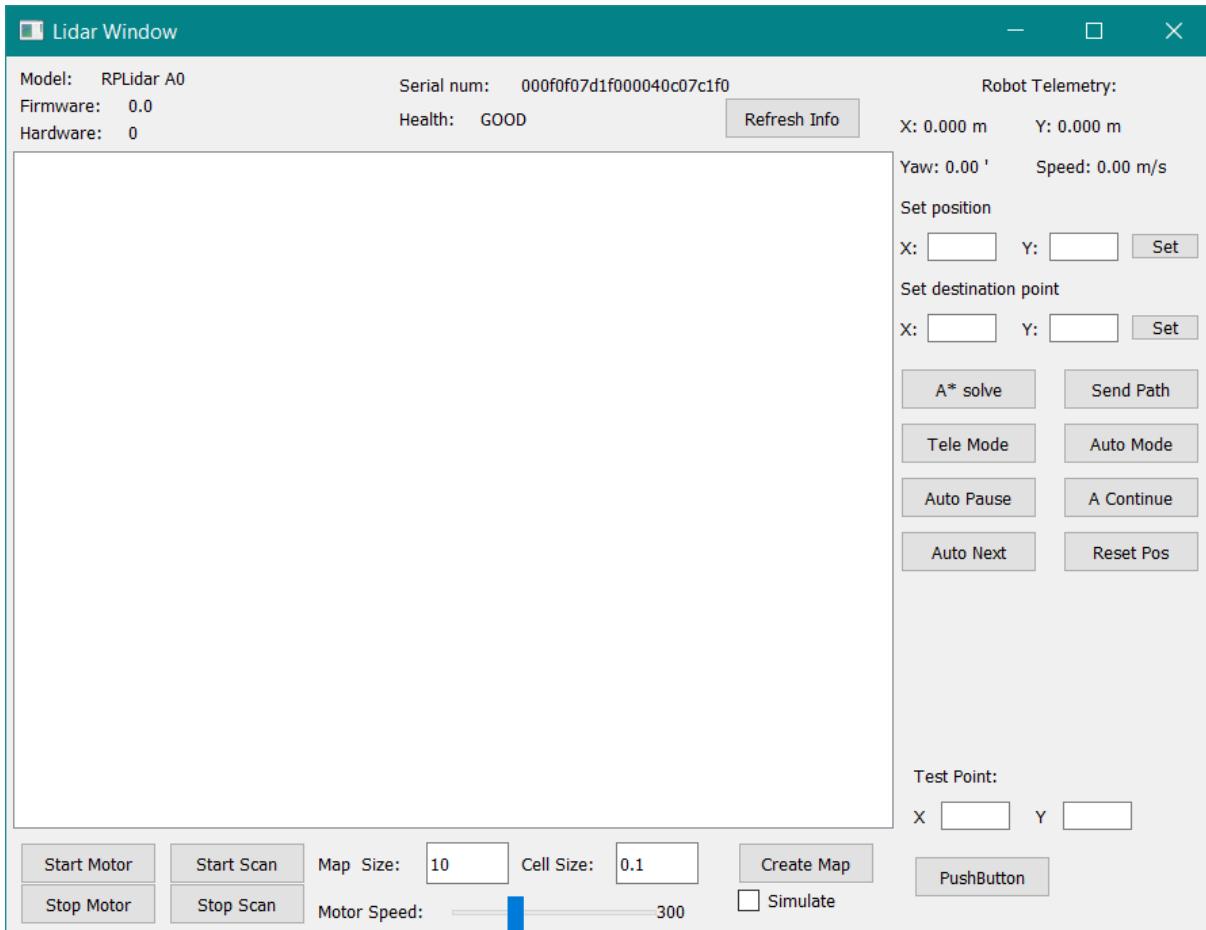
3.4.1. Interfejs graficzny

Na interfejs graficzny użytkownika składają się dwa okna. Pierwsze z nich widoczne tuż po uruchomieniu programu pozwala na wskazanie portów szeregowych do których podłączony jest LIDAR oraz konwerter USB/USART realizujący połączenie z platformą mobilną (rys. 21). Jeżeli któreś z urządzeń zostało podłączone po uruchomieniu programu nie będzie ono widoczne w polu wyboru. Kliknięcie przycisku Scan Ports umożliwia ponowne przeskanowanie aktywnych portów i umieszczenie ich w rozwijanej liście wyboru.

Otwarcie głównego okna programu (rys. 22) możliwe jest dopiero po nawiązaniu komunikacji z sensorem. Po otwarciu portu szeregowego, który jest przez niego używany, aktywowany zostaje przycisk Lidar Window wywołujący nowe okno.



Rys. 21. Okno wybory kanałów komunikacji z LIDAR-em oraz robotem.

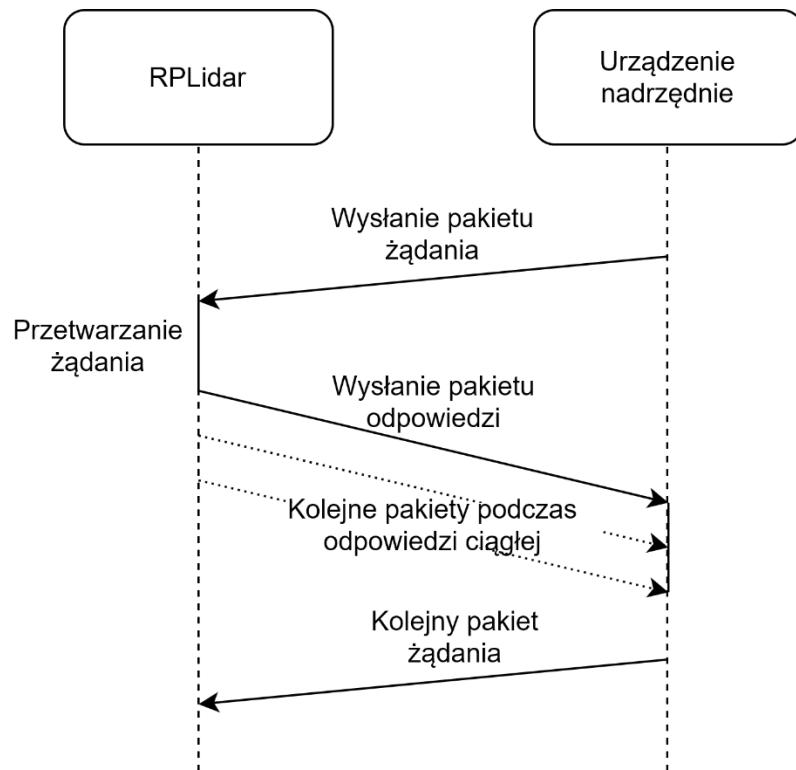


Rys. 22. Główne okno programu.

Na pierwszym planie głównego okna programu znajduje się pole, na którym wizualizowana jest mapa, generowana podczas działania algorytmu. Dodatkowo zawiera ono w górnej części etykiety tekstowe które informują o podłączonym urządzeniu oraz o aktualnej pozycji robota. W lewym dolnym rogu znajdują się przyciski oraz suwak. Są one odpowiedzialne za kontrolę silnika napędzającego głowicę LIDAR-u oraz uruchamiające i przerywające skanowanie. Tuż obok znajdują się pola tekstowe do których użytkownik wprowadza rozmiar mapy oraz wielkość pojedynczej komórki zawierającej informację. Wyświetlenie pustej mapy odbywa się poprzez wcisnięcie przycisku Create Map. Podczas gdy zaznaczony jest opcja Similate na mapie pojawiają się generowane losowo przeszkody. Po prawej od obszaru rysowania mapy znajdują się pola tekstowe które umożliwiają ustalenie pozycji robota oraz wskazanie punktu docelowego. Poniżej znajdują się natomiast przyciski odpowiedzialne na wyznaczenie trasy, przesłanie jej do robota oraz kontrole trybu autonomicznego. W prawym dolnym rogu znajduje się miejsce do wprowadzenia punktu dla przetestowania algorytmu mapującego.

3.4.2. Obsługa skanera laserowego RPLidar A2M8

Komunikacja LIDAR-u RPLidar A2M8 z urządzeniem nadzorującym, którym może być komputer lub mikrokontroler, odbywa przy udziale pakietów wysyłanych poprzez port szeregowy. Wymiana informacji odbywa się na zasadzie żądań wysyłanych do skanera, który po ich przetworzeniu przesyła pojedynczą odpowiedź lub ich ciąg (rys. 23). Początek komunikacji zawsze rozpoczynany jest przez urządzenie nadzorujące, LIDAR nie może automatycznie rozpoczęć nadawania.



Rys. 23. System żądań i odpowiedzi stosowany przez LIDAR.

Pakiet żądania budowany jest według określonego schematu (tabela 5). Pierwszy przesyłany bajt informuje o rozpoczęciu nowej transmisji (0xA5). Kolejny bajt zawiera zakodowany numer komendy. Część z komend wymagają również wysyłania dodatkowych parametrów. Jeżeli są one wymagane należy określić ich ilość oraz obliczyć sumę kontrolną z pełnej ramki danych. Producent przewidział komendy:

- STOP (0x25) – zatrzymuje trwający pomiar, powrót do stanu bezczynności,
- RESET (0x40) powoduje ponowne uruchomienie elektroniki wewnętrz LIDAR-u,
- SCAN (0x20) – rozpoczęcie skanowania trybie normalnym (4000 pomiarów na sekundę),

- EXPRESS_SCAN (0x82) – rozpoczęcie skanowania w trybie o zwiększonej liczbie pomiarów (8000 na sekundę),
- FORCE_SCAN (0x21) – wykonanie pomiaru przy zatrzymanym silniku głowicy,
- GET_INFO (0x50) – pobranie informacji o podłączonym urządzeniu,
- GET_HEALTH (0x52) – pobranie danych o poprawności działania,
- MOTOR_SPEED_CRTL (0xA8) – zmiana prędkości obrotowej głowicy pomiarowej.

W celu tworzenia pakietów żądań nadawanych do LIDAR-u zostały utworzone funkcje przyjmujące jako argumenty wywoływaną komendę oraz parametry dodatkowe (lis. 12). Efektem ich działania jest wygenerowanie odpowiedniej ramki danych oraz wysłanie ich poprzez wybrany port szeregowy.

Tabela 5

Schemat tworzenia żądań wysyłanych do skanera

Bajt	1	2	3	4 — n -1	n
			Opcjonalnie		
Zawartość	Znak początku pakietu	Komenda	Ilość bajtów parametrów	Dane parametrów (max 255 B)	Suma kontrolna

```

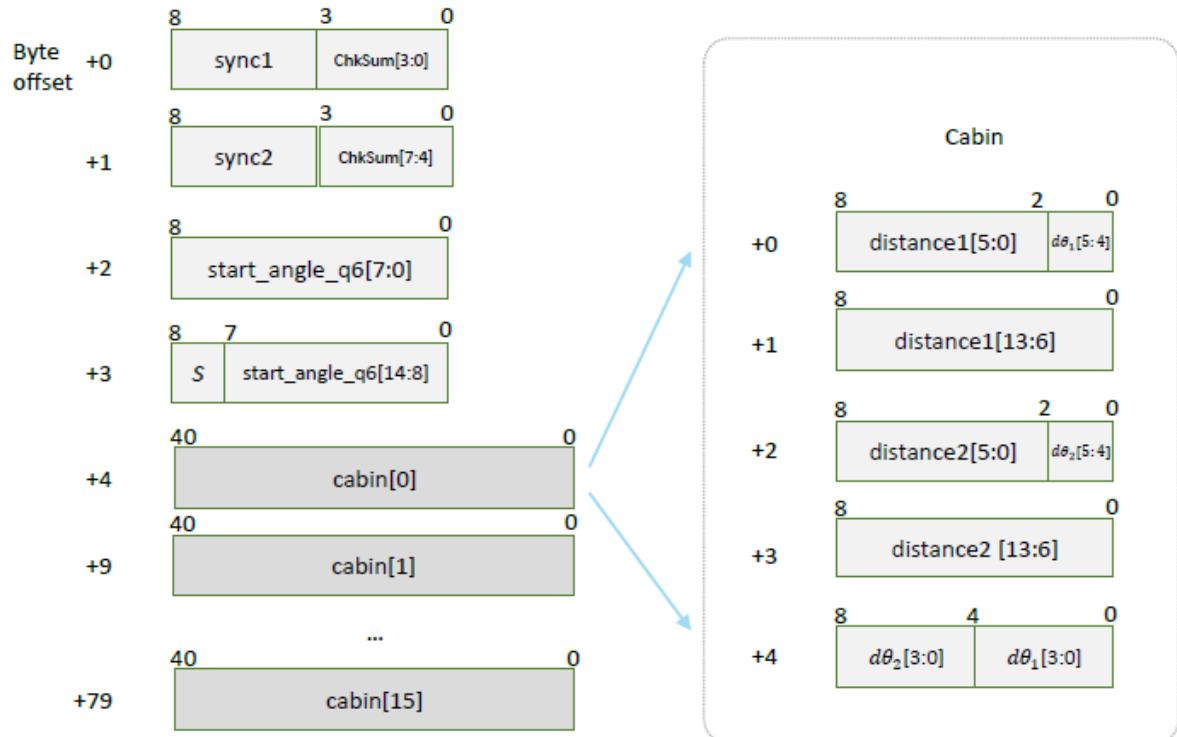
void RPLidar::sendCommand(u_int8_t cmd) {
    u_int8_t buff[2];
    buff[SYNC_FLAG] = START_FLAG;
    buff[CMD] = cmd;
    ptr_serialPort->write((char*) buff, 2);
}

void RPLidar::sendCommandPayload(u_int8_t cmd, u_int8_t *payload,u_int8_t size)
{
    u_int8_t buff[20];
    buff[0] = START_FLAG;
    buff[1] = cmd;
    buff[2] = size;
    for (int i = 0; i < size; i++) {
        buff[3 + i] = *payload;
        payload++;
    }
    u_int8_t checksum = 0;
    for (u_int8_t i = 0; i < 3 + size; i++) {
        checksum ^= buff[i];
    }
    buff[3 + size] = checksum;
    ptr_serialPort->write((char*) buff, 4 + size);
}

```

Lis. 12. Funkcje wysyłające żądanie do LIDAR-u.

Rozpoczęcie skanowania możliwe jest po ustabilizowaniu się prędkości obrotowej głowicy LIDAR-u oraz odczytaniu informacji o poprawnym działaniu. Jeżeli LIDAR zwrócił informacje o braku błędów możliwe jest wysłanie żądania o rozpoczęciu pomiarów. W zaproponowanym rozwiążaniu wykorzystano tryb skanowania o zwiększonej liczbie pomiarów, jest to tryb pracy preferowany przez producenta urządzenia. Podczas skanowania wysyłane są pakiety odpowiedzi zawierające zakodowane wyniki serii 32 pomiarów (rys. 24).



Rys. 24. Schemat pakietu odpowiedzi LIDAR-u pracującego w trybie *express scan* [28].

Wynik pomiaru LIDAR-owego składa się z odległości oraz kąta pod jakim został on wykonany (rys. 15). W otrzymanym pakuiecie znajduje się 16 grup zawierających po dwa pomiary. Zmierzony dystans przesyłany jest na 14 bitach. Drugim parametrem pomiaru jest współczynnik kompensacyjny kąta pomiaru ($d\theta_i$). W celu wyznaczenia kąta θ_k -tego pomiaru z należy skorzystać z następującej zależności (2):

$$\theta_k = \theta_i + \frac{\theta_i - \theta_{i-1}}{32} \cdot k - d\theta_k \quad (2)$$

gdzie: θ_k - kąt pod jakim został wykonany k -ty pomiar serii,

θ_i - kąt pod jakim został wykonany pierwszy pomiar i -tej serii,

$d\theta_k$ - poprawka wyznaczenia kąta k -tego pomiaru.

W aplikacji została przygotowana funkcja (lis. 13) realizująca odbiór pakietu z danymi pomiarowymi oraz ich zdekodowanie. Wyznaczone punkty pomiarowe zostają zapisane w 32 elementowej tablicy, gdzie każda z komórek zawiera dane jednego pomiaru. Dane te zapisywane są w globalnej strukturze danych co umożliwia ich łatwy dostęp z każdego miejsca programu. Po przetworzeniu pakietu wyemitowany zostaje sygnał, który wykorzystywany jest podczas aktualizowania mapy.

```

void RPLidar::getExpressData() {
    int bytes_to_read = ptr_serialPort->bytesAvailable();
    QByteArray arr = ptr_serialPort->readAll();
    u_int8_t multi_packed = bytes_to_read / 84;
    for (u_int8_t j = 0; j < multi_packed; j++) {
        if (((arr[0 + j * 84] >> 4) == 0xA)&& ((arr[1 + j * 84] >> 4) == 0x5)) {
            pData->data_express.start_angle = float((u_int16_t)(  

                arr[2 + j * 84] + ((arr[3 + j * 84] & 0x7F) << 8)) / 64.0f;  

            pData->data_express.delta_angle = pData->data_express.start_angle  

                - pData->data_express.last_start_angle;  

            if (pData->data_express.delta_angle < 0)
                pData->data_express.delta_angle = 360.0f
                + pData->data_express.delta_angle;  

            pData->data_express.last_start_angle =
                pData->data_express.start_angle;  

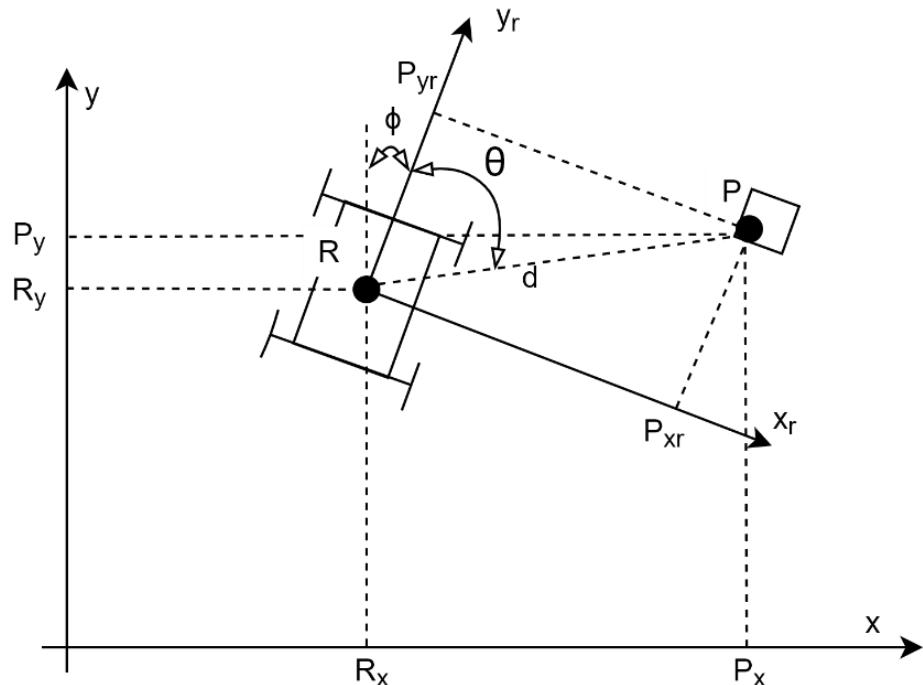
            bool newscan = arr[3 + j * 84] & (1 << 7);
            for (u_int8_t i = 0; i < 16; i++) {
                u_int16_t dist1 = (arr[4 + i * 5 + j * 84] >> 2)
                    + (arr[5 + i * 5 + j * 84] << 6);
                u_int16_t dist2 = (arr[6 + i * 5 + j * 84] >> 2)
                    + (arr[7 + i * 5 + j * 84] << 6);
                float dfi1 = (float) ((int8_t)((u_int8_t)(arr[8 + i * 5 + j * 84]&0x0F)
                    + ((u_int8_t)(arr[4 + i * 5 + j * 84] & 0x03)<< 4)) << 2)) / 32.0f;
                float dfi2 = (float) ((int8_t)((u_int8_t)(arr[8 + i * 5 + j * 84] >> 4)
                    + ((u_int8_t)(arr[6 + i * 5 + j * 84] & 0x03)<< 4)) << 2)) / 32.0f;
                float fi1 = pData->data_express.start_angle
                    + (pData->data_express.delta_angle / 32.0f)* (2 * i + 1) - dfi1;
                float fi2 = pData->data_express.start_angle
                    + (pData->data_express.delta_angle / 32.0f)* (2 * i + 2) - dfi2;
                pData->data_express.measure[i * 2].distance = dist1;
                pData->data_express.measure[i * 2].angle = fi1;
                pData->data_express.measure[i * 2 + 1].distance = dist2;
                pData->data_express.measure[i * 2 + 1].angle = fi2;
            }
            emit measureComplete();
            qDebug()<<"Packet ok!";
        }
    }
}

```

Lis. 13. Odbiór pakietu zawierającego wyniki pomiarów w trybie *express scan*.

