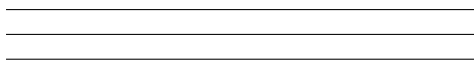


PROJEKT

WIZUALIZACJA DANYCH SENSORYCZNYCH

Mapowanie obszaru za pomocą czujników ultradźwiękowych

Michał Trela, 259312



Prowadzący:
dr inż. Bogdan Kreczmer

Katedra Cybernetyki i Robotyki
Wydziału Elektroniki, Fotoniki i
Mikrosystemów
Politechniki Wrocławskiej

31 maja 2024

Spis treści

| | | |
|----------|---|-----------|
| 1 | Charakterystyka tematu projektu | 1 |
| 2 | Etapy realizacji projektu, kamienie milowe | 1 |
| 3 | Specyfikacja finalnego produktu | 1 |
| 4 | Harmonogram realizacji podcelów | 2 |
| 4.1 | Wykres Gantta | 3 |
| 5 | Projekt interfejsu | 4 |
| 5.1 | Okno powitalne, okno połączenia | 4 |
| 5.2 | Okno wizualizacji mapowania | 4 |
| 6 | Wstępne rezultaty | 7 |
| 6.1 | Układ elektroniczny | 7 |
| 6.1.1 | Kod programu | 7 |
| 6.1.2 | Format ramki | 9 |
| 6.2 | Aplikacja | 10 |
| 6.2.1 | Aktualny wygląd aplikacji | 10 |
| 6.3 | Dokumentacja | 11 |
| 6.3.1 | Przykładowa strona z dokumentacji Doxygen | 11 |
| 7 | Rezultaty zaawansowane | 12 |
| 7.1 | Aplikacja | 12 |
| 7.1.1 | Wygląd aplikacji | 12 |
| 7.2 | Dokumentacja | 15 |
| 7.3 | Kod źródłowy | 16 |
| 8 | Rezultaty prawie końcowe | 17 |
| 8.1 | Aplikacja | 17 |
| 8.1.1 | Wygląd aplikacji | 17 |
| 8.2 | Dokumentacja | 20 |
| 8.3 | Kod źródłowy | 20 |
| 9 | Rezultaty końcowe | 21 |
| 9.1 | Zmiany w aplikacji | 21 |
| 9.1.1 | Wygląd aplikacji | 21 |
| 9.2 | Efekt końcowy | 24 |

1 Charakterystyka tematu projektu

Projekt zakłada mapowanie obszaru bazując na danych sensorycznych otrzymanych z czujników ultradźwiękowych robota Pioneer 3-DX z uwzględnieniem pozycji robota w przestrzeni. Obsługa czujników oraz przesyłanie danych będzie odbywać się w oparciu o framework ROS2, natomiast wizualizacja zostanie zrealizowana za pomocą biblioteki Qt. W związku z ograniczoną dostępnością robota Pioneer, prototyp projektu zostanie wykonany bazując na Arduino UNO, czujnikach ultradźwiękowych typu HC-SR04 oraz bibliotekach micro-ROS.

2 Etapy realizacji projektu, kamienie milowe

Etap I, Prototyp:

- Przegląd literatury i zasobów internetowych
- Schemat komunikacji pomiędzy Arduino (micro-ROS) a jednostką główną (ROS2).
- Obsługa czujników i przesył danych do jednostki głównej.
- Wizualizacja danych z prototypu.
- Implementacja mapowania obszaru z uwzględnieniem pozycji w przestrzeni.

Etap II, Robot Pioneer:

- Schemat komunikacji między robotem Pioneer (ROS2) a komputerem głównym (ROS2).
- Obsługa czujników i przesył danych do jednostki głównej.
- Implementacja mapowania obszaru z uwzględnieniem pozycji w przestrzeni.

3 Specyfikacja finalnego produktu

Efekt końcowym aplikacji wizualizująca mapę obszaru za pomocą robota i jego czujników ultradźwiękowych z różnych punktów w przestrzeni.

4 Harmonogram realizacji podcelów

- 26.03.2023 - Zakończenie przeglądu literatury i zasobów internetowych
- 30.03.2023 - Projekt interfejsu graficznego