3.4.3. Wyznaczenie pozycji przeszkody

Skaner laserowy zwraca wynik pomiaru w postaci odległości d pomiędzy jego głowicą a przeszkodą oraz kąta θ pod jakim znajduje się ta przeszkoda (rys. 15). LIDAR wykorzystuje biegunowy układ współrzędnych w określeniu pozycji przeszkody względem punktu osi obrotu głowicy pomiarowej. Urządzenie zamocowane jest na przemieszczającej się swobodnie platformie mobilnej. Jej położenie R w kartezjańskim układzie współrzędnych opisują trzy wartości: R_x - pozycja wzdłuż osi x , R_y - pozycja wzdłuż osi y , ϕ - orientacja względem osi y .



Rys. 25. Położenie przeszkody w układzie współrzędnych mapy i robota.

W celu określenia pozycji przeszkody w układzie współrzędnych mapy należy wyznaczyć jej pozycję w lokalnym układzie współrzędnych robota przechodząc z współrzędnych biegunowych na kartezjańskie (3) a następnie wykorzystując macierz przekształcenia jednorodnego (4) otrzymać współrzędne P_x oraz P_y punktu pojedynczego pomiaru (rys. 25).

$$\begin{aligned} P_{xr} &= d \cdot \sin \theta \\ P_{yr} &= d \cdot \cos \theta \end{aligned} \quad (3)$$

$$\begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi & R_x \\ \sin\varphi & \cos\varphi & R_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} P_{rx} \\ R_{ry} \\ 1 \end{bmatrix} \quad (4)$$

Przekształcając (3) i (4) wykorzystując tożsamości trygonometryczne otrzymujemy:

$$\begin{aligned} P_x &= R_x + d \cdot \sin(\varphi + \theta) \\ P_y &= R_y + d \cdot \cos(\varphi + \theta) \end{aligned} \quad (5)$$

Powyższe zależności (5) zostały zaimplementowane w programie w postaci funkcji (lis. 14). Jako argument przyjmuje ona dane punktu pomiarowego, a informację o aktualnym położeniu robota pobiera bezpośrednio z globalnej struktury danych. W wyniku jej działania otrzymujemy punkt, którego współrzędne zawarte są w indywidualnym strukturalnym typie danych.

```

typedef struct {
    u_int16_t distance;           // [mm]
    float angle;                 // [ ]
} e_singlemeasure;

typedef struct {
    float xpos;
    float ypos;
} e_singlepoint;

e_singlepoint RPLidar::fromPolarToCartesian(e_singlemeasure measure) {
    e_singlepoint tmp_point;
    tmp_point.xpos = pData->roverpos.xpos+ measure.distance
        * qSin(qDegreesToRadians(measure.angle + pData->roverpos.yaw))
        / 1000.0;
    tmp_point.ypos = pData->roverpos.ypos+ measure.distance
        * qCos(qDegreesToRadians(measure.angle + pData->roverpos.yaw))
        / 1000.0;
    return tmp_point;
}

```

Lis. 14. Funkcja wyznaczająca położenie przeszkody.

3.4.4. Generowanie mapy

Na podstawie punktów położenia przeszkody zbudować można dwuwymiarową mapę. W projektowanej aplikacji składa się ona komórek w kształcie kwadratu reprezentujących wycinek otoczenia. Zawierają one informacje o typie jaki został do nich przypisany. Są to następujące typy: przestrzeń wolna, przeszkoda, strefa bezpieczeństwa. Na potrzeby algorytmu planowania ruchu, komórki mogą przyjmować również typy: początek trasy, koniec trasy, punkt do rozpatrzenia, punkt rozpatrzony, tor ruchu, punkt pośredni trasy.

Po określeniu przez użytkownika aplikacji wielkości mapy oraz rozmiaru pojedynczej komórki możliwe jest utworzenie obiektu jej klasy. Wywoływany wtedy konstruktor (lis. 15). Wykorzystując wprowadzone parametry tworzy on w pamięci operacyjnej dwuwymiarową tablicę komórek mapy. Przypisane zostają im m. in. współrzędne środka w układzie współrzędnych mapy oraz wartości początkowe dla zmiennych biorących udział podczas planowania trasy. W przypadku wybrania trybu symulacyjnego do losowo wybranych komórek zostaje przypisana obecność przeszkody. Podczas pracy konstruktor oblicza i wysyła na konsole całkowitą liczbę pól oraz ilość pamięci potrzebnej do przechowywania mapy.

```

map::map(int mapSize, float cellSize, bool simulate, e_map_type mapType,
         QObject *parent) :QObject(parent) {
    map_size = mapSize;
    cell_size = cellSize;
    simulate_mode = simulate;
    num_of_cells = (mapSize / cellSize) + 1;
    map_cell = new e_map_cell*[num_of_cells];
    for (int i = 0; i < num_of_cells; i++) {
        map_cell[i] = new e_map_cell[num_of_cells];
    }
    if (mapType == centered) {
        shift_coef = (num_of_cells - 1) * cell_size / 2.0;
    } else {
        shift_coef = 0;
    }
    for (int x = 0; x < num_of_cells; x++) {
        for (int y = 0; y < num_of_cells; y++) {
            map_cell[x][y].x_pos = (float) (x * cell_size - shift_coef);
            map_cell[x][y].y_pos = (float) (y * cell_size - shift_coef);
            map_cell[x][y].pos.x_index = x;
            map_cell[x][y].pos.y_index = y;
            map_cell[x][y].g = 0;
            map_cell[x][y].f = 0;
            map_cell[x][y].h = 0;
            map_cell[x][y].have_parent = false;
            map_cell[x][y].cell_type = clear;
            if (simulate_mode) {
                if ((qrand() % 100) < 1)
                    map_cell[x][y].cell_type = obstacle;
            }
        }
    }
    qDebug() << "Cells in row: " << num_of_cells << "Total cells:"
        << num_of_cells * num_of_cells << " Size:"
        << num_of_cells * num_of_cells * sizeof(e_map_cell) / 1024
        << "KB Shift Coef: " << shift_coef / cell_size;
    end_cell.x_index = 0;
    end_cell.y_index = 0;
    for (int i = 0; i < BUF_LEN; i++) frame_buf[i] = 0;
    bytes_to_send = 0;
}

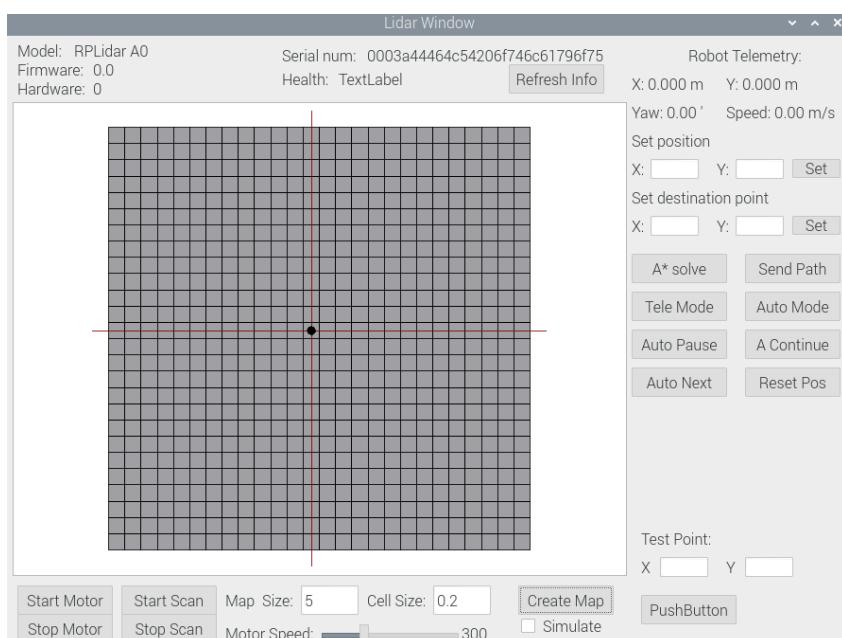
```

Lis. 15. Konstruktor klasy map.

Zwizualizowanie utworzonej w pamięci mapy możliwe jest poprzez wywołanie funkcji jej rysującej na polu graficznym znajdującym się w oknie aplikacji (lis. 16). Każda z komórek mapy odzwierciedlana jest poprzez kwadrat, który zmienia swój kolor w zależności od przyjętego przez nią typu. Ich rozmiar jest zależny od parametru wielkości komórki mapy wprowadzanego podczas tworzenia jej obiektu. Na ekranie dodatkowo zostają wyświetlane linie osi x oraz y wskazujące początek układu współrzędnych oraz punkt wskazujący pozycję robota (rys. 26).

```
void map::drawMap(QGraphicsScene *scene) {
    scene->clear();
    cell = new QGraphicsRectItem**[num_of_cells];
    for (int i = 0; i < num_of_cells; i++) {
        cell[i] = new QGraphicsRectItem*[num_of_cells];
    }
    for (int i = 0; i < num_of_cells; i++) {
        for (int j = 0; j < num_of_cells; j++) {
            cell[i][j] = scene->addRect(map_cell[i][j].x_pos * 100,
                map_cell[i][j].y_pos * -100, cell_size * 100,
                cell_size * 100, QPen(Qt::black), QBrush(Qt::gray));
            if (map_cell[i][j].cell_type == obstacle)
                cell[i][j]->setBrush(QBrush(Qt::darkBlue));
        }
    }
    scene->addLine(map_cell[0][0].x_pos * 100 - 100 * cell_size,
        cell_size * 50, map_cell[num_of_cells - 1][0].x_pos * 100 + 200 *
        cell_size, cell_size * 50, QPen(Qt::darkRed));
    scene->addLine(cell_size * 50, map_cell[0][0].y_pos * -100 + 200 *
        cell_size, cell_size * 50, map_cell[0][num_of_cells - 1].y_pos *
        -100 - 100 * cell_size, QPen(Qt::darkRed));
    rover_pos = scene->addEllipse(0, 0, 10, 10, QPen(Qt::black),
        QBrush(Qt::black));
}
```

Lis. 16. Funkcja rysująca mapę w oknie głównym aplikacji.



Rys. 26. Wizualizacja pustej mapy.

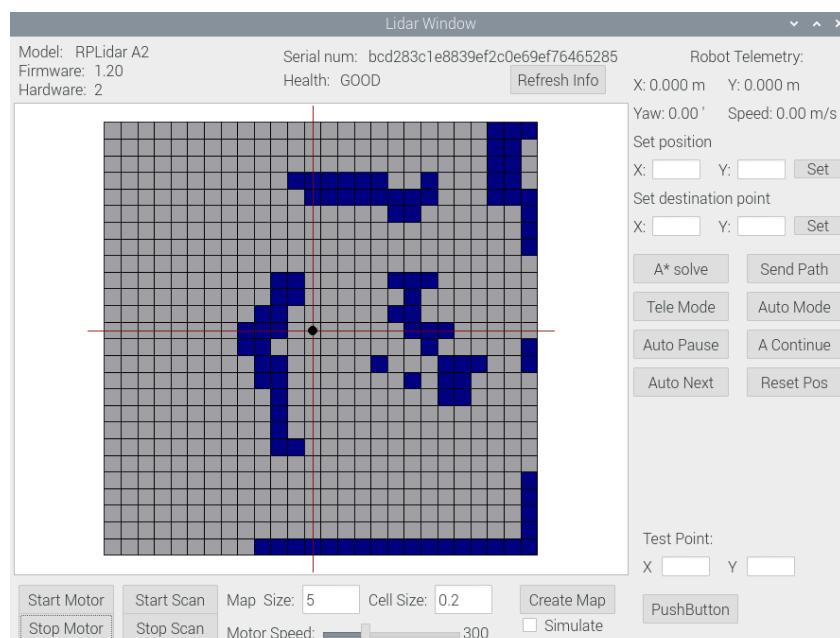
Po rozpoczęciu skanowania otoczenia przez LIDAR następuje aktualizacja pól mapy. Po przetworzeniu danych wyemitowany sygnał zakończonego pomiaru (lis. 13) powoduje wywołanie funkcji wyznaczającej dla każdego z zestawu 32 pomiarów pozycję przeskody, a następnie numer pola mapy, które to pole zostaje następnie zaktualizowane (rys. 27).

```

void lidarwindow::measure_calculate() {
    for (int i = 0; i < 32; i++) {
        if (rplidar->pData->data_express.measure[i].distance != 0)
            Map->actualizeCell(Map->getCellPos(
                rplidar->fromPolarToCartesian(
                    rplidar->pData->data_express.measure[i])),obstacle);
    }
}

void map::actualizeCell(e_cell_pos pos, e_cell_type type) {
    QBrush brush(Qt::black);
    if (type == obstacle) brush.setColor(Qt::darkBlue);
    else if (type == start) brush.setColor(Qt::green);
    else if (type == end) brush.setColor(Qt::red);
    else if (type == open_set) brush.setColor(Qt::darkGreen);
    else if (type == close_set) brush.setColor(Qt::darkCyan);
    else if (type == path_cell) brush.setColor(Qt::darkRed);
    else if (type == path_to_send) brush.setColor(QColor(168, 0, 0));
    else if (type == safe_zone_cell) brush.setColor(QColor(0, 0, 168));
    else brush.setColor(Qt::gray);
    e_cell_type tmptype = map_cell[pos.x_index][pos.y_index].cell_type;
    cell[pos.x_index][pos.y_index]->setBrush(brush);
    map_cell[pos.x_index][pos.y_index].cell_type = type;
    if (((tmptype == path_cell) || (tmptype == path_to_send))
        && (type == obstacle)) {
        emit reCalculatePath();
    }
}

```



Rys. 27. Aktualizowana mapa podczas wykonywania pomiarów.

Funkcja aktualizująca pola mapy wizualizuje wszystkie typy jakie mogą one przyjąć a każdemu przypada inny kolor. Są one przypisywane następująco:

- szary – wolna przestrzeń,
- ciemnoniebieski – przeszkoda,
- jasnoniebieski – strefa bezpieczeństwa wokół przeszkody,
- zielony – początek trasy,
- czerwony – koniec trasy,
- ciemnozielony – punkt do rozpatrzenia przez algorytm wyznaczania trasy,
- niebiesko-zielony – punkt rozpatrzony,
- ciemnoczerwony – wyznaczona trasa przejazdu.

3.4.5. Algorytm wyznaczania trasy

Wyznaczanie trasy w przypadku niepełnej znajomości otoczenia nie gwarantuje osiągnięcia celu a w przypadku jego powodzenia nie jest ona optymalna. W projekcie systemu przyjmuje, że jeżeli nie znamy czy w danym obszarze znajduje się przeszkoda to zakładamy, że obszar ten jest wolny od przeszkód. W przypadku gdy na wyznaczonej ścieżce podczas aktualizacji mapy pojawi się przeszkoda algorytm ponownie wyznaczy trasę od punktu, w którym robot znajduje się aktualnie. W aplikacji za planowanie przejazdu odpowiedzialny jest algorytm A*. Jest to algorytm heurystyczny wyznaczający m.in. najkrótszą ścieżkę pomiędzy dwoma punktami. Jest to algorytm zupełny i optymalny, co oznacza, że jeżeli występuje połączenie pomiędzy punktami to zostanie znalezione najlepsze rozwiązanie [31]. Działanie algorytmu oparte jest na minimalizacji funkcji celu (6), zdefiniowanej jako suma funkcji kosztu $g(x)$ oraz funkcji heurystycznej $h(x)$.

$$f(x) = g(x) + h(x) \quad (6)$$

W każdej kolejnej iteracji algorytm przedłuża już utworzoną ścieżkę o kolejny punkt, wybierając taki, w którym wartość funkcji będzie najmniejsza. Funkcja $g(x)$ określa rzeczywisty koszt dojścia do danego pola. Funkcja $f(x)$ jest to tak zwana heurystyczna funkcja celu. Oszacowuje ona koszt dotarcia od punktu startowego do docelowego.

Przed uruchomieniem algorytmu (lis. 17) należy określić punkt docelowy trasy robota. Przed rozpoczęciem przeszukiwania mapy zostają wyczyszczone wszystkie wymagane zmienne biorące udział w wyznaczaniu trasy. Dodatkowo wokół przeszkód zostaje utworzona strefa bezpieczeństwa, czyli punkty wyłączone możliwości ruchu robota.

```

void map::aStarSolver() {
    clearAStar();
    createSafeZone(false);
    openSet.append(getCellFromIndex(start_cell));
    int winner = 0;
    e_map_cell current;
    e_map_cell tmp_cell;
    while (openSet.length() > 0) {
        for (int i = 0; i < openSet.length(); i++) {
            if (winner >= openSet.length())
                winner = openSet.length() - 1;
            if (openSet[i].f < openSet[winner].f)
                winner = i;
        }
        current = openSet[winner];
        if ((current.pos.x_index == getCellFromIndex(end_cell).pos.x_index)
        && (current.pos.y_index == getCellFromIndex(end_cell).pos.y_index)){
            qDebug() << "Solve complete!";
            path.append(current);
            path.append(tmp_cell);
            int i = 0;
            while (tmp_cell.have_parent) {
                i++;
                path.append(getCellFromIndex(tmp_cell.parent));
                tmp_cell = getCellFromIndex(tmp_cell.parent);
            }
            for (int i = 0; i < path.length(); i++) {
                actualizeCell(path[i].pos, path_cell);
            }
            return;
        }
        openSet.remove(winner);
        closeSet.append(current);
        QVector<e_map_cell*> neighbours = getNeighbours(current.pos);
        for (int i = 0; i < neighbours.length(); i++) {
            if ((isInclude(closeSet, *neighbours[i]) == false)
            && ((neighbours[i]->cell_type != obstacle)
            && (neighbours[i]->cell_type != safe_zone_cell))) {
                float temp_g = current.g
                    + heuristic(*neighbours[i], current);
                if (isInclude(openSet, *neighbours[i]) == true) {
                    if (temp_g < neighbours[i]->g) {
                        neighbours[i]->g = temp_g;
                        neighbours[i]->h = heuristic(*neighbours[i],
                            getCellFromIndex(end_cell));
                        neighbours[i]->f = neighbours[i]->g+neighbours[i]->h;
                        neighbours[i]->parent = current.pos;
                        neighbours[i]->have_parent = true;
                    }
                } else {
                    neighbours[i]->g = temp_g;
                    neighbours[i]->h = heuristic(*neighbours[i],
                        getCellFromIndex(end_cell));
                    neighbours[i]->f = neighbours[i]->g+neighbours[i]->h;
                    neighbours[i]->parent = current.pos;
                    neighbours[i]->have_parent = true;
                    openSet.append(*neighbours[i]);
                }
            }
        }
    }
}

```

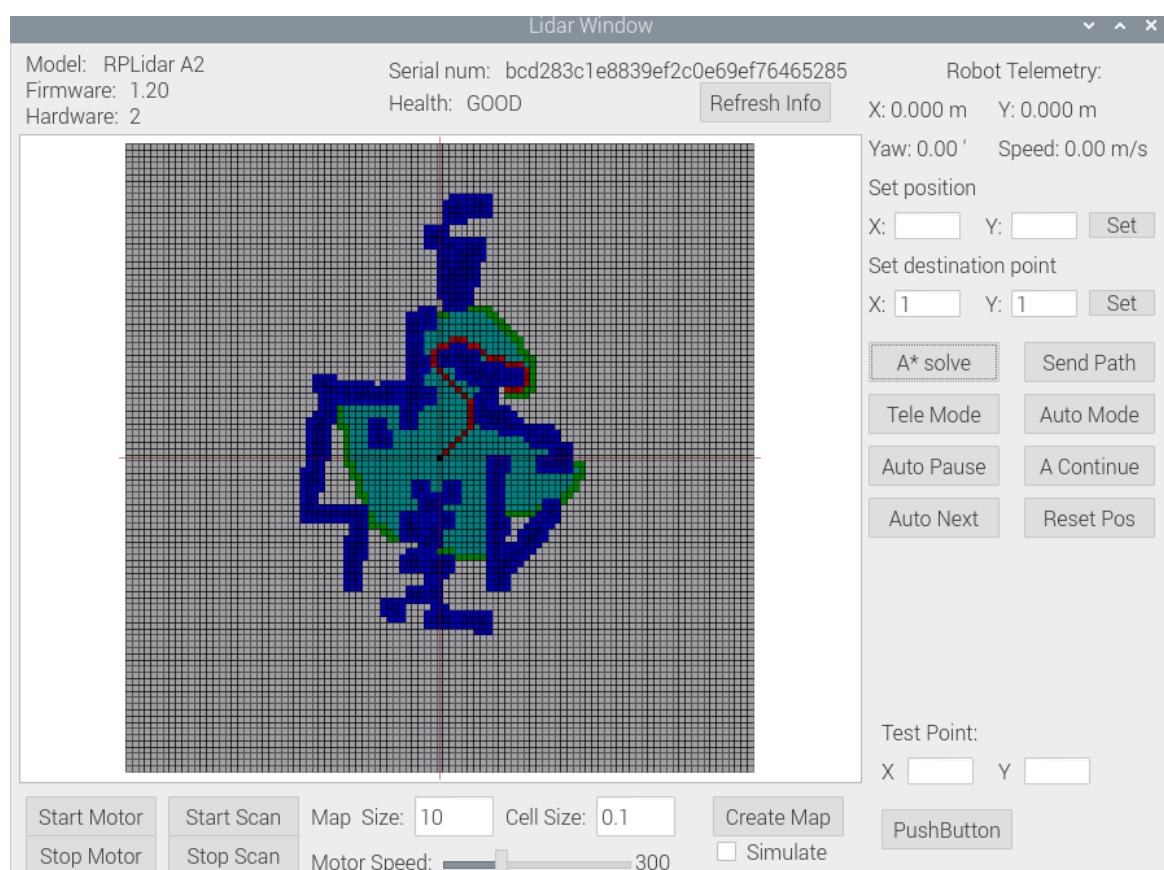
```

        }
        for (int i = 0; i < closeSet.length(); i++) {
            actualizeCell(closeSet[i].pos, close_set);
        }
        for (int i = 0; i < openSet.length(); i++) {
            actualizeCell(openSet[i].pos, open_set);
        }
        tmp_cell = current;
    }
    qDebug() << "No solution";
    return;
}

```

Lis. 17. Implementacja algorytmu A*.

W wyniku działania zaimplementowanego algorytmu A* wyznaczona zostaje ścieżka, po której ma poruszać się robot (rys. 28). Trasa jak i rozpatrywane przez algorytm punkty są wizualizowane na mapie. Po przesłaniu punktów pośrednich do robota możliwe jest uruchomienie trybu autonomicznego.



Rys. 28. Otrzymana trasa ruchu robota w wyniku działania algorytmu A*

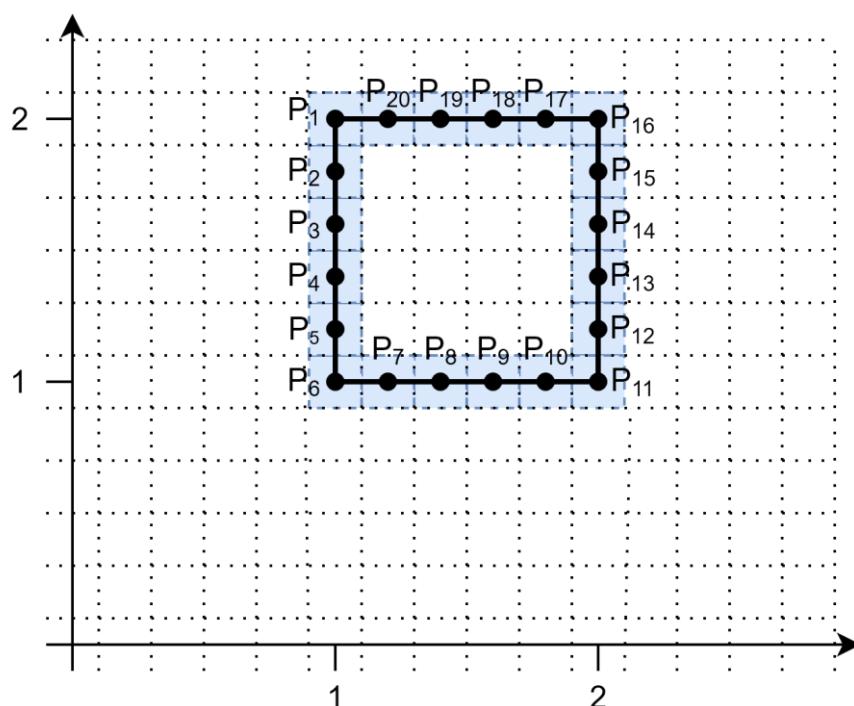
4. BADANIA SYMULACYJNE I EKSPERYMENTALNE SYSTEMU

4.1. BADANIE POPRAWNOŚCI DZIAŁANIA ALGORYTMU MAPOWANIA

Poprawność działania algorytmu mapowania zaimplementowanego w projekcie została przeprowadzona z wykorzystaniem dwóch metod. Pierwsza z nich polegała na zasymulowaniu serii punktów, w których znajduje się przeszkoda i porównaniu utworzonej mapy z przewidywanym rozmieszczeniem przeszkód. Druga metoda polegała na wykonaniu testowego przejazdu robota z uruchomionym systemem mapującym przy znanym położeniu przeszkód w otoczeniu robota.

4.1.1. Tworzenie mapy na podstawie punktów testowych

Sprawdzenie poprawności systemu mapowania może zostać przeprowadzone poprzez przekazanie do algorytmu testowych punktów, które tworzą obrys wirtualnej przeszkody w kształcie kwadratu o boku 1 m. Podczas wykonywania testu została wygenerowana pusta mapa z parametrem wielkości boku pojedynczego pola ustalonym na 0,2 m. Po wprowadzeniu współrzędnych punktów od P_1 do P_{20} algorytm powinien zaktualizować wskazane na rysunku (rys. 29) komórki.

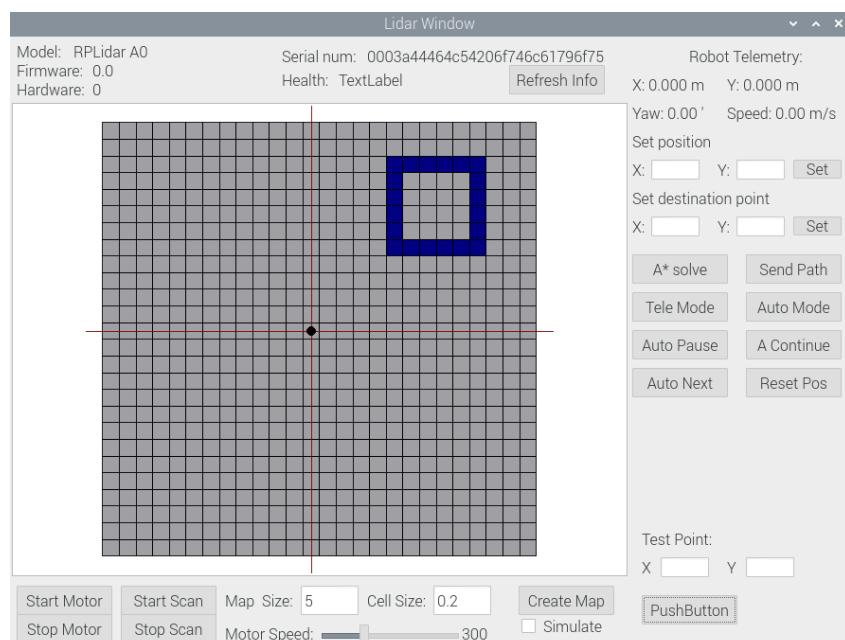


Rys. 29. Punkty testowe oraz przewidywane wyznaczenie pól mapy.

Tabela 6

Współrzędne testowych punktów

Punkt	Współrzędna <i>x</i> [m]	Współrzędna <i>y</i> [m]
P_1	0,96	1,99
P_2	0,99	1,82
P_3	1,03	1,60
P_4	1,00	1,38
P_5	1,01	1,19
P_6	0,98	0,98
P_7	1,17	1,03
P_8	1,41	0,99
P_9	1,58	1,00
P_{10}	1,83	1,02
P_{11}	2,00	0,96
P_{12}	1,98	1,22
P_{13}	2,03	1,39
P_{14}	2,01	1,61
P_{15}	1,97	1,78
P_{16}	2,00	1,95
P_{17}	1,78	2,00
P_{18}	1,59	2,04
P_{19}	1,42	1,99
P_{20}	0,99	1,96

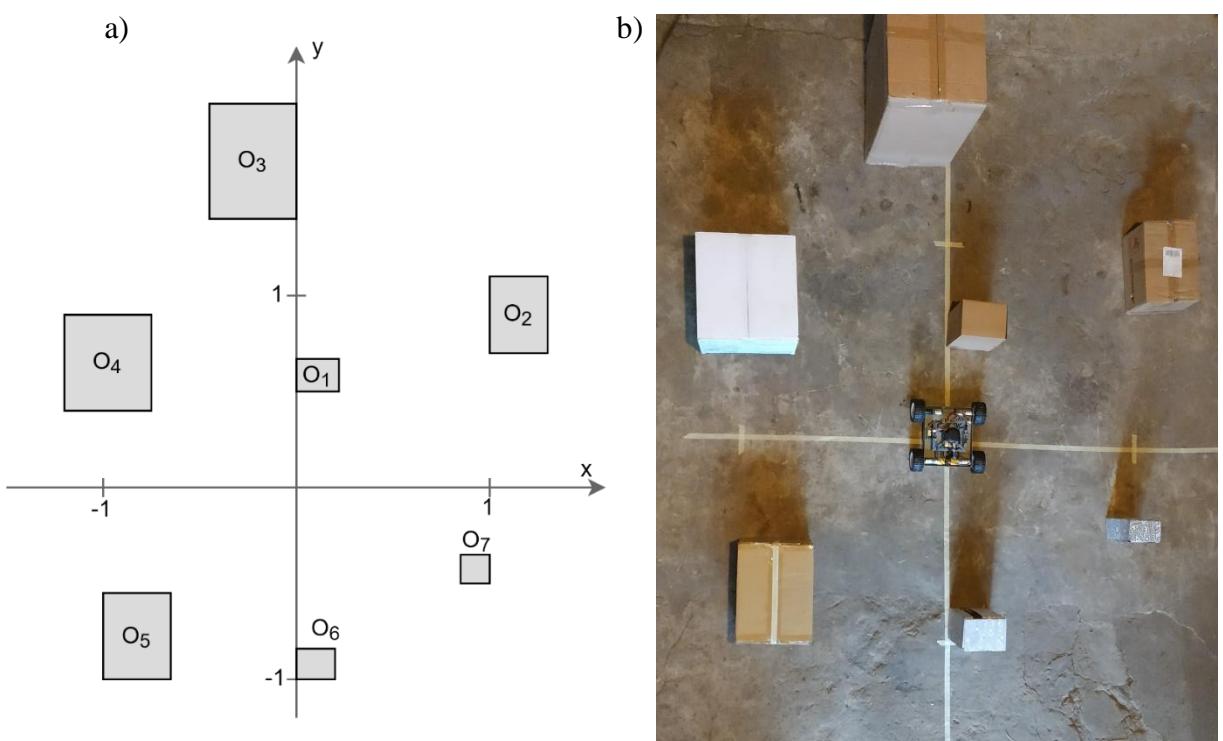


Rys. 30. Mapa otrzymana po wprowadzeniu punktów testowych.

Po przeprowadzeniu badania poprawności działania systemu mapowania została otrzymana mapa (rys. 30). Zaznaczone jako przeszkoda pola pokrywają się z przewidywanymi. Wnioskiem z przeprowadzonego testu jest stwierdzenie, że system w prawidłowy sposób przetwarza punkty otrzymane z pomiarów na odpowiednie pola mapy, które reprezentują obecność przeszkody.

4.1.2. Przejazd testowy.

Na potrzeby eksperymentu, który ma na celu sprawdzenie poprawności działania systemu map w warunkach rzeczywistych, został przygotowany tor (rys. 31). Rozmieszczo na nim 7 przeszkód o podstawach w kształcie prostokątów. Po wyznaczeniu pozycji umieszczonych obiektów wyznaczono współrzędne ich wierzchołków (tabela 7). Badanie polegało na wykonaniu przejazdu robotem przez tor skanując przestrzeń za pomocą LIDAR-u, w wyniku którego została otrzymana mapa. Parametry wprowadzone do programu generującego mapę to: rozmiar mapy 5 m na 5 m, rozmiar komórki 0,1 m. Daje to łączną liczbę 2601 pól oraz 91 kB zajętej pamięci operacyjnej.



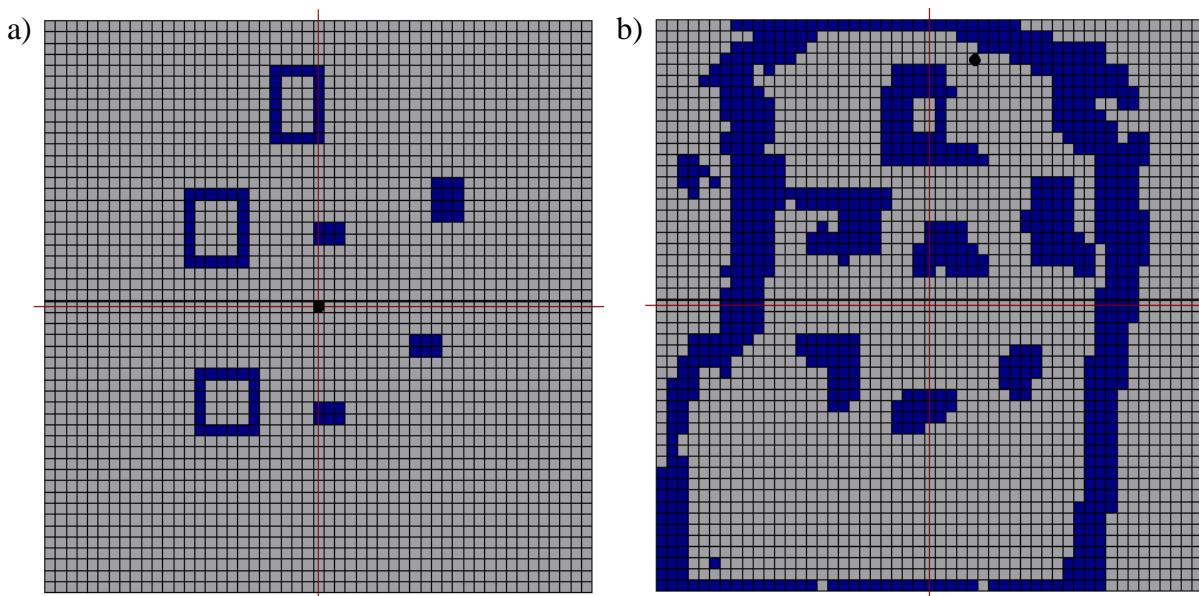
Rys. 31. Tor testowy a) schemat rozmieszczenia przeszkód b) przygotowany tor.

Tabela 7

Współrzędne wierzchołków przeszkód na torze

Obiekt	Lewy dolny wierzchołek		Prawy dolny wierzchołek		Prawy górny wierzchołek		Lewy górny wierzchołek	
	x [m]	y [m]	x [m]	y [m]	x [m]	y [m]	x [m]	y [m]
O_1	0,00	0,50	0,22	0,50	0,22	0,67	0,00	0,67
O_2	1,00	0,70	1,30	0,70	1,30	1,10	1,00	1,10
O_3	-0,45	1,40	0,00	1,40	0,00	2,00	-0,45	2,00
O_4	-1,20	0,40	-0,75	0,40	-0,75	0,90	-1,20	0,90
O_5	-1,00	-1,00	-0,65	-1,00	-0,65	-0,55	-1,00	-0,55
O_6	0,00	-1,00	0,20	-1,00	0,20	-0,85	0,00	-0,85
O_7	0,85	-0,45	1,00	-0,45	1,00	-0,35	0,85	-0,35

Po przeprowadzonym przejeździe testowym otrzymaną mapę (rys. 32 b), którą porównano z mapą utworzoną poprzez ręczne wprowadzenie do algorytmu punktów pomiarowych zgodnych z rozmieszczeniem przeszkód (rys. 32 a). Wizualizacja otoczenia otrzymana podczas eksperymentu oprócz wykrytych przeszkód zawiera obrys pomieszczenia, w którym został przygotowany tor. System mapowania poprawnie wykrył i wyznaczył pozycję przeszkód. Jednak ich obrys jest powiększony co spowodowane jest błędami wynikającymi z przesunięcia pozycji rzeczywistej robota a otrzymanej w wyniku algorytmu odometrycznego. Błąd pozycji robota bezpośrednio wpływa na wyznaczenie pozycji przeszkody co związane jest z sposobem jej wyznaczania (5). Otrzymana mapa może z powodzeniem posłużyć podczas planowania ruchu robota.



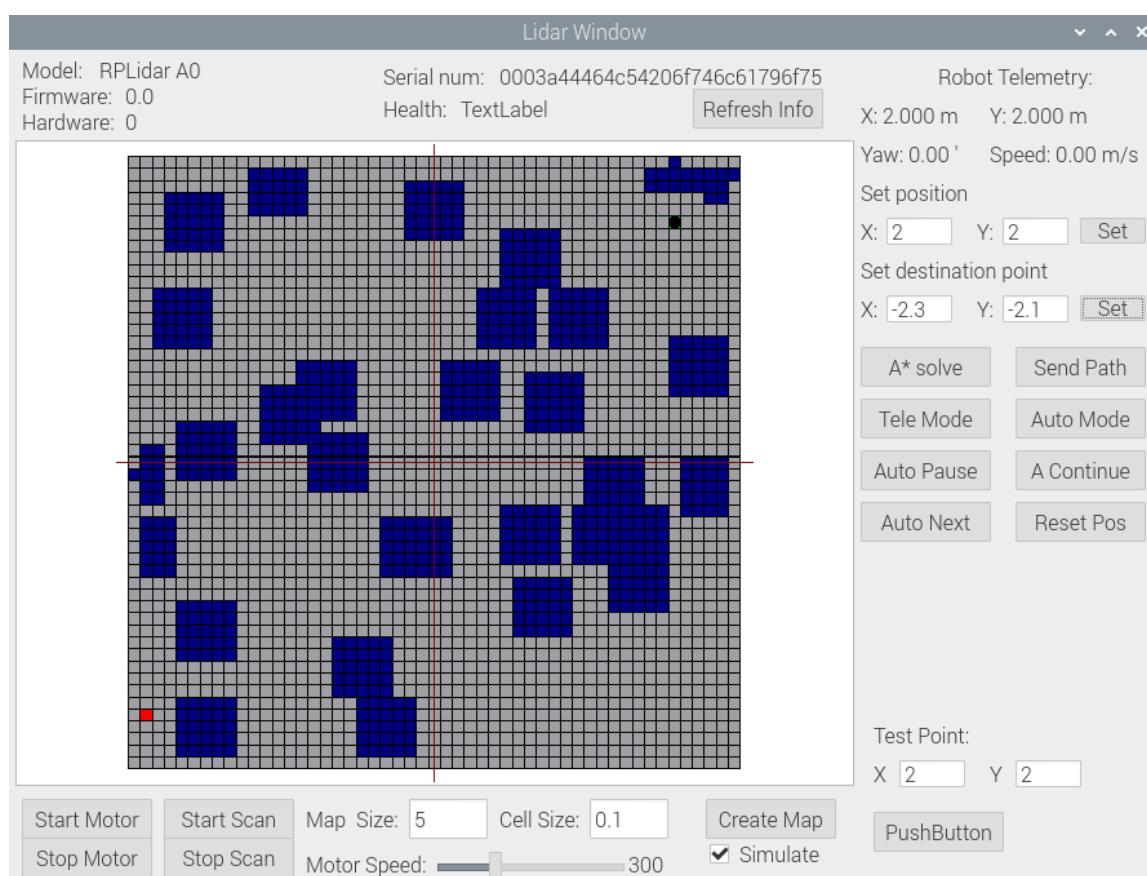
Rys. 32. Mapa toru na podstawie: a) punktów wprowadzonych ręcznie, b) przejazdu testowego.

4.2. BADANIE POPRAWNOŚCI DZIAŁANIA ALGORYTMU PLANOWANIA RUCHU

Działanie algorytmu planowania ruchu robota mobilnego zostało sprawdzone zarówno w warunkach symulacyjnych jak i podczas przeprowadzonego eksperymentu. Badanie symulacyjne opiera się na przeszukiwaniu mapy z losowo rozmieszczonymi przeszkodami. Podstawą eksperymentu jest natomiast określenie zachowania się algorytmu oraz jego wynikowej trasy w warunkach rzeczywistych na przygotowanym torze testowym.

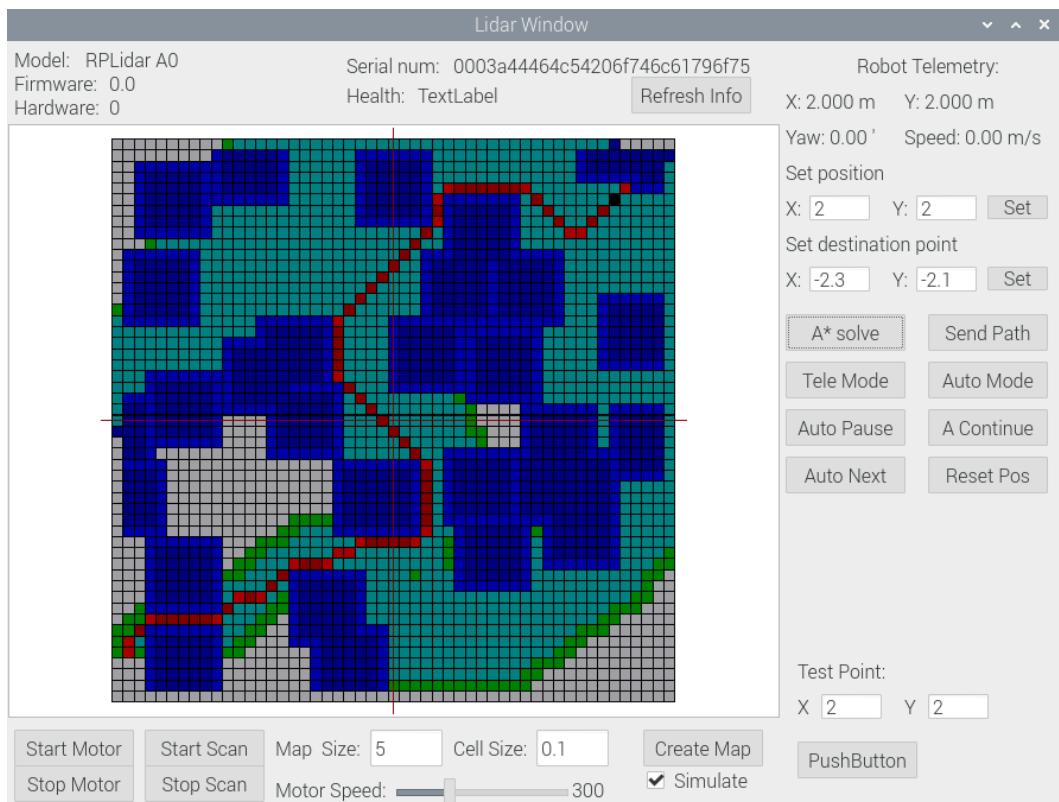
4.2.1. Wyznaczenie toru ruchu w losowo wygenerowanej mapie

Losowo wygenerowana mapa może posłużyć do sprawdzenia poprawności działania algorytmu A*. W przypadku mapy symulacyjnej możemy uznać że jest ona zupełna, czyli zostały w niej zawarte wszystkie przeszkody miesiące się w jej obrębie. W trybie testowym przed rozpoczęciem wyznaczania trasy należy ustalić punkt początkowy oraz punkt docelowy (rys. 33).

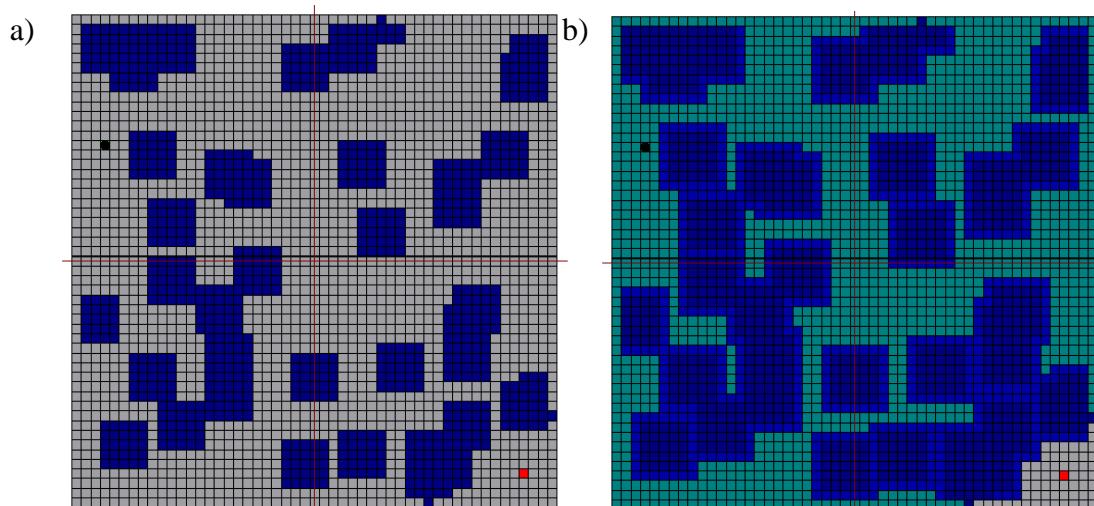


Rys. 33. Wygenerowana losowo mapa z wskazanymi punktami startu i końca trasy

W wyniku działania algorytmu otrzymana została ścieżka ruchu oznaczona za pomocą koloru czerwonego. Wokół przeszkód (kolor ciemno-niebieski) zostały wyznaczone strefy bezpieczeństwa (kolor niebieski), w których zabronione jest przeprowadzenie trasy. Zwizualizowane również zostały punkty, które brały udział podczas przeszukiwania mapy (kolor turkusowy) oraz punkty kandydujące do rozpatrzenia przez algorytm (kolor zielony) (rys. 34).



Rys. 34. Wyznaczona trasa



Rys. 35. Działanie algorytmu gdy nie możliwe jest wyznaczenie trasy a) widok przed b) widok po zakończeniu pracy algorytmu.

W przypadku gdy rozmieszczenie przeszkód na mapie uniemożliwia wyznaczenie toru ruchu algorytm, po przeanalizowaniu umożliwiających przejazdów pól mapy, zwraca informację o braku rozwiązania (rys. 35).

Otrzymane rozwiązanie jest optymalne z punktu widzenia heurystycznej funkcji celu (6), która zawiera człon $g(x)$ wyznaczający koszt przemieszczenia się z punktu startu do aktualnie rozpatrywanego punktu oraz człon heurystyczny $h(x)$, który przewiduje koszt dotarcia do celu. Wartość ta jest wyznaczana na podstawie odległości pomiędzy aktualnym punktem a punktem docelowym (funkcja euklidesowa). Funkcja ta jest monotoniczna czyli szacowana wartość jest zawsze mniejsza lub równa od rzeczywistej odległości do wyznaczonego punktu.

Zastosowany algorytm wykorzystujący euklidesową funkcję heurystyczną zwraca rozwiązanie nieoptymalne. W celu otrzymania optymalnego rozwiązania należy zmodyfikować przeszukiwanie z wykorzystaniem zmodyfikowanej funkcji heurystycznej.

4.2.2. Wyznaczenie ścieżki ruchu robota na torze testowym

Działanie zaproponowanego algorytmu zostało również sprawdzone w warunkach rzeczywistych na przygotowanym torze testowym (rys. 31). Eksperyment polegał na wykonaniu przejazdu w sposób autonomiczny do wyznaczonych punktów docelowych omijając przeszkody (rys. 36).



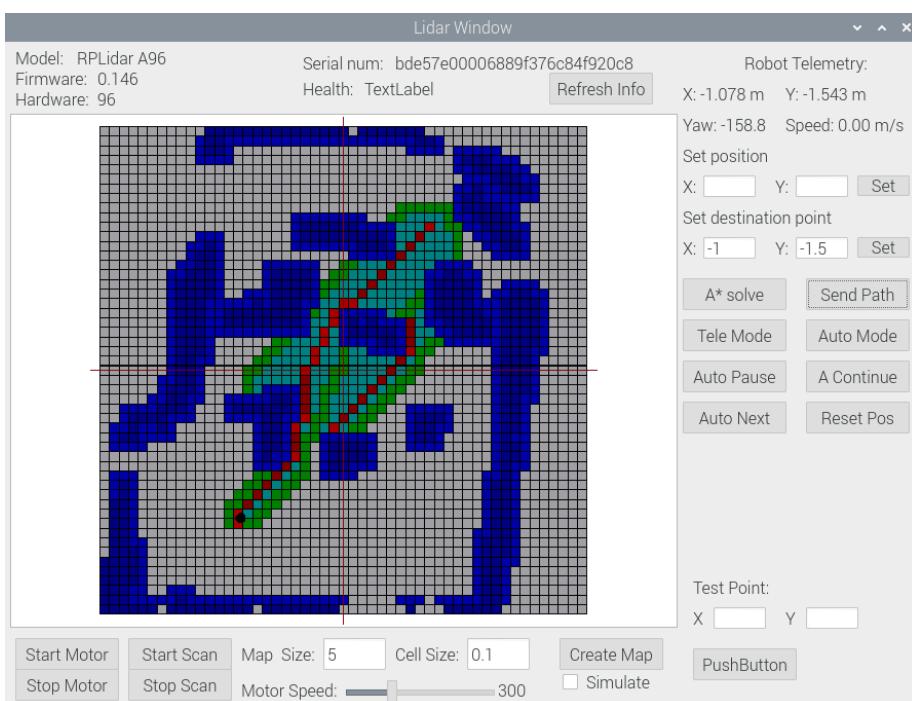
Rys. 36. Robot omijający przeszkodę.

Robot został umieszczony w punkcie startowym (początek układu współrzędnych mapy) a następnie do systemu mapowania i planowania ruchu wprowadzony został punkt docelowy $P_1(0,85; 1,5)$. Został uruchomiony LIDAR oraz algorytm wyznaczający trasę. Wyznaczone punkty docelowe po zostały wysłane do układu sterowania podwoziem i zainicjowano tryb

autonomiczny (rys. 37). Po osiągnięciu punktu P_1 został wyznaczony nowy punkt docelowy $P_2(-1; -1,5)$. Została wyznaczona nowa trasa, która następnie została przesłana do platformy mobilnej. Po ponownym uruchomieniu trybu autonomicznego robot pokierował się w stronę nowego punktu docelowego (rys. 38). W trakcie obu przejazdów przeszkody znajdujące się na torze testowym zostały poprawnie ominiete.



Rys. 37. Wyznaczona trasa do punktu P_1



Rys. 38. Wyznaczona trasa z punktu P_1 do punktu P_2

5. WNIOSKI I PODSUMOWANIE

Rozdział ten przeznaczony jest przedstawieniu wniosków wyciągniętych podczas realizacji projektu. Przedstawiono w nim zdobyte nowe umiejętności i rozwój obecnych oraz zebrane doświadczenia. Znalazły się w nim również plany modyfikacji i rozwoju zaproponowanych rozwiązań.

5.1. WNIOSKI

Celem pracy było wykonanie projektu systemu mapowania otoczenia i planowania ruchu robota mobilnego. W jego skład wchodziło przygotowanie zarówno strony sprzętowej jak i oprogramowania platformy mobilnej oraz jednostki przetwarzającej dane. Zagadnienia związane z autonomicznym poruszaniem się robota oraz systemów samolokalizacji i mapowania wymagają dużego nakładu prac podczas realizacji tego typu projektu. Przygotowanie odpowiednich algorytmów zdolnych do samodzielnego przygotowania przez robota mapy otoczenia i ścieżki ruchu związane jest z przeprowadzaniem odpowiednich testów na podstawie których wprowadzane są ich udoskonalenia. Na działanie systemu mapowania otoczenia ma wpływ poprawna estymacja aktualnego położenia platformy mobilnej na której jest on zamontowany. Błędne oszacowanie pozycji i orientacji w układzie współrzędnych powoduje błędne wyznaczenie pozycji przeszkód mimo poprawnego działania sensorów skanujących otoczenie. System mapowania otoczenia może również służyć jako dodatkowa wizualizacja przestrzeni wokół robota np. w zastosowaniach inspekcyjnych gdzie obraz z kamer jest niewystarczający.

5.2. NABYTE UMIEJĘTNOŚCI

Prace związane z projektem systemu przyczyniły się do utrwalenia wiedzy zdobytej w trakcie toku nauczania oraz zdobyciu nowych umiejętności związanych z szeroko pojętą automatyką. Projekt opierał się głównie rozwoju wiedzy z dziedzin takich jak elektronika oraz programowanie. Znalazły się w nim również elementy mechaniki i konstrukcji elementów a także telekomunikacji.

Podczas realizacji projektu nabyte zostały podstawy tworzenia aplikacji dla systemu Android oraz programów graficznych z wykorzystaniem środowiska programistycznego Qt oraz rozwinięte zostały umiejętności programowania w języku C systemów wbudowanych opartych o mikrokontrolery STM32.

5.3. MOŻLIWOŚCI ROZWOJU SYSTEMU

Przygotowany podczas realizacji projektu system jest podstawą dalszych prac i badań związanych z pojazdami autonomicznymi. Udoskonalenia wymaga zarówno oprogramowanie platformy mobilnej jak i programu odpowiedzialnego za generowanie mapy oraz algorytm wyznaczający ścieżkę ruchu.

W przypadku układu sterowania podwozia robota możliwa jest jego rozbudowa o moduł nawigacji inercyjnej, który pozwoli zminimalizować błędy estmacji jego położenia podczas przemieszczania się. Modyfikacją należy też poddać algorytm trybu autonomicznego tak aby robot jak najlepiej odwzorowywał wyznaczoną trasę ruchu.

Do procesu generowania mapy należy wprowadzić możliwość wykrycia przeszkód przemieszczających się w przestrzeni wokół robota. Obecnie pojedyncze wystąpienie pomiaru wskazującego na obecność przeszkody powoduje jest stałe umieszczenie w wygenerowanej mapie.

Algorytm A* przeszukujący mapę w celu wyznaczenia ścieżki ruchu poprawnie wyznacza tor ruchu omijając występujące przeszkody z uwzględnieniem stref bezpieczeństwa wokół nich. Otrzymane rozwiązanie jednak nie zawsze jest rozwiązaniem optymalnym co związane jest z zastosowaną heurystyczną funkcją celu. Problem ten może być rozwiązyany poprzez odpowiednią modyfikację tej funkcji lub poprzez zastąpienie algorytmu A* np. algorytmem propagacji fali który zawsze wyznaczy optymalne rozwiązanie jednak kosztem zwiększonej liczby operacji związanej z przeanalizowaniem każdej komórki mapy.

LITERATURA

- [1] M. M. Wieliczko, „Autonomiczne auta: wizja niedalekiej przyszłości”, *Autobusy: technika, eksploatacja, systemy transportowe*, nr 3, ss. 41–44, 2019.
- [2] W. Owczarzak, „Pojazdy autonomiczne na przykładzie samojeżdżącego samochodu Google'a”, *Logistyka*, nr 3, ss. 3680–3684, 2015.
- [3] M. Januszka, M. Adamczyk, i W. Moczulski, „Nieholonomiczny autonomiczny robot mobilny do inspekcji obiektów technicznych”, *Problemy Robotyki, Elektronika*, nr 166, ss. 143–152, 2008.
- [4] A. Typiak, „Dobór systemu rozpoznania otoczenia dla bezzałogowej platformy lądowej”, *Logistyka*, nr 3, ss. 6468–6475, 2014.
- [5] P. Richard, N. Pouliot, i S. Montambault, „Introduction of a LIDAR-based obstacle detection system on the LineScout power line robot”, w *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2014, ss. 1734–1740.
- [6] M. Karbowniczek, „Czujniki ultradźwiękowe w automatyce przemysłowej”, *Elektronika Praktyczna*, nr 5, ss. 122–129, 2015.
- [7] A. Staniszewska, „Systemy wizyjne – nieodzowny element nowoczesnej kontroli”, *Automatyka*, nr 6, 2017.
- [8] Z. Wawerek, „Machine vision, widzenie maszynowe albo...”, *Pomiary Automatyka Robotyka*, nr 4, ss. 18–19, 2008.
- [9] R. Tadeusiewicz, *Systemy wizyjne robotów przemysłowych*. Wydawnictwa Naukowo-Techniczne, 1992.
- [10] K. Łygas, „Algorytmy systemu wizyjnego stanowiska badawczego aplikacji pick and place”, *Autobusy: technika, eksploatacja, systemy transportowe*, nr 6, ss. 900–907, 2017.
- [11] C. Zieliński, „Quo vadis robotyko?”, *Pomiary Automatyka Robotyka*, nr 5, ss. 5–15, 2010.
- [12] Ł. Mitka, M. Ciszewski, A. Kudriashov, T. Buratowski, i M. Giergiel, „Robot z laserowym czujnikiem odległości do budowy map 2d”, *Modelowanie inżynierskie*, nr 61, ss. 27–33, 2017.
- [13] S. Kohlbrecher, J. Meyer, T. Gruber, K. Petersen, U. Klingauf, i O. Von Stryk, „Hector open source modules for autonomous mapping and navigation with rescue robots”, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, t. 8371 LNAI, ss. 624–631, 2014.
- [14] E. H. C. Harik i A. Korsaeth, „Combining hector SLAM and artificial potential field for autonomous navigation inside a greenhouse”, *Robotics*, t. 7, nr 2, 2018.

STRONY WWW

- [15]. <https://www.radartutorial.eu/> (stan z dnia 14.08.2020).
- [16]. https://pl.wikipedia.org/wiki/Kontrola_ruchu_lotniczego#/media/Plik:Deister-radar.jpg (stan z dnia 14.08.2020).
- [17]. <https://polska-org.pl/8375846,foto.html> (stan z dnia 14.08.2020).
- [18] <https://www.bosch-mobility-solutions.com/en/products-and-services/passenger-cars-and-light-commercial-vehicles/driver-assistance-systems/automatic-emergency-braking/long-range-radar-sensor/> (stan z dnia 14.08.2020).
- [19]. <https://elektronikab2b.pl/biznes/51090-branza-samochodowa-uksztaltuje-rynek-skannerow-lidar> (stan z dnia 14.08.2020).
- [20]. <https://motofocus.pl/informacje/nowosci/80263/nowa-tehnologia-w-wyposazeniu-samochodow-co-to-jest-lidar> (stan z dnia 14.08.2020).
- [21]. <https://strefainzyniera.pl/artykul/1032/czujnik-ultradzwiekowy> (stan z dnia 15.08.2020).
- [22]. [https://leksykon.forbot.pl/Dalmierz_optyczny_\(SHARP\).73.htm](https://leksykon.forbot.pl/Dalmierz_optyczny_(SHARP).73.htm) (stan z dnia 15.08.2020).
- [23]. <https://www.teamhector.de/> (stan z dnia 16.08.2020).
- [24]. <https://trifo.com/> (stan z dnia 16.08.2020).
- [25]. <https://wobit.com.pl/produkt/7999/edukacyjne-roboty-mobilne/mobot-explorer-a0-robot-mobilny-bez-elektroniki-sterujacej/> (stan z dnia 17.08.2020).
- [26]. <https://botland.com.pl/pl/arduino-shield-kontrolery-silnikow-i-serw/2262-vnh5019-dwukanalowy-sterownik-silnikow-24v12a-shield-dla-arduino-pololu-2507.html> (stan z dnia 18.08.2020).
- [27]. <https://botland.com.pl/pl/moduly-wifi/6807-modul-wifi-wifi232-z-wbudowana-antena-pcb-waveshare-7500.html> (stan z dnia 18.08.2020).
- [28]. <https://www.slamtec.com/en/Lidar/A2> (stan z dnia 19.08.2020).
- [29]. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/> (stan z dnia 19.08.2020).
- [30]. <https://docs.microsoft.com/pl-pl/xamarin/android/internals/architecture> (stan z dnia 24.08.2020).
- [31]. https://pl.wikipedia.org/wiki/Algorytm_A* (stan z dnia 27.08.2020).

STRESZCZENIE

Projekt systemu mapowania otoczenia oraz planowania ruchu robota mobilnego.

Praca miała na celu wykonanie projektu systemu mapowania otoczenia i planowania ruchu robota mobilnego. Działanie zaproponowanych rozwiązań zostało zbadane przy pomocy badań symulacyjnych i eksperymentalnych. Skanowanie otoczenia umożliwił LIDAR zamocowany na platformie mobilnej, której układ sterowania oparty jest o mikrokontroler STM32. Przetwarzanie danych pomiarowych i planowanie trasy ruchu robota z wykorzystaniem algorytmu A* odbywa się poprzez aplikację okienkową zaprojektowaną dla platformy rozwojowej RaspberryPi. Podczas realizacji projektu zostały użyte narzędzia takie jak: Keil uVison, Visual Studio, QtCreator, które umożliwiały kodowanie odpowiednio w języku C, C# i C++.

ABSTRACT

Project of the environment mapping system and mobile robot motion planning.

The aim of the thesis was to design a system for mapping the environment and planning the motion of a mobile robot. The operation of the proposed solutions was tested by simulation and experimental tests. Scanning of the environment was possible by LIDAR mounted on a mobile platform of robot, the control system of which is based on the STM32 microcontroller. Processing of measurement data and planning the route of the robot's movement using the A * search algorithm is carried out through a window application designed for the RaspberryPi single board computer. During the project implementation, tools such as Keil uVison, Visual Studio, QtCreator were used, which enabled coding in C, C # and C ++.

ZAŁĄCZNIKI

Załącznik nr 1 (autonomy.pro)

```
#-----
# 
# Project created by QtCreator 2020-03-14T19:08:17
#
#-----

QT      += core gui serialport

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = untitled
TEMPLATE = app

SOURCES += main.cpp \
          mainwindow.cpp \
          rplidar.cpp \
          lidarwindow.cpp \
          worker.cpp \
          map.cpp

HEADERS  += mainwindow.h \
            rplidar.h \
            lidarwindow.h \
            worker.h \
            map.h

FORMS    += mainwindow.ui \
            lidarwindow.ui
```

Załacznik nr 2 (mainwindow.ui)

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>MainWindow</class>
<widget class="QMainWindow" name="MainWindow">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>577</width>
<height>372</height>
</rect>
</property>
<property name="windowTitle">
<string>Autonomy App</string>
</property>
<widget class="QWidget" name="centralWidget">
<widget class=" QLabel" name="label">
<property name="geometry">
<rect>
<x>10</x>
<y>30</y>
<width>81</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Com Port:</string>
</property>
</widget>
<widget class="QComboBox" name="comboBox">
<property name="geometry">
<rect>
<x>100</x>
<y>30</y>
<width>141</width>
<height>31</height>
</rect>
</property>
</widget>
<widget class="QPushButton" name="OpenButton">
<property name="geometry">
<rect>
<x>10</x>
<y>110</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Open</string>
</property>
</widget>
<widget class="QPushButton" name="CloseButton">
<property name="enabled">
<bool>false</bool>
</property>
<property name="geometry">
<rect>
```

```

<x>140</x>
<y>110</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Close</string>
</property>
</widget>
<widget class="QTextEdit" name="textEdit">
<property name="geometry">
<rect>
<x>30</x>
<y>230</y>
<width>331</width>
<height>131</height>
</rect>
</property>
<property name="readOnly">
<bool>false</bool>
</property>
</widget>
<widget class="QComboBox" name="comboBox_baud">
<property name="geometry">
<rect>
<x>100</x>
<y>70</y>
<width>141</width>
<height>31</height>
</rect>
</property>
</widget>
<widget class="QLabel" name="label_2">
<property name="geometry">
<rect>
<x>10</x>
<y>70</y>
<width>81</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Baud rate:</string>
</property>
</widget>
<widget class="QPushButton" name="btnNewWindow">
<property name="enabled">
<bool>false</bool>
</property>
<property name="geometry">
<rect>
<x>10</x>
<y>170</y>
<width>111</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Lidar Window</string>
</property>

```

```

</widget>
<widget class="QPushButton" name="pushButton">
<property name="geometry">
<rect>
<x>130</x>
<y>170</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Scan Ports</string>
</property>
</widget>
<widget class="QLabel" name="label_3">
<property name="geometry">
<rect>
<x>30</x>
<y>0</y>
<width>191</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Lidar Com Port settings:</string>
</property>
</widget>
<widget class="QLabel" name="label_4">
<property name="geometry">
<rect>
<x>340</x>
<y>0</y>
<width>191</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Robot Com Port settings:</string>
</property>
</widget>
<widget class="QLabel" name="label_5">
<property name="geometry">
<rect>
<x>300</x>
<y>30</y>
<width>81</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Com Port:</string>
</property>
</widget>
<widget class="QLabel" name="label_6">
<property name="geometry">
<rect>
<x>300</x>
<y>70</y>
<width>81</width>
<height>31</height>
</rect>

```

```

</property>
<property name="text">
<string>Baud rate:</string>
</property>
</widget>
<widget class="QPushButton" name="RobotPortOpenButton">
<property name="geometry">
<rect>
<x>300</x>
<y>110</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Open</string>
</property>
</widget>
<widget class="QPushButton" name="RobotPortCloseButton">
<property name="enabled">
<bool>false</bool>
</property>
<property name="geometry">
<rect>
<x>420</x>
<y>110</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Close</string>
</property>
</widget>
<widget class="QComboBox" name="RobotPortComboBox">
<property name="geometry">
<rect>
<x>390</x>
<y>30</y>
<width>141</width>
<height>31</height>
</rect>
</property>
</widget>
<widget class="QComboBox" name="RobotBaudComboBox">
<property name="geometry">
<rect>
<x>390</x>
<y>70</y>
<width>141</width>
<height>31</height>
</rect>
</property>
</widget>
</widget>
<widget class="QStatusBar" name="statusBar"/>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>

```

Załacznik nr 3 (lidarwindow.ui)

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>lidarwindow</class>
<widget class="QMainWindow" name="lidarwindow">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>892</width>
<height>649</height>
</rect>
</property>
<property name="windowTitle">
<string>Lidar Window</string>
</property>
<widget class="QWidget" name="centralwidget">
<widget class="QPushButton" name="btnDevInfo">
<property name="geometry">
<rect>
<x>530</x>
<y>30</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Refresh Info</string>
</property>
</widget>
<widget class="QLabel" name="label_2">
<property name="geometry">
<rect>
<x>10</x>
<y>0</y>
<width>68</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Model:</string>
</property>
</widget>
<widget class="QLabel" name="label_serial">
<property name="geometry">
<rect>
<x>380</x>
<y>0</y>
<width>271</width>
<height>41</height>
</rect>
</property>
<property name="text">
<string>TextLabel</string>
</property>
</widget>
<widget class="QLabel" name="label_health">
<property name="geometry">
<rect>
```

```

<x>350</x>
<y>30</y>
<width>68</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>TextLabel</string>
</property>
</widget>
<widget class="QLabel" name="label_firmware">
<property name="geometry">
<rect>
<x>90</x>
<y>20</y>
<width>68</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>TextLabel</string>
</property>
</widget>
<widget class="QLabel" name="label_6">
<property name="geometry">
<rect>
<x>290</x>
<y>30</y>
<width>68</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Health:</string>
</property>
</widget>
<widget class="QLabel" name="label_4">
<property name="geometry">
<rect>
<x>10</x>
<y>40</y>
<width>81</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Hardware:</string>
</property>
</widget>
<widget class="QLabel" name="label_3">
<property name="geometry">
<rect>
<x>10</x>
<y>20</y>
<width>71</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Firmware:</string>
</property>

```

```

</widget>
<widget class="QLabel" name="label_5">
<property name="geometry">
<rect>
<x>290</x>
<y>0</y>
<width>91</width>
<height>41</height>
</rect>
</property>
<property name="text">
<string>Serial num:</string>
</property>
</widget>
<widget class="QLabel" name="label">
<property name="geometry">
<rect>
<x>70</x>
<y>0</y>
<width>141</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>TextLabel</string>
</property>
</widget>
<widget class="QLabel" name="label_hardware">
<property name="geometry">
<rect>
<x>90</x>
<y>40</y>
<width>68</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>TextLabel</string>
</property>
</widget>
<widget class="QPushButton" name="btnMotorStart">
<property name="geometry">
<rect>
<x>10</x>
<y>580</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Start Motor</string>
</property>
</widget>
<widget class="QPushButton" name="btnStopMotor">
<property name="geometry">
<rect>
<x>10</x>
<y>610</y>
<width>101</width>
<height>31</height>
</rect>

```

```

</property>
<property name="text">
  <string>Stop Motor</string>
</property>
</widget>
<widget class="QPushButton" name="btn_f1">
<property name="geometry">
  <rect>
    <x>120</x>
    <y>580</y>
    <width>101</width>
    <height>31</height>
  </rect>
</property>
<property name="text">
  <string>Start Scan</string>
</property>
</widget>
<widget class="QPushButton" name="btn_f2">
<property name="geometry">
  <rect>
    <x>120</x>
    <y>610</y>
    <width>101</width>
    <height>31</height>
  </rect>
</property>
<property name="text">
  <string>Stop Scan</string>
</property>
</widget>
<widget class="QGraphicsView" name="mapView">
<property name="geometry">
  <rect>
    <x>5</x>
    <y>70</y>
    <width>650</width>
    <height>500</height>
  </rect>
</property>
</widget>
<widget class="QSlider" name="motor_speed_slider">
<property name="geometry">
  <rect>
    <x>329</x>
    <y>620</y>
    <width>151</width>
    <height>26</height>
  </rect>
</property>
<property name="maximum">
  <number>1000</number>
</property>
<property name="singleStep">
  <number>5</number>
</property>
<property name="pageStep">
  <number>50</number>
</property>
<property name="value">
  <number>300</number>

```

```

</property>
<property name="orientation">
<enum>Qt::Horizontal</enum>
</property>
</widget>
<widget class="QLabel" name="motor_speed_label">
<property name="geometry">
<rect>
<x>480</x>
<y>620</y>
<width>61</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string/>
</property>
</widget>
<widget class="QLabel" name="label_7">
<property name="geometry">
<rect>
<x>230</x>
<y>620</y>
<width>101</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Motor Speed:</string>
</property>
</widget>
<widget class="QLabel" name="label_8">
<property name="geometry">
<rect>
<x>230</x>
<y>580</y>
<width>71</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Map Size:</string>
</property>
</widget>
<widget class="QLineEdit" name="map_size_text">
<property name="geometry">
<rect>
<x>310</x>
<y>580</y>
<width>61</width>
<height>31</height>
</rect>
</property>
<property name="maxLength">
<number>5</number>
</property>
<property name="cursorPosition">
<number>0</number>
</property>
</widget>
<widget class="QLabel" name="label_9">

```

```

<property name="geometry">
<rect>
<x>380</x>
<y>580</y>
<width>71</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Cell Size:</string>
</property>
</widget>
<widget class="QPushButton" name="map_create_button">
<property name="geometry">
<rect>
<x>540</x>
<y>580</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Create Map</string>
</property>
</widget>
<widget class="QLineEdit" name="cell_size_text">
<property name="geometry">
<rect>
<x>450</x>
<y>580</y>
<width>61</width>
<height>31</height>
</rect>
</property>
</widget>
<widget class="QPushButton" name="pushButton">
<property name="geometry">
<rect>
<x>670</x>
<y>590</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>PushButton</string>
</property>
</widget>
<widget class="QLabel" name="label_10">
<property name="geometry">
<rect>
<x>720</x>
<y>10</y>
<width>141</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Robot Telemetry:</string>
</property>
</widget>

```

```

<widget class="QLabel" name="telemetry_x">
<property name="geometry">
<rect>
<x>660</x>
<y>40</y>
<width>81</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>X:</string>
</property>
</widget>
<widget class="QLabel" name="telemetry_y">
<property name="geometry">
<rect>
<x>760</x>
<y>40</y>
<width>81</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Y:</string>
</property>
</widget>
<widget class="QLabel" name="telemetry_yaw">
<property name="geometry">
<rect>
<x>660</x>
<y>70</y>
<width>81</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Yaw:</string>
</property>
</widget>
<widget class="QLabel" name="telemetry_speed">
<property name="geometry">
<rect>
<x>760</x>
<y>70</y>
<width>121</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Speed:</string>
</property>
</widget>
<widget class="QLabel" name="label_11">
<property name="geometry">
<rect>
<x>670</x>
<y>520</y>
<width>81</width>
<height>21</height>
</rect>
</property>

```

```

<property name="text">
    <string>Test Point:</string>
</property>
</widget>
<widget class="QLabel" name="label_12">
<property name="geometry">
    <rect>
        <x>670</x>
        <y>550</y>
        <width>68</width>
        <height>21</height>
    </rect>
</property>
<property name="text">
    <string>X</string>
</property>
</widget>
<widget class="QLabel" name="label_13">
<property name="geometry">
    <rect>
        <x>760</x>
        <y>550</y>
        <width>68</width>
        <height>21</height>
    </rect>
</property>
<property name="text">
    <string>Y</string>
</property>
</widget>
<widget class="QLineEdit" name="point_x">
<property name="geometry">
    <rect>
        <x>690</x>
        <y>550</y>
        <width>51</width>
        <height>21</height>
    </rect>
</property>
</widget>
<widget class="QLineEdit" name="point_y">
<property name="geometry">
    <rect>
        <x>780</x>
        <y>550</y>
        <width>51</width>
        <height>21</height>
    </rect>
</property>
</widget>
<widget class="QCheckBox" name="simulate_checkbox">
<property name="geometry">
    <rect>
        <x>540</x>
        <y>610</y>
        <width>91</width>
        <height>26</height>
    </rect>
</property>
<property name="text">
    <string>Simulate</string>

```

```

</property>
<property name="tristate">
<bool>false</bool>
</property>
</widget>
<widget class="QLabel" name="label_14">
<property name="geometry">
<rect>
<x>660</x>
<y>130</y>
<width>71</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>X:</string>
</property>
</widget>
<widget class="QLabel" name="label_15">
<property name="geometry">
<rect>
<x>750</x>
<y>130</y>
<width>68</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Y:</string>
</property>
</widget>
<widget class="QLineEdit" name="set_pos_x">
<property name="geometry">
<rect>
<x>680</x>
<y>130</y>
<width>51</width>
<height>21</height>
</rect>
</property>
</widget>
<widget class="QLineEdit" name="set_pos_y">
<property name="geometry">
<rect>
<x>770</x>
<y>130</y>
<width>51</width>
<height>21</height>
</rect>
</property>
</widget>
<widget class="QLabel" name="label_16">
<property name="geometry">
<rect>
<x>660</x>
<y>100</y>
<width>111</width>
<height>21</height>
</rect>
</property>
<property name="text">

```

```

<string>Set position</string>
</property>
</widget>
<widget class="QLabel" name="label_17">
<property name="geometry">
<rect>
<x>660</x>
<y>190</y>
<width>71</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>X:</string>
</property>
</widget>
<widget class="QLabel" name="label_18">
<property name="geometry">
<rect>
<x>660</x>
<y>160</y>
<width>171</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Set destination point</string>
</property>
</widget>
<widget class="QLineEdit" name="set_end_x">
<property name="geometry">
<rect>
<x>680</x>
<y>190</y>
<width>51</width>
<height>21</height>
</rect>
</property>
</widget>
<widget class="QLabel" name="label_19">
<property name="geometry">
<rect>
<x>750</x>
<y>190</y>
<width>68</width>
<height>21</height>
</rect>
</property>
<property name="text">
<string>Y:</string>
</property>
</widget>
<widget class="QLineEdit" name="set_end_y">
<property name="geometry">
<rect>
<x>770</x>
<y>190</y>
<width>51</width>
<height>21</height>
</rect>
</property>

```

```

</widget>
<widget class="QPushButton" name="set_pos">
<property name="geometry">
<rect>
<x>830</x>
<y>130</y>
<width>51</width>
<height>20</height>
</rect>
</property>
<property name="text">
<string>Set</string>
</property>
</widget>
<widget class="QPushButton" name="set_end">
<property name="geometry">
<rect>
<x>830</x>
<y>190</y>
<width>51</width>
<height>20</height>
</rect>
</property>
<property name="text">
<string>Set</string>
</property>
</widget>
<widget class="QPushButton" name="solve_astar">
<property name="geometry">
<rect>
<x>660</x>
<y>230</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>A* solve</string>
</property>
</widget>
<widget class="QPushButton" name="btn_sendPath">
<property name="geometry">
<rect>
<x>780</x>
<y>230</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Send Path</string>
</property>
</widget>
<widget class="QPushButton" name="btn_TeleMode">
<property name="geometry">
<rect>
<x>660</x>
<y>270</y>
<width>101</width>
<height>31</height>
</rect>

```

```

</property>
<property name="text">
    <string>Tele Mode</string>
</property>
</widget>
<widget class="QPushButton" name="btn_AutoMode">
<property name="geometry">
    <rect>
        <x>780</x>
        <y>270</y>
        <width>101</width>
        <height>31</height>
    </rect>
</property>
<property name="text">
    <string>Auto Mode</string>
</property>
</widget>
<widget class="QPushButton" name="btn_AutoPause">
<property name="geometry">
    <rect>
        <x>660</x>
        <y>310</y>
        <width>101</width>
        <height>31</height>
    </rect>
</property>
<property name="text">
    <string>Auto Pause</string>
</property>
</widget>
<widget class="QPushButton" name="btn_AutoContinue">
<property name="geometry">
    <rect>
        <x>780</x>
        <y>310</y>
        <width>101</width>
        <height>31</height>
    </rect>
</property>
<property name="text">
    <string>A Continue</string>
</property>
</widget>
<widget class="QPushButton" name="btn_AutoNext">
<property name="geometry">
    <rect>
        <x>660</x>
        <y>350</y>
        <width>101</width>
        <height>31</height>
    </rect>
</property>
<property name="text">
    <string>Auto Next</string>
</property>
</widget>
<widget class="QPushButton" name="pushButton_3">
<property name="geometry">
    <rect>
        <x>780</x>

```

```
<y>350</y>
<width>101</width>
<height>31</height>
</rect>
</property>
<property name="text">
<string>Reset Pos</string>
</property>
</widget>
</widget>
</resources/>
<connections/>
</ui>
```

Załącznik nr 5 (mainwindow.h)

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>
#include <QThread>
#include "rplidar.h"
#include "lidarwindow.h"
#include "worker.h"
#include "map.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_OpenButton_clicked();
    void Timer100msElapsed();
    void on_CloseButton_clicked();
    void on_btnNewWindow_clicked();
    void on_pushButton_clicked();
    void on_RobotPortOpenButton_clicked();
    void on_RobotPortCloseButton_clicked();

signals:
    void doBlink();

private:
    Ui::MainWindow *ui;
    lidarwindow *lidar_window;
    QSerialPort serialPort, robotSerialPort;
    QTimer *timer100ms;
    RPLidar *rplidar;
    QThread *myThread;
    Worker *myWorker;
    bool State;
};

#endif // MAINWINDOW_H
```

Załącznik nr 6 (mainwindow.cpp)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QObject>
#include <QCoreApplication>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QTextStream>
#include <QProcess>
#include <QThread>
#include <QTimer>
#include <unistd.h>
#include <sys/types.h>
#include <QDebug>
#include "rplidar.h"
#include "worker.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new
Ui::MainWindow)
{
    ui->setupUi(this);

    rplidar = new RPLidar();

    foreach (const QSerialPortInfo &serialPortInfo,
              QSerialPortInfo::availablePorts())
    {
        ui->comboBox->addItem(serialPortInfo.portName());
        ui->RobotPortComboBox->addItem(serialPortInfo.portName());
    }
    ui->comboBox_baud->addItem(
        QStringLiteral("115200"), QSerialPort::Baud115200);
    ui->comboBox_baud->addItem(
        QStringLiteral("57600"), QSerialPort::Baud57600);
    ui->comboBox_baud->addItem(
        QStringLiteral("19200"), QSerialPort::Baud19200);

    ui->RobotBaudComboBox->addItem(
        QStringLiteral("115200"), QSerialPort::Baud115200);
    ui->RobotBaudComboBox->addItem(
        QStringLiteral("57600"), QSerialPort::Baud57600);
    ui->RobotBaudComboBox->addItem(
        QStringLiteral("19200"), QSerialPort::Baud19200);

    timer100ms = new QTimer(this);
    timer100ms->setInterval(1000);

    connect(timer100ms, SIGNAL(timeout()), this, SLOT(Timer100msElapsed()));

    myThread = new QThread(this);
    myWorker = new Worker();
    myWorker->moveToThread(myThread);
    State = false;
    myThread->start();
    connect(this, SIGNAL(doBlink()), myWorker, SLOT(blink()));
    ui->btnNewWindow->setEnabled(true);
}
```

```

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::Timer100msElapsed()
{
    qDebug() << "Timer";

    QByteArray arr = serialPort.readAll();
    QString str = "";
    qDebug() << arr.size();
    for(int i=0; i<arr.size(); i++)
    {
        str += QString::number(arr[i],16);
        str += " ";
    }
    ui->textEdit->setText(str);
}

void MainWindow::on_OpenButton_clicked()
{
    qDebug() << "Wciśnięto Open";
    serialPort.setPortName(ui->comboBox->currentText());
    serialPort.setBaudRate(ui->comboBox_baud->currentData().toInt());
    serialPort.setDataBits(QSerialPort::Data8);
    serialPort.setParity(QSerialPort::NoParity);
    serialPort.setStopBits(QSerialPort::OneStop);

    if (!serialPort.isOpen())
    {
        serialPort.open(QIODevice::ReadWrite);
        if (serialPort.isOpen())
        {
            ui->OpenButton->setEnabled(false);
            ui->comboBox->setEnabled(false);
            ui->comboBox_baud->setEnabled(false);
            ui->CloseButton->setEnabled(true);
            ui->btnNewWindow->setEnabled(true);
            serialPort.setDataTerminalReady(false);
            rplidar->setPort(&serialPort);
        }
    }
}

void MainWindow::on_CloseButton_clicked()
{
    rplidar->setMotorSpeed(0);
    QThread::msleep(2000);
    serialPort.close();
    if(!serialPort.isOpen())){
        ui->OpenButton->setEnabled(true);
        ui->CloseButton->setEnabled(false);
        ui->comboBox->setEnabled(true);
        ui->comboBox_baud->setEnabled(true);
        ui->btnNewWindow->setEnabled(false);
    }
}

```

```

        timer100ms->stop();

    }

void MainWindow::on_btnNewWindow_clicked()
{
    lidar_window = new lidarwindow(this);
    lidar_window->show();
    lidar_window->setDataPointer(rplidar);
    lidar_window->setPointers(myThread,myWorker);
}

void MainWindow::on_pushButton_clicked()
{
    ui->comboBox->clear();
    ui->RobotPortComboBox->clear();
    foreach (const QSerialPortInfo &serialPortInfo,
    QSerialPortInfo::availablePorts())
    {
        ui->comboBox->addItem(serialPortInfo.portName());
        ui->RobotPortComboBox->addItem(serialPortInfo.portName());
    }
}

void MainWindow::on_RobotPortOpenButton_clicked()
{
    robotSerialPort.setPortName(ui->RobotPortComboBox->currentText());
    robotSerialPort.setBaudRate(
                ui->RobotBaudComboBox->currentData().toInt());
    robotSerialPort.setDataBits(QSerialPort::Data8);
    robotSerialPort.setParity(QSerialPort::NoParity);
    robotSerialPort.setStopBits(QSerialPort::OneStop);

    if (!(robotSerialPort.isOpen()))
    {
        robotSerialPort.open(QIODevice::ReadWrite);
        if (robotSerialPort.isOpen())
        {
            ui->RobotPortOpenButton->setEnabled(false);
            ui->RobotPortComboBox->setEnabled(false);
            ui->RobotBaudComboBox->setEnabled(false);
            ui->RobotPortCloseButton->setEnabled(true);
            rplidar->setRobotPort(&robotSerialPort);
        }
    }
}

void MainWindow::on_RobotPortCloseButton_clicked()
{
    robotSerialPort.close();
    if(!(robotSerialPort.isOpen())){
        ui->RobotPortOpenButton->setEnabled(true);
        ui->RobotPortCloseButton->setEnabled(false);
        ui->RobotPortComboBox->setEnabled(true);
        ui->RobotBaudComboBox->setEnabled(true);
    }
}

```

Załącznik nr 7 (lidarwindow.h)

```
#ifndef LIDARWINDOW_H
#define LIDARWINDOW_H

#include <QMainWindow>
#include <QTimer>
#include <QDebug>
#include <QThread>
#include <QGraphicsScene>
#include <QMessageBox>
#include <QIntValidator>
#include <QDoubleValidator>
#include "rplidar.h"
#include "worker.h"
#include "map.h"

namespace Ui {
class lidarwindow;
}

class lidarwindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit lidarwindow(QWidget *parent = 0);
    void setDataPointer(RPLidar *ptr);
    void setPointers(QThread *tptr, Worker *wptr);
    ~lidarwindow();

public slots:
    void reCalcPath(void);

private slots:
    void Timer100msElapsed();

    void on_btnDevInfo_clicked();
    void on_btnMotorStart_clicked();
    void on_btnStopMotor_clicked();
    void on_btn_f1_clicked();
    void on_btn_f2_clicked();
    void on_motor_speed_slider_valueChanged(int value);
    void on_map_create_button_clicked();
    void on_pushButton_clicked();
    void measure_calculate();
    void on_set_pos_clicked();
    void on_set_end_clicked();
    void on_solve_astar_clicked();
}
```

```

void on_btn_sendPath_clicked();
void on_btn_TeleMode_clicked();
void on_btn_AutoMode_clicked();
void on_btn_AutoPause_clicked();
void on_btn_AutoContinue_clicked();
void on_btn_AutoNext_clicked();
void on_pushButton_3_clicked();

private:
Ui::lidarwindow *ui;
QTimer *timer100ms;
RPLidar *rplidar;
QThread *myThread;
Worker *myWorker;
QGraphicsScene *scene;
map* Map;
bool map_created;
int motor_speed;
};

#endif // LIDARWINDOW_H

```

Załącznik nr 7 (lidarwindow.cpp)

```
#include "lidarwindow.h"
#include "ui_lidarwindow.h"

lidarwindow::lidarwindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::lidarwindow)
{
    ui->setupUi(this);
    scene = new QGraphicsScene(this);
    ui->mapView->setScene(scene);
    map_created = false;

    timer100ms = new QTimer();
    timer100ms->setInterval(100);
    connect(timer100ms, SIGNAL(timeout()), this, SLOT(Timer100msElapsed()));
    timer100ms->start();

    motor_speed = ui->motor_speed_slider->value();
    ui->motor_speed_label->setText(QString::number(motor_speed));
    ui->map_size_text->setValidator(new QIntValidator(0,1000,this));
    ui->cell_size_text->setValidator(new QDoubleValidator(0,10,2,this));
    ui->point_x->setValidator(new QDoubleValidator(-10,10,4,this));
    ui->point_y->setValidator(new QDoubleValidator(-10,10,4,this));
    ui->set_pos_x->setValidator(new QDoubleValidator(-100,100,2,this));
    ui->set_pos_y->setValidator(new QDoubleValidator(-100,100,2,this));
    ui->set_end_x->setValidator(new QDoubleValidator(-100,100,2,this));
    ui->set_end_y->setValidator(new QDoubleValidator(-100,100,2,this));
    ui->map_size_text->setText("10");
    ui->cell_size_text->setText("0.1");

}

void lidarwindow::setDataPointer(RPLidar *ptr)
{
    rplidar = ptr;
}

void lidarwindow::setPointers(QThread *tpr, Worker *wptr)
{
    myThread = tpr;
    myWorker = wptr;
}

void lidarwindow::Timer100msElapsed()
{
    if (rplidar->ptr_robotSerialPort != 0) rplidar->getRobotTelemetry();
    e_deviceinfo dev_info = rplidar->pData->dev_info;
    e_devicehealth dev_health = rplidar->pData->dev_health;
    ui->label->setText("RPLidar A"+QString::number(dev_info.hardware));
    ui->label_firmware-
>setText(QString::number((u_int8_t)(dev_info.firmware>>8))+". "+QString::num
ber((u_int8_t)(dev_info.firmware)));
    ui->label_hardware->setText(QString::number(dev_info.hardware));

    QString str;
    for(int i=0; i<16; i++)
    {
        str += QString::number(dev_info.serial[i],16);
    }
}
```

```

}

ui->label_serial->setText(str);
switch (dev_health.dev_status) {
case 0:
    ui->label_health->setText("GOOD");
    break;
case 1:
    ui->label_health->setText("WARNING!");
    break;
case 2:
    ui->label_health->setText("ERROR!!");
CODE:+QString::number(dev_health.error_code));
default:
    break;
}

ui->telemetry_x->setText(
    "X: "+QString::number(rplidar->pData->roverpos.xpos,'f',3)+" m");
ui->telemetry_y->setText(
    "Y: "+QString::number(rplidar->pData->roverpos.ypos,'f',3)+" m");
ui->telemetry_yaw->setText(
    "Yaw: "+QString::number(rplidar->pData->roverpos.yaw,'f',2)+" ");
ui->telemetry_speed->setText(
"Speed: "+QString::number(rplidar->pData->roverpos.speed,'f',2)+" m/s");

if(map_created) Map->actualizeRoverPos(
    rplidar->pData->roverpos.xpos,rplidar->pData->roverpos.ypos);
}

lidarwindow::~lidarwindow()
{
    delete ui;
}

void lidarwindow::on_btnDevInfo_clicked()
{
    qDebug()<<rplidar->getDeviceInfo().hardware;
    qDebug()<<rplidar->getDeviceHealth().dev_status;
}

void lidarwindow::on_btnMotorStart_clicked()
{
    rplidar->setMotorSpeed(motor_speed);
}

void lidarwindow::on_btnStopMotor_clicked()
{
    rplidar->setMotorSpeed(0);
}

void lidarwindow::on_btn_f1_clicked()
{
    rplidar->startExpressScan();
    rplidar->getResponseDescriptor(true);
    connect(rplidar->ptr_serialPort,SIGNAL(
        readyRead()),rplidar,SLOT(getExpressData()));

connect(rplidar,SIGNAL(measureComplete()),this,SLOT(measure_calculate()));
}

```

```

void lidarwindow::on_btn_f2_clicked()
{
    disconnect(rplidar->ptr_serialPort, SIGNAL(
        readyRead()), rplidar, SLOT(getExpressData()));

    disconnect(rplidar, SIGNAL(measureComplete()), this, SLOT(measure_calculate()));
}

rplidar->sendCommand(STOP);
rplidar->getResponseDescriptor(true);
}

void lidarwindow::on_motor_speed_slider_valueChanged(int value)
{
    motor_speed = value;
    rplidar->setMotorSpeed(motor_speed);
    ui->motor_speed_label->setText(QString::number(motor_speed));
}

void lidarwindow::on_map_create_button_clicked()
{
    int mapSize = ui->map_size_text->text().toInt();
    float cellSize = ui->cell_size_text->text().toFloat();

    Map = new map(mapSize,cellSize,ui->simulate_checkbox->isChecked());

    Map->drawMap(scene);
    Map->createSafeZone(true);
    Map->createSafeZone(true);
    ui->mapView->fitInView(scene->itemsBoundingRect(),Qt::KeepAspectRatio);
    map_created = true;

    connect(Map, SIGNAL(reCalculatePath()),this,SLOT(reCalcPath()));

    QMessageBox msg;
    msg.setText("Map created!");
    msg.exec();
    qDebug() << "Map created!";
}

void lidarwindow::on_pushButton_clicked()
{
    e_singlepoint t_point;
    t_point.xpos = ui->point_x->text().toFloat();
    t_point.ypos = ui->point_y->text().toFloat();

    Map->actualizeCell(Map->getCellPos(t_point),obstacle);
}

void lidarwindow::measure_calculate()
{
    for (int i=0; i<32; i++) {
        if (rplidar->pData->data_express.measure[i].distance != 0)
            Map->actualizeCell(Map->getCellPos(rplidar->fromPolarToCartesian(
                rplidar->pData->data_express.measure[i])),obstacle);
    }
}

```

```

void lidarwindow::on_set_pos_clicked()
{
    e_singlepoint tmp_point;
    tmp_point.xpos = ui->set_pos_x->text().toFloat();
    tmp_point.ypos = ui->set_pos_y->text().toFloat();

    //qDebug() << "ok";
    if (Map->simulate_mode) {
        rplidar->pData->roverpos.xpos = tmp_point.xpos;
        rplidar->pData->roverpos.ypos = tmp_point.ypos;

        e_cell_pos tmp_cell;
        tmp_cell = Map->getCellPos(tmp_point);
        Map->actualizeCell(Map->start_cell, no_info);
        Map->start_cell = tmp_cell;
        Map->actualizeCell(Map->start_cell, start);
    }
}

void lidarwindow::on_set_end_clicked()
{
    e_singlepoint tmp_point;
    tmp_point.xpos = ui->set_end_x->text().toFloat();
    tmp_point.ypos = ui->set_end_y->text().toFloat();

    e_cell_pos tmp_cell=Map->getCellPos(tmp_point);
    Map->actualizeCell(Map->end_cell, no_info);
    Map->end_cell = tmp_cell;
    Map->actualizeCell(Map->end_cell, end);
}

void lidarwindow::on_solve_astar_clicked()
{
    if (Map->simulate_mode == false) {
        e_singlepoint tmp_point;
        tmp_point.xpos = rplidar->pData->roverpos.xpos;
        tmp_point.ypos = rplidar->pData->roverpos.ypos;

        e_cell_pos tmp_cell;
        tmp_cell = Map->getCellPos(tmp_point);
        Map->actualizeCell(Map->start_cell, no_info);
        Map->start_cell = tmp_cell;
        Map->actualizeCell(Map->start_cell, start);
        Map->openSet.append(Map->getCellFromIndex(Map->start_cell));
    }
    Map->aStarSolver();
    if (Map->path.length() > 0) {
        Map->preparePathFrame();
    }
}

void lidarwindow::reCalcPath()
{
    rplidar->sendRoboCommand(cmd_autopause);
    on_set_end_clicked();
    on_solve_astar_clicked();
    on_btn_sendPath_clicked();
    if (Map->bytes_to_send > 0) rplidar->sendRoboCommand(cmd_autostart);
}

```

```

void lidarwindow::on_btn_sendPath_clicked()
{
    if (Map->bytes_to_send > 0) rplidar->ptr_robotSerialPort-
>write((char*)Map->frame_buf,Map->bytes_to_send);
}

void lidarwindow::on_btn_TeleMode_clicked()
{
    rplidar->sendRoboCommand(cmd_wifi);
}

void lidarwindow::on_btn_AutoMode_clicked()
{
    rplidar->sendRoboCommand(cmd_autostart);
}

void lidarwindow::on_btn_AutoPause_clicked()
{
    rplidar->sendRoboCommand(cmd_autopause);
}

void lidarwindow::on_btn_AutoContinue_clicked()
{
    rplidar->sendRoboCommand(cmd_autocontinue);
}

void lidarwindow::on_btn_AutoNext_clicked()
{
    rplidar->sendRoboCommand(cmd_autonextpoint);
}

void lidarwindow::on_pushButton_3_clicked()
{
    rplidar->sendRoboCommand(cmd_resetpos);
}

```

Załącznik nr 8 (rplidar.h)

```
#ifndef RPLIDAR_H
#define RPLIDAR_H

#include <QObject>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QtMath>

#define START_FLAG 0xA5
#define STOP 0x25
#define RESET 0x40
#define SCAN 0x20
#define EXPRESS_SCAN 0x82
#define FORCE_SCAN 0x21
#define GET_INFO 0x50
#define GET_HEALTH 0x52
#define GET_SAMPLERATE 0x59
#define GET_LIDAR_CONF 0x84

#define u_int8_t uint8_t
#define u_int16_t uint16_t
#define u_int32_t uint32_t

enum packed
{
    SYNC_FLAG =0,
    CMD =1
};

enum e_frame_type
{
    frametype_cmd = 10,
    frametype_header = 155,
    frametype_robo_pos = 156,
};

enum e_frame_cmd{
    cmd_null = 0,
    cmd_wifi = 2,
    cmd_autostart = 3,
    cmd_autopause = 9,
    cmd_autocontinue = 10,
    cmd_autonextpoint = 11,
    cmd_resetpos = 12
};

typedef struct
{
    u_int8_t model;
    u_int16_t firmware;
    u_int8_t hardware;
    u_int8_t serial[16];
}e_deviceinfo;

typedef struct
{
    u_int8_t dev_status;
    u_int16_t error_code;
```

```

} e_devicehealth;

typedef struct
{
    u_int16_t distance;      // [mm]
    float angle;           // [']
} e_singlemeasure;

typedef struct
{
    float xpos;
    float ypos;
} e_singlepoint;

typedef struct
{
    float last_start_angle;
    float start_angle;
    float delta_angle;
    bool new_scan;
    e_singlemeasure measure[32];
} e_expressdata;

typedef struct
{
    float xpos;
    float ypos;
    float yaw;
    float speed;
} e_roverpos;

typedef struct
{
    e_deviceinfo dev_info;
    e_devicehealth dev_health;
    e_expressdata data_express;
    e_roverpos roverpos;
    bool data_error;
} e_data;

class RPLidar : public QObject
{
    Q_OBJECT
public:
    e_data *pData;
    QSerialPort *ptr_serialPort, *ptr_robotSerialPort;
    u_int8_t cmd_buf[5];
    explicit RPLidar(QObject *parent = 0);
    void sendCommand(u_int8_t cmd);
    void sendCommandPayload(u_int8_t cmd, u_int8_t* payload, u_int8_t size);
    void setPort(QSerialPort *port);
    void setRobotPort(QSerialPort *roboPort);
    void setMotorSpeed(u_int16_t speed);
    void startExpreeessScan(void);
    void getResponseDescriptor(bool on_term);
    void sendRoboCommand(e_frame_cmd cmd);
    u_int16_t crc16(u_int8_t* packet, u_int32_t nBytes);
    e_deviceinfo getDeviceInfo();
}

```

```
    e_devicehealth getDeviceHealth();  
    e_singlepoint fromPolarToCartesian(e_singlemeasure measure);  
  
signals:  
    void measureComplete();  
  
public slots:  
    void getData(void);  
    void getExpressData(void);  
    void getRobotTelemetry(void);  
  
private:  
};  
  
#endif // RPLIDAR_H
```

Załącznik nr 9 (rplidar.cpp)

```
#include "rplidar.h"
#include <QThread>
#include <QDebug>

RPLidar::RPLidar(QObject *parent) :
    QObject(parent)
{
    ptr_robotSerialPort = 0;
    pData = new e_data;
    pData->dev_info.firmware=0;
    pData->dev_info.hardware=0;
    pData->dev_info.model=0;
    pData->roverpos.xpos=0;
    pData->roverpos.ypos=0;
    pData->roverpos.yaw=0;
    pData->roverpos.speed=0;
}

void RPLidar::setPort(QSerialPort *port)
{
    ptr_serialPort = port;
}

void RPLidar::setRobotPort(QSerialPort *roboPort)
{
    ptr_robotSerialPort = roboPort;
}

u_int16_t RPLidar::crc16(u_int8_t *packet, u_int32_t nBytes)
{
    u_int16_t crc = 0;
    for(u_int32_t byte=0; byte < nBytes; byte++)
    {
        crc =crc ^ ((u_int16_t)packet[byte]<<8);
        for (u_int8_t bit = 0; bit < 8; bit++)
        {
            if(crc & 0x8000) crc = (crc << 1) ^ 0x1021;
            else crc = crc<<1;
        }
    }
    return crc;
}

void RPLidar::sendCommand(u_int8_t cmd)
{
    u_int8_t buff[2];
    buff[SYNC_FLAG]=START_FLAG;
    buff[CMD]=cmd;
    ptr_serialPort->write((char*)buff,2);
}

void RPLidar::sendCommandPayload(u_int8_t cmd, u_int8_t* payload, u_int8_t size)
{
    u_int8_t buff[20];
    buff[0]=START_FLAG;
```

```

buff[1]=cmd;
buff[2]=size;
for (int i=0; i<size; i++) {
    buff[3+i]= *payload;
    payload++;
}
u_int8_t checksum = 0;

for(u_int8_t i=0; i<3+size; i++) {
    checksum ^= buff[i];
}

buff[3+size]=checksum;
ptr_serialPort->write((char*)buff,4+size);

}

void RPLidar::startExpressScan(void)
{
    getDeviceInfo();
    getDeviceHealth();
    QThread::msleep(100);
    u_int8_t payload[5]={0, 0, 0, 0, 0};
    sendCommandPayload(EXPRESS_SCAN,payload,5);
}

void RPLidar::setMotorSpeed(u_int16_t speed)
{
    u_int8_t buff[10];
    buff[0]=START_FLAG;
    buff[1]=0xF0;
    buff[2]=0x02;
    buff[3]=(u_int8_t)speed;
    buff[4]=(u_int8_t)(speed>>8);
    u_int8_t checksum = 0;

    for(u_int8_t i=0; i<5; i++) {
        checksum ^= buff[i];
    }
    buff[5]=checksum;
    ptr_serialPort->write((char*)buff,6);
}

e_deviceinfo RPLidar::getDeviceInfo()
{
    QByteArray buff;
    e_deviceinfo tmp_dev_info;
    sendCommand(GET_INFO);

    if (ptr_serialPort->waitForReadyRead(1000)) {
        buff = ptr_serialPort->readAll();
    }

    if(((u_int8_t)buff[0] == 0xA5) && ((u_int8_t)buff[1] == 0x5A))
    {
        tmp_dev_info.model = buff[7];
        tmp_dev_info.firmware = buff[8]+(buff[9]<<8);
        tmp_dev_info.hardware = buff[10];

        for( u_int8_t i=0; i <16; i++)

```

```

        {
            tmp_dev_info.serial[i]=buff[11+i];
        }
        pData->data_error = false;
    }
    else
    {
        pData->data_error = true;
    }

    pData->dev_info = tmp_dev_info;
    return tmp_dev_info;
}

e_devicehealth RPLidar::getDeviceHealth()
{
    sendCommand(GET_HEALTH);
    QByteArray buff;
    e_devicehealth tmp_devhealth;
    if (ptr_serialPort->waitForReadyRead(1000)){
        buff = ptr_serialPort->readAll();
    }

    if(((u_int8_t)buff[0] == 0xA5) && ((u_int8_t)buff[1] == 0x5A))
    {
        tmp_devhealth.dev_status = buff[7];
        tmp_devhealth.error_code = buff[8] + (buff[9]<<8);
        pData->data_error = false;
    }
    else
    {
        pData->data_error = true;
    }

    pData->dev_health = tmp_devhealth;
    return tmp_devhealth;
}

void RPLidar::getResponseDescriptor(bool on_term)
{
    if(ptr_serialPort->waitForReadyRead(1000)){
        QByteArray arr = ptr_serialPort->readAll();
        if (on_term==true){
            QString str = "";
            for(int i=0; i<arr.size(); i++)
            {
                str += QString::number(arr[i],16);
                str += " ";
            }
            qDebug()<<str;
        }
    }
}

void RPLidar::getData(void)
{
    while(ptr_serialPort->bytesAvailable()>=5) {
        QString str = "";
        char buff;
        for(int i=0; i<5; i++)
        {
            ptr_serialPort->getChar(&buff);

```

```

        str += QString::number((u_int8_t)buff,16);
        str += " ";
    }
    qDebug()<<str;
}
ptr_serialPort->readAll();
}

void RPLidar::getExpressData()
{
    int bytes_to_read = ptr_serialPort->bytesAvailable();
    QByteArray arr = ptr_serialPort->readAll();
    //qDebug()<<bytes_to_read<<" "<<arr.size();
    u_int8_t multi_packed=bytes_to_read/84;
    for(u_int8_t j=0; j<multi_packed;j++)
    {
        if(((arr[0+j*84]>>4)==0xA)&&((arr[1+j*84]>>4)==0x5))
        {
            //qDebug()<<"Packet ok!";
            pData->data_express.start_angle = float((u_int16_t)(arr[2+j*84]
+ ((arr[3+j*84]&0x7F)<<8))/64.0f;
            pData->data_express.delta_angle =
            pData->data_express.start_angle - pData->data_express.last_start_angle;
            if (pData->data_express.delta_angle < 0)
                pData->data_express.delta_angle = 360.0f + pData->data_express.delta_angle;
            pData->data_express.last_start_angle =
            pData->data_express.start_angle;
            bool newscan = arr[3+j*84] & (1<<7);
            //qDebug()<<pData->data_express.start_angle<<" "<<pData-
>data_express.delta_angle<<" "<<newscan;
            for (u_int8_t i=0; i<16; i++)
            {
                u_int16_t dist1 =
                    (arr[4+i*5+j*84]>>2)+(arr[5+i*5+j*84]<<6);
                u_int16_t dist2 =
                    (arr[6+i*5+j*84]>>2)+(arr[7+i*5+j*84]<<6);
                float dfil = (float)((int8_t)((u_int8_t)
(arr[8+i*5+j*84]&0x0F)+((u_int8_t)(arr[4+i*5+j*84]&0x03)<<4))<<2))/32.0f;
                float dfi2 = (float)((int8_t)((u_int8_t)
(arr[8+i*5+j*84]>>4)+((u_int8_t)(arr[6+i*5+j*84]&0x03)<<4))<<2))/32.0f;
                float fil = pData->data_express.start_angle + (pData-
>data_express.delta_angle/32.0f)*(2*i+1)-dfil;
                float fi2 = pData->data_express.start_angle + (pData-
>data_express.delta_angle/32.0f)*(2*i+2)-dfi2;
                //qDebug()<<dist1<<" "<<fil<<" "<<dfil<<" "<<dist2<<
                "<<fi2<<" "<<dfi2;
                pData->data_express.measure[i*2].distance = dist1;
                pData->data_express.measure[i*2].angle = fil;
                pData->data_express.measure[i*2+1].distance = dist2;
                pData->data_express.measure[i*2+1].angle = fi2;
            }
            emit measureComplete();
        }
    }
}

void RPLidar::getRobotTelemetry()
{
    QByteArray arr = ptr_robotSerialPort->readAll();
    u_int8_
```

```

        for(int i=0;i<arr.size();i++)
    {
        buf[i] = static_cast<u_int8_t>(arr[i]);
        //qDebug()<<buf[i];
    }
    if(buf[0] == frametype_header)
    {
        if(buf[1]==frametype_robo_pos) {
            u_int16_t crc1 = crc16(buf,18);
            u_int16_t crc2 = ((u_int16_t)buf[18]<<8)+buf[19];
            if(crc1==crc2)
            {
                //qDebug()<<"Get telemetry!";
                pData->roverpos.speed=static_cast<double>(((int32_t)((u_int32_t)buf[2]<<24)
                + ((u_int32_t)buf[3]<<16) + ((u_int32_t)buf[4]<<8) + (u_int32_t)buf[5])) /
                1000.0;
                pData->roverpos.yaw=static_cast<double>(((int32_t)((u_int32_t)buf[6]<<24) +
                ((u_int32_t)buf[7]<<16) + ((u_int32_t)buf[8]<<8) + (u_int32_t)buf[9])) /
                1000.0;
                pData->roverpos.xpos=static_cast<double>(((int32_t)((u_int32_t)buf[10]<<24)
                + ((u_int32_t)buf[11]<<16) + ((u_int32_t)buf[12]<<8) + (u_int32_t)buf[13])) /
                1000000.0;
                pData->roverpos.ypos=static_cast<double>(((int32_t)((u_int32_t)buf[14]<<24)
                + ((u_int32_t)buf[15]<<16) + ((u_int32_t)buf[16]<<8) + (u_int32_t)buf[17])) /
                1000000.0;
            }
        }
    }
}

void RPLidar::sendRoboCommand(e_frame_cmd cmd) {
    cmd_buf[0]=frametype_header;
    cmd_buf[1]=frametype_cmd;
    cmd_buf[2]=cmd;

    u_int16_t crc=crc16(cmd_buf,3);
    cmd_buf[3] = (u_int8_t)(crc>>8);
    cmd_buf[4] = (u_int8_t) crc;

    ptr_robotSerialPort->write((char*)cmd_buf,5);
}

e_singlepoint RPLidar::fromPolarToCartesian(e_singlemeasure measure)
{
    e_singlepoint tmp_point;

    tmp_point.xpos=pData->roverpos.xpos+measure.distance*qSin(
        qDegreesToRadians(measure.angle+pData->roverpos.yaw))/1000.0;
    tmp_point.ypos=pData->roverpos.ypos+measure.distance*qCos(
        qDegreesToRadians(measure.angle+pData->roverpos.yaw))/1000.0;
    //qDebug()<<"X: "<

```

Załącznik nr 10 (map.h)

```
#ifndef MAP_H
#define MAP_H

#include <QObject>
#include <QDebug>
#include <QGraphicsScene>
#include <QGraphicsRectItem>
#include "rplidar.h"

#define BUF_LEN 1000

typedef enum {
    centered,
    left_bottom_corner
} e_map_type;

typedef enum {
    angle_0,
    angle_45,
    angle_90,
    angle_125,
    angle_err
} e_path_angle;

typedef enum {
    no_info,
    clear,
    obstacle,
    start,
    end,
    open_set,
    close_set,
    path_cell,
    path_to_send,
    safe_zone_cell
} e_cell_type;

typedef struct {
    u_int16_t x_index;
    u_int16_t y_index;
} e_cell_pos;

typedef struct {

    e_cell_pos pos;
    float x_pos;
    float y_pos;

    float f;
    float g;
    float h;

    e_cell_type cell_type;
    e_cell_pos parent;
    bool have_parent;
}

e_map_cell;
```

```

class map : public QObject
{
    Q_OBJECT

private:

public:
    int map_size, num_of_cells, bytes_to_send;
    float cell_size, shift_coef;
    bool simulate_mode;
    e_map_cell **map_cell;
    e_cell_pos start_cell, end_cell;
    u_int8_t frame_buf[BUF_LEN];

    QVector<e_map_cell> openSet;
    QVector<e_map_cell> closeSet;
    QVector<e_map_cell> path;

    QGraphicsRectItem ***cell;
    QGraphicsEllipseItem *rover_pos;

    explicit map(int mapSize = 10, float cellSize = 0.25, bool simulate =
false, e_map_type mapType = centered, QObject *parent = 0);
    e_cell_pos getCellPos(e_singlepoint point);
    e_map_cell getCellFromIndex(e_cell_pos pos);
    QVector<e_map_cell*> getNeighbours(e_cell_pos pos);
    e_path_angle getAngle(e_map_cell fr_p, e_map_cell to_p);
    u_int16_t crc16(u_int8_t* packet, u_int32_t nBytes);
    bool isInclude(QVector<e_map_cell> container, e_map_cell element);
    float heuristic(e_map_cell to_point, e_map_cell from_point);
    void drawMap(QGraphicsScene *scene);
    void clearAStar(void);
    void createSafeZone(bool expand);

    void actualizeCell(e_cell_pos pos, e_cell_type type);
    void preparePathFrame(void);

signals:
    void reCalculatePath(void);

public slots:
    void actualizeRoverPos(float xpos, float ypos);
    void aStarSolver(void);

};

#endif // MAP_H

```

Załącznik nr 10 (map.cpp)

```
#include "map.h"

map::map(int mapSize, float cellSize, bool simulate, e_map_type mapType,
QObject *parent) :
    QObject(parent)
{
    map_size = mapSize;
    cell_size = cellSize;
    simulate_mode = simulate;
    num_of_cells = (mapSize / cellSize) +1;
    map_cell = new e_map_cell*[num_of_cells];
    for (int i=0; i< num_of_cells; i++){
        map_cell[i] = new e_map_cell[num_of_cells];
    }
    if (mapType == centered){
        shift_coef = (num_of_cells-1)*cell_size/2.0;
    }else{
        shift_coef = 0;
    }
    for (int x=0; x< num_of_cells; x++){
        for (int y=0; y<num_of_cells; y++) {
            map_cell[x][y].x_pos=(float)(x*cell_size - shift_coef);
            map_cell[x][y].y_pos=(float)(y*cell_size - shift_coef);
            map_cell[x][y].pos.x_index=x;
            map_cell[x][y].pos.y_index=y;
            map_cell[x][y].g = 0;
            map_cell[x][y].f = 0;
            map_cell[x][y].h =0;
            map_cell[x][y].have_parent = false;
            map_cell[x][y].cell_type=no_info;
            if (simulate_mode){
                if ((qrand()%100)<1) map_cell[x][y].cell_type = obstacle;
            }
            // qDebug() << "Cell x:" << x << " y:" << y << " Pos:
            x:<<map_cell[x][y].x_pos << " m y:<<map_cell[x][y].y_pos << " m";
        }
    }
    qDebug() << "Cells in row: " << num_of_cells << "Total
cells:" << num_of_cells * num_of_cells <<
Size:" << num_of_cells * num_of_cells * sizeof(e_map_cell) / 1024 << "KB Shift Coef:
" << shift_coef / cell_size;

    end_cell.x_index = 0;
    end_cell.y_index = 0;
    for (int i=0; i< BUF_LEN; i++) {
        frame_buf[i]=0;
    }
    bytes_to_send = 0;
}

e_cell_pos map::getCellPos(e_singlepoint point)
{
    e_cell_pos tmp_cell;
    int num_x = point.xpos/cell_size;
    float mod_x = fmod(point.xpos,cell_size);
    if (mod_x>cell_size/2) num_x++;
    else if (mod_x < -1*cell_size/2) num_x--;
    int num_y = point.ypos/cell_size;
```

```

        float mod_y = fmod(point.ypos,cell_size);
        if (mod_y > cell_size/2) num_y++;
        else if (mod_y < -1*cell_size/2) num_y--;
        tmp_cell.x_index = num_x + shift_coef/cell_size;
        tmp_cell.y_index = num_y + shift_coef/cell_size;

        if (tmp_cell.x_index >= num_of_cells) tmp_cell.x_index =
                                         num_of_cells-1;
        if (tmp_cell.y_index >= num_of_cells) tmp_cell.y_index =
                                         num_of_cells-1;
        //qDebug()<<"num x:"<<num_x<<" y:"<<num_y<<" mod: x:"<<mod_x<<" y:
        " <<mod_y;
        //qDebug()<<"Cell: X:"<<tmp_cell.x_index<<" Y:"<<tmp_cell.y_index<<" 
Pos: x:"<<point.xpos<<" y:"<<point.ypos;
        return tmp_cell;
    }

void map::drawMap(QGraphicsScene *scene)
{
    scene->clear();
    cell = new QGraphicsRectItem**[num_of_cells];
    for (int i=0; i<num_of_cells; i++){
        cell[i]= new QGraphicsRectItem*[num_of_cells];
    }
    for (int i=0; i<num_of_cells; i++){
        for(int j=0; j<num_of_cells; j++){
            cell[i][j]=scene->addRect(map_cell[i][j].x_pos*100,map_cell[i][j].y_pos*-
100,cell_size*100, cell_size*100,QPen(Qt::black),QBrush(Qt::gray));
            if (map_cell[i][j].cell_type == obstacle)
                cell[i][j]->setBrush(QBrush(Qt::darkBlue));
        }
    }
    start_cell.x_index = num_of_cells/2;
    start_cell.y_index = num_of_cells/2;
    actualizeCell(start_cell, start);

    scene->addLine(map_cell[0][0].x_pos*100-
100*cell_size,cell_size*50,map_cell[num_of_cells-
1][0].x_pos*100+200*cell_size,cell_size*50,QPen(Qt::darkRed));
    scene->addLine(cell_size*50,map_cell[0][0].y_pos*-
100+200*cell_size,cell_size*50,map_cell[0][num_of_cells-1].y_pos*-100-
100*cell_size,QPen(Qt::darkRed));
    rover_pos = scene->addEllipse(
        0,0,10,10,QPen(Qt::black),QBrush(Qt::black));
}

void map::actualizeCell(e_cell_pos pos, e_cell_type type){
    QBrush brush(Qt::black);

    if (type == obstacle) brush.setColor(Qt::darkBlue);
    else if (type == start) brush.setColor(Qt::green);
    else if (type == end) brush.setColor(Qt::red);
    else if (type == clear) brush.setColor(Qt::white);
    else if (type == open_set) brush.setColor(Qt::darkGreen);
    else if (type == close_set) brush.setColor(Qt::darkCyan);
    else if (type == path_cell) brush.setColor(Qt::darkRed);
    else if (type == path_to_send) brush.setColor(QColor(168,0,0));
    else if (type == safe_zone_cell) brush.setColor(QColor(0,0,168));
    else brush.setColor(Qt::gray);
}

```

```

e_cell_type tmptype = map_cell[pos.x_index][pos.y_index].cell_type;
cell[pos.x_index][pos.y_index]->setBrush(brush);
map_cell[pos.x_index][pos.y_index].cell_type = type;

if (((tmptype == path_cell) || (tmptype == path_to_send)) && (type ==
obstacle))
{
    emit reCalculatePath();
}
}

e_map_cell map::getCellFromIndex(e_cell_pos pos) {
    return map_cell[pos.x_index][pos.y_index];
}

QVector<e_map_cell*> map::getNeighbours(e_cell_pos pos) {

    QVector<e_map_cell*> neighbours;
    if (pos.x_index < num_of_cells-1)
neighbours.append(&map_cell[pos.x_index+1][pos.y_index]);
    if (pos.x_index > 0) neighbours.append(&map_cell[pos.x_index-
1][pos.y_index]);
    if (pos.y_index < num_of_cells-1)
neighbours.append(&map_cell[pos.x_index][pos.y_index+1]);
    if (pos.y_index > 0)
neighbours.append(&map_cell[pos.x_index][pos.y_index-1]);
    if ((pos.x_index > 0) && (pos.y_index > 0))
neighbours.append(&map_cell[pos.x_index-1][pos.y_index-1]);
    if ((pos.x_index < num_of_cells-1) && (pos.y_index > 0))
neighbours.append(&map_cell[pos.x_index+1][pos.y_index-1]);
    if ((pos.x_index > 0) && (pos.y_index < num_of_cells-1))
neighbours.append(&map_cell[pos.x_index-1][pos.y_index+1]);
    if ((pos.x_index < num_of_cells-1) && (pos.y_index < num_of_cells-1))
neighbours.append(&map_cell[pos.x_index+1][pos.y_index+1]);

    return neighbours;
}

bool map::isInclude(QVector<e_map_cell> container, e_map_cell element) {
    for (int j=0; j<container.length(); j++) {
        if ((container[j].pos.x_index ==
element.pos.x_index) && (container[j].pos.y_index == element.pos.y_index)) {
            //qDebug() << "true";
            return true;
        }
    }
    //qDebug() << "false";
    return false;
}

float map::heuristic(e_map_cell to_point, e_map_cell from_point) {
    int dx = abs(to_point.pos.x_index - from_point.pos.x_index);
    int dy = abs(to_point.pos.y_index - from_point.pos.y_index);

    float dist = sqrt((double)(dx*dx+dy*dy));
    return dist;
}

```

```

u_int16_t map::crc16(u_int8_t *packet, u_int32_t nBytes)
{
    u_int16_t crc = 0;
    for(u_int32_t byte=0; byte < nBytes; byte++)
    {
        crc = crc ^ ((u_int16_t)packet[byte]<<8);
        for (u_int8_t bit = 0; bit < 8; bit++)
        {
            if(crc & 0x8000) crc = (crc << 1) ^ 0x1021;
            else crc = crc<<1;
        }
    }
    return crc;
}

void map::aStarSolver()
{
    clearAStar();
    createSafeZone(false);
    openSet.append(getCellFromIndex(start_cell));
    int winner = 0;
    e_map_cell current;
    e_map_cell tmp_cell;
    while (openSet.length()>0){
        for (int i=0; i< openSet.length();i++){
            //qDebug()<

```

```

QVector<e_map_cell*> neighbours = getNeighbours(current.pos);
//qDebug() << neighbours.length();
for (int i=0; i< neighbours.length(); i++) {

    //qDebug() << i;
    if((isInclude(closeSet,*neighbours[i])==false) &&
((neighbours[i]->cell_type != obstacle)&&(neighbours[i]->cell_type != safe_zone_cell)))
    {
        //qDebug() << "OK!";
        float temp_g = current.g +
heuristic(*neighbours[i],current);
        if (isInclude(openSet,*neighbours[i])==true){
            if (temp_g < neighbours[i]->g){
                neighbours[i]->g= temp_g;
                neighbours[i]->h =
heuristic(*neighbours[i],getCellFromIndex(end_cell));
                neighbours[i]->f = neighbours[i]->g +
neighbours[i]->h;
                neighbours[i]->parent = current.pos;
                neighbours[i]->have_parent = true;
            }
        }
        else {
            neighbours[i]->g = temp_g;
            neighbours[i]->h =
heuristic(*neighbours[i],getCellFromIndex(end_cell));
            neighbours[i]->f = neighbours[i]->g + neighbours[i]->h;
            neighbours[i]->parent = current.pos;
            neighbours[i]->have_parent = true;
            openSet.append(*neighbours[i]);
        }
    }
    for (int i = 0; i<closeSet.length();i++) {
        actualizeCell(closeSet[i].pos,close_set);
    }

    for (int i = 0; i<openSet.length();i++) {
        actualizeCell(openSet[i].pos,open_set);
    }
    tmp_cell = current;

}

qDebug() << "No solution";
return;
}

e_path_angle map::getAngle(e_map_cell fr_p, e_map_cell to_p){
    if (fr_p.pos.y_index == to_p.pos.y_index) return angle_0;
    else if ((fr_p.pos.x_index+1 == to_p.pos.x_index)&&(fr_p.pos.y_index+1
== to_p.pos.y_index)) return angle_45;
    else if ((fr_p.pos.x_index-1 == to_p.pos.x_index)&&(fr_p.pos.y_index-1
== to_p.pos.y_index)) return angle_45;
    else if (fr_p.pos.x_index == to_p.pos.x_index) return angle_90;
}

```

```

        else if ((fr_p.pos.x_index-1 == to_p.pos.x_index)&&(fr_p.pos.y_index+1
== to_p.pos.y_index)) return angle_45;
        else if ((fr_p.pos.x_index+1 == to_p.pos.x_index)&&(fr_p.pos.y_index-1
== to_p.pos.y_index)) return angle_45;
        else
        {
            qDebug() << "angle error!";
            return angle_err;
        }
    }

void map::clearAStar(void)
{
    openSet.clear();
    closeSet.clear();
    path.clear();
    bytes_to_send = 0;

    for (int i=0; i<num_of_cells; i++){
        for (int j=0; j<num_of_cells;j++) {
            map_cell[i][j].f = 0;
            map_cell[i][j].g = 0;
            map_cell[i][j].h = 0;
            map_cell[i][j].have_parent = false;
        }
    }
}

void map::createSafeZone(bool expand) {
    //bool mask[3][3] =
    {{false,true,false},{true,true,true},{false,true,false}};
    bool mask[3][3] = {{true,true,true},{true,true,true},{true,true,true}};
    bool **map_obstacle;
    map_obstacle = new bool*[num_of_cells];

    for (int i=0; i < num_of_cells; i++){
        map_obstacle[i]= new bool[num_of_cells];
        for (int j=0; j< num_of_cells; j++) {
            map_obstacle[i][j] = false;
            if (map_cell[i][j].cell_type == obstacle) map_obstacle[i][j] =
true;
            //qDebug() << i << " " << j << " " << map_obstacle[i][j];
        }
    }
    for (int i=1; i< num_of_cells-1; i++){
        for(int j=1; j< num_of_cells-1; j++) {
            int sum=0;
            for(int k=0; k<3; k++){
                for(int l=0; l<3; l++) {
                    if((map_cell[i-1+k][j-
1+l].cell_type==obstacle)&&(mask[l][k]==true)) sum++;
                }
            }
            //qDebug() << sum;
            if (sum>0) map_obstacle[i][j] = true;
            else map_obstacle[i][j] = false;
        }
    }
}

```

```

for (int i = 0; i< num_of_cells; i++) {
    for (int j=0; j<num_of_cells; j++) {
        if (map_obstacle[i][j] == true) {
            e_cell_pos tmp_pos;
            tmp_pos.x_index = i;
            tmp_pos.y_index = j;
            if(expand == true){
                actualizeCell(tmp_pos,obstacle);
            }
            else
                if(map_cell[tmp_pos.x_index][tmp_pos.y_index].cell_type != obstacle)
                    actualizeCell(tmp_pos,safe_zone_cell);
        }
    }
}

delete map_obstacle;
}

void map::preparePathFrame(void) {
    int path_lenght = path.length();
    int num_of_points = 0;

    e_path_angle last_angle, current_angle;
    frame_buf[0]=155;
    frame_buf[1]=7;
    frame_buf[2]=0;

    int32_t current_x = path[path_lenght-1].x_pos*1000;
    int32_t current_y = path[path_lenght-1].y_pos*1000;
    actualizeCell(path[path_lenght-1].pos,path_to_send);

    frame_buf[3] = (u_int8_t)(current_x>>24);
    frame_buf[4] = (u_int8_t)(current_x>>16);
    frame_buf[5] = (u_int8_t)(current_x>>8);
    frame_buf[6] = (u_int8_t)current_x;

    frame_buf[7] = (u_int8_t)(current_y>>24);
    frame_buf[8] = (u_int8_t)(current_y>>16);
    frame_buf[9] = (u_int8_t)(current_y>>8);
    frame_buf[10] = (u_int8_t)current_y;
    num_of_points++;

    for (int i=path_lenght-3; i>=0; i--){
        last_angle=getAngle(path[i+2],path[i+1]);
        current_angle=getAngle(path[i+1],path[i]);

        if (current_angle != last_angle){
            current_x = path[i+1].x_pos*1000;
            current_y = path[i+1].y_pos*1000;

            actualizeCell(path[i+1].pos,path_to_send);

            frame_buf[11 + 8*(num_of_points-1)] =
(u_int8_t)(current_x>>24);
            frame_buf[12 + 8*(num_of_points-1)] =
(u_int8_t)(current_x>>16);
            frame_buf[13 + 8*(num_of_points-1)] = (u_int8_t)(current_x>>8);
            frame_buf[14 + 8*(num_of_points-1)]= (u_int8_t)current_x;
        }
    }
}

```

```

        frame_buf[15 + 8*(num_of_points-1)] =
(u_int8_t)(current_y>>24);
        frame_buf[16 + 8*(num_of_points-1)] =
(u_int8_t)(current_y>>16);
        frame_buf[17 + 8*(num_of_points-1)] = (u_int8_t)(current_y>>8);
frame_buf[18 + 8*(num_of_points-1)] = (u_int8_t)current_y;
last_angle = current_angle;
num_of_points++;
}

if (i==0){
    current_x = path[i].x_pos*1000;
    current_y = path[i].y_pos*1000;

    actualizeCell(path[i].pos,path_to_send);

    frame_buf[11 + 8*(num_of_points-1)] =
(u_int8_t)(current_x>>24);
    frame_buf[12 + 8*(num_of_points-1)] =
(u_int8_t)(current_x>>16);
    frame_buf[13 + 8*(num_of_points-1)] = (u_int8_t)(current_x>>8);
    frame_buf[14 + 8*(num_of_points-1)]= (u_int8_t)current_x;

    frame_buf[15 + 8*(num_of_points-1)] =
(u_int8_t)(current_y>>24);
    frame_buf[16 + 8*(num_of_points-1)] =
(u_int8_t)(current_y>>16);
    frame_buf[17 + 8*(num_of_points-1)] = (u_int8_t)(current_y>>8);
    frame_buf[18 + 8*(num_of_points-1)] = (u_int8_t)current_y;
    num_of_points++;
}
}

frame_buf[2]=num_of_points;
u_int16_t crc=crc16(frame_buf,3+8*num_of_points);
frame_buf[19 + 8*(num_of_points-2)] = (u_int8_t)(crc>>8);
frame_buf[20 + 8*(num_of_points-2)] = (u_int8_t) crc;

bytes_to_send = 5 + 8*(num_of_points);

qDebug() << "Frame prepared! " << num_of_points << "Bytes to send:
" << bytes_to_send;

qDebug() << "CRC: " << (u_int8_t)(crc>>8) << " " << (u_int8_t) crc;

qDebug() << frame_buf[0] << " " << frame_buf[1] << " " << frame_buf[2];
for (int i=3; i< bytes_to_send; i+=8){
    qDebug() << frame_buf[i] << " " << frame_buf[i+1] << " " << frame_buf[i+2] <<
    " << frame_buf[i+3] << " " << frame_buf[i+4] << " " << frame_buf[i+5] <<
    " << frame_buf[i+6] << " " << frame_buf[i+7];
}

void map::actualizeRoverPos(float xpos, float ypos)
{
    rover_pos->setPos(xpos*100,ypos*-100);
}

```


Kielce, dnia 04.09.2020.

SZCZEPAN KOSTECKI 83998

Imię i nazwisko studenta, nr albumu

CZARYŻ 53 29-445 SECEMIN

Adres zamieszkania

AUTOMATYKA I ROBOTYKA, AP, V, STACJONARNE

Kierunek, specjalność, rok studiów, rodzaj studiów (stacjonarne, niestacjonarne)

DR HAB. INZ. PAWEŁ KASKI PROF. PŁK

Opiekun pracy dyplomowej inżynierskiej/magisterskiej*

OŚWIADCZENIE

Przedkładając w roku akademickim 20 19 / 20 opiekunowi pracy dyplomowej inżynierskiej/magisterskiej*, powołanemu przez Dziekana Wydziału MECHATRONIKI I BUDOWY MASZYNNY

Politechniki Świętokrzyskiej, pracę dyplomową inżynierską/magisterską* pod tytułem:

PROJEKT SYSTEMU MAPOWANIA OTOCZENIA ORAZ PLANOWANIA RUCHU ROBOTA MOBILNEGO

oswiadcza, że:

- 1) przedstawiona praca dyplomowa inżynierska/magisterska* została opracowana przeze mnie samodzielnie, stosownie do wskazówek merytorycznych opiekuna pracy,
- 2) przy wykonywaniu pracy dyplomowej inżynierskiej/magisterskiej* wykorzystano materiały źródłowe, w granicach dozwolonego użytku wymieniając autora, tytuł pozycji i miejsce jej publikacji,
- 3) praca dyplomowa inżynierska/magisterska* nie zawiera żadnych danych, informacji i materiałów, których publikacja nie jest prawnie dozwolona,
- 4) przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem stopnia zawodowego/naukowego w wyższej uczelni,
- 5) niniejsza wersja pracy jest identyczna z załączoną treścią elektroniczną (na CD i w systemie Archiwum Prac Dyplomowych).

Przyjmuję do wiadomości, iż w przypadku ujawnienia naruszenia przepisów ustawy o prawie autorskim i prawach pokrewnych, praca dyplomowa inżynierska/magisterska* może być unieważniona przez Uczelnię, nawet po przeprowadzeniu obrony pracy.

Zostałem uprzedzony:

- 1) o odpowiedzialności karnej wynikającej z art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t. j. Dz. U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”,
- 2) odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t. j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwany dalej sądem koleżeńskim”

Prawdziwość powyższego oświadczenia potwierdzam własnoręcznym podpisem.


czytelny podpis studenta

*) niepotrzebne skreślić

SŁCZEPAN KOSTECKI 83998

Kielce, dn....04.09.2020.

Imię i nazwisko studenta nr albumu

CZARYZ 53 28-765 SECEMIN

Adres zamieszkania

AUTOMATYKA I ROBOTYKA, AP, V, STACJONARNE

Kierunek, specjalność, rok studiów, rodzaj studiów (dzienne, zaoczne)

DR HAB. INZ. PAWEŁ ŁASKI PROF. PSK

Opiekun pracy dyplomowej ~~inżynierskiej/magisterskiej*~~

OŚWIADCZENIE

Zgodnie z ustawą z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. nr 24, poz. 83) wyrażam zgodę na udostępnianie mojej pracy dla celów naukowych i dydaktycznych.

Kostecki Sławomir

czytelny podpis studenta



Praca dyplomowa została opracowana z wykorzystaniem aparatury zakupionej z projektu
**„ROZWÓJ BAZY BADAWCZEJ SPECJALISTYCZNYCH LABORATORIÓW UCZELNI PUBLICZNYCH
REGIONU ŚWIĘTOKRZYSKIEGO”**

Priorytet 2: Infrastruktura sfery B+R , Działanie 2.2

Wsparcie tworzenia wspólnej infrastruktury badawczej jednostek naukowych

Programu Operacyjnego Innowacyjna Gospodarka

Projekt nr POIG 02.02.00-26-023/08-00