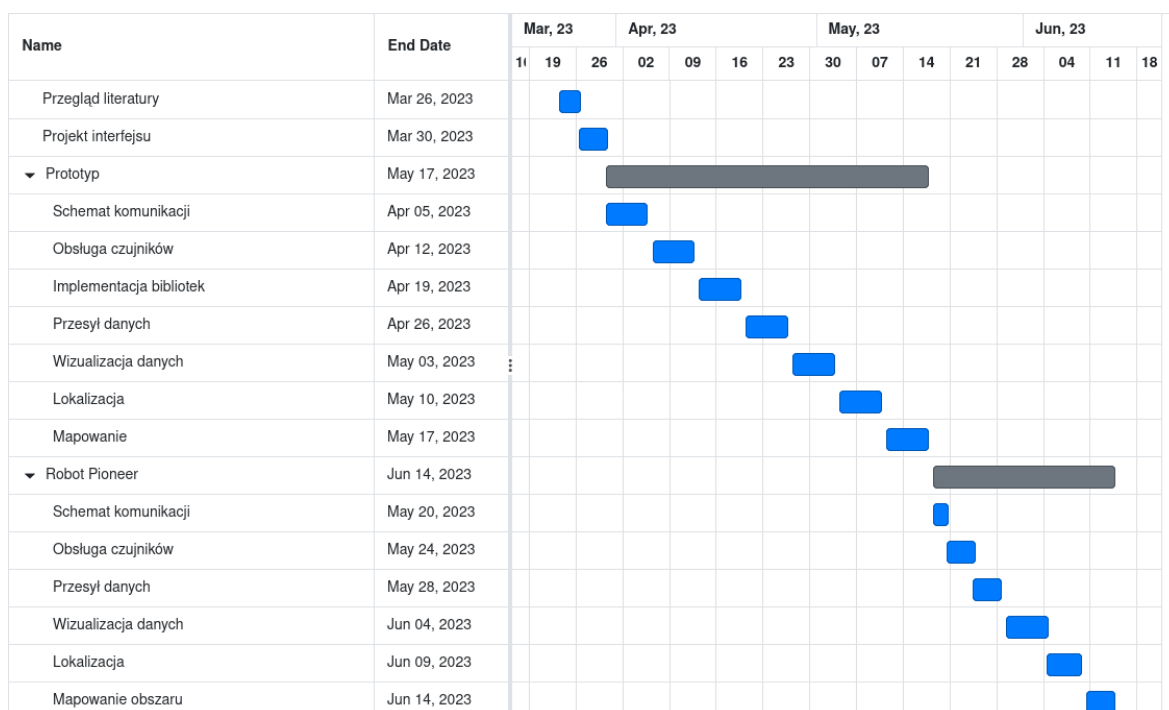
Etap I - Prototyp

- 05.04.2023 - Schemat komunikacji pomiędzy urządzeniami
- 12.04.2023 - Obsługa czujników
- 19.04.2023 - Implementacja bibliotek micro-ROS
- 26.04.2023 - Przesył danych sensorycznych do jednostki głównej
- 03.05.2023 - Wizualizacja danych sensorycznych w jednostce głównej
- 10.05.2023 - Lokalizacji prototypu w przestrzeni
- 17.05.2023 - Mapowanie obszaru

Etap II - Robot Pioneer

- 20.05.2023 - Schemat komunikacji pomiędzy urządzeniami
- 24.05.2023 - Obsługa czujników
- 28.05.2023 - Przesył danych do jednostki głównej
- 04.06.2023 - Wizualizacja danych sensorycznych w jednostce głównej
- 09.06.2023 - Lokalizacja robota w przestrzeni
- 14.06.2023 - Mapowanie obszaru

4.1 Wykres Gantta

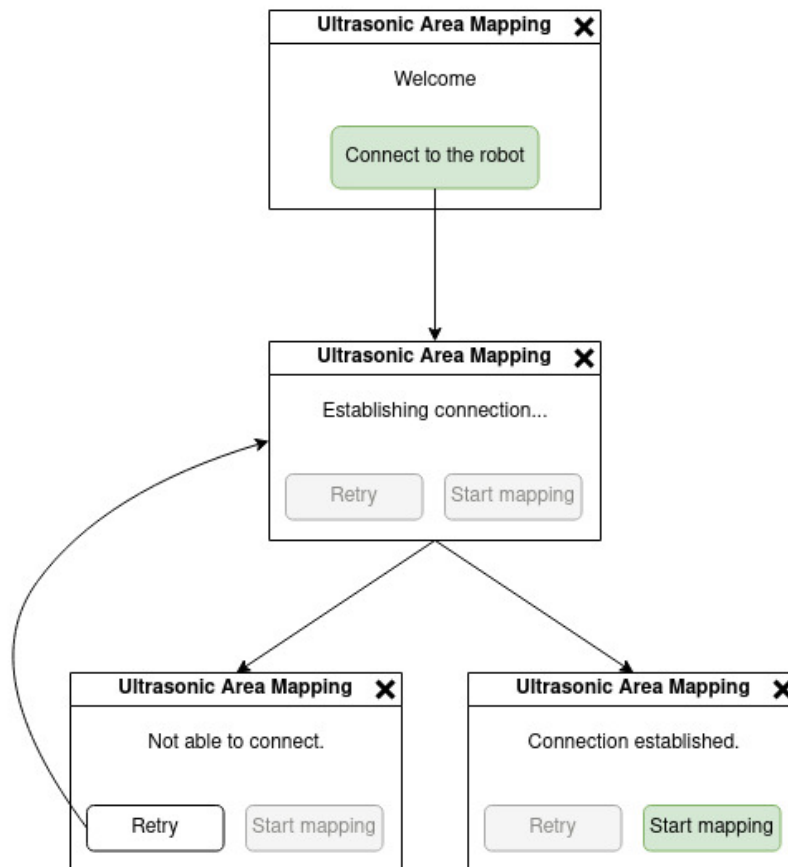


Rysunek 1: Wykres Gantta uzyskany za pomocą serwisu Online Gantt

5 Projekt interfejsu

5.1 Okno powitalne, okno połączenia

Pierwszym oknem wyświetlającym się po uruchomieniu aplikacji jest okno powitalne. Po kliknięciu przycisku 'Connect to the robot' przechodzimy do okna łączenia z robotem. Podczas ustanawiania połączenia przyciski 'Retry' oraz 'Start mapping' są nieaktywne. Jeśli połączenie nie zostanie ustanowione to przycisk 'Retry' staje się aktywny. Po jego przyciśnięciu ustanawianie połączenia jest ponawiane. Jeśli połączenie zostanie ustanowione to przycisk 'Start mapping' staje się aktywny. Po jego przyciśnięciu przechodzimy do okna mapowania obszaru.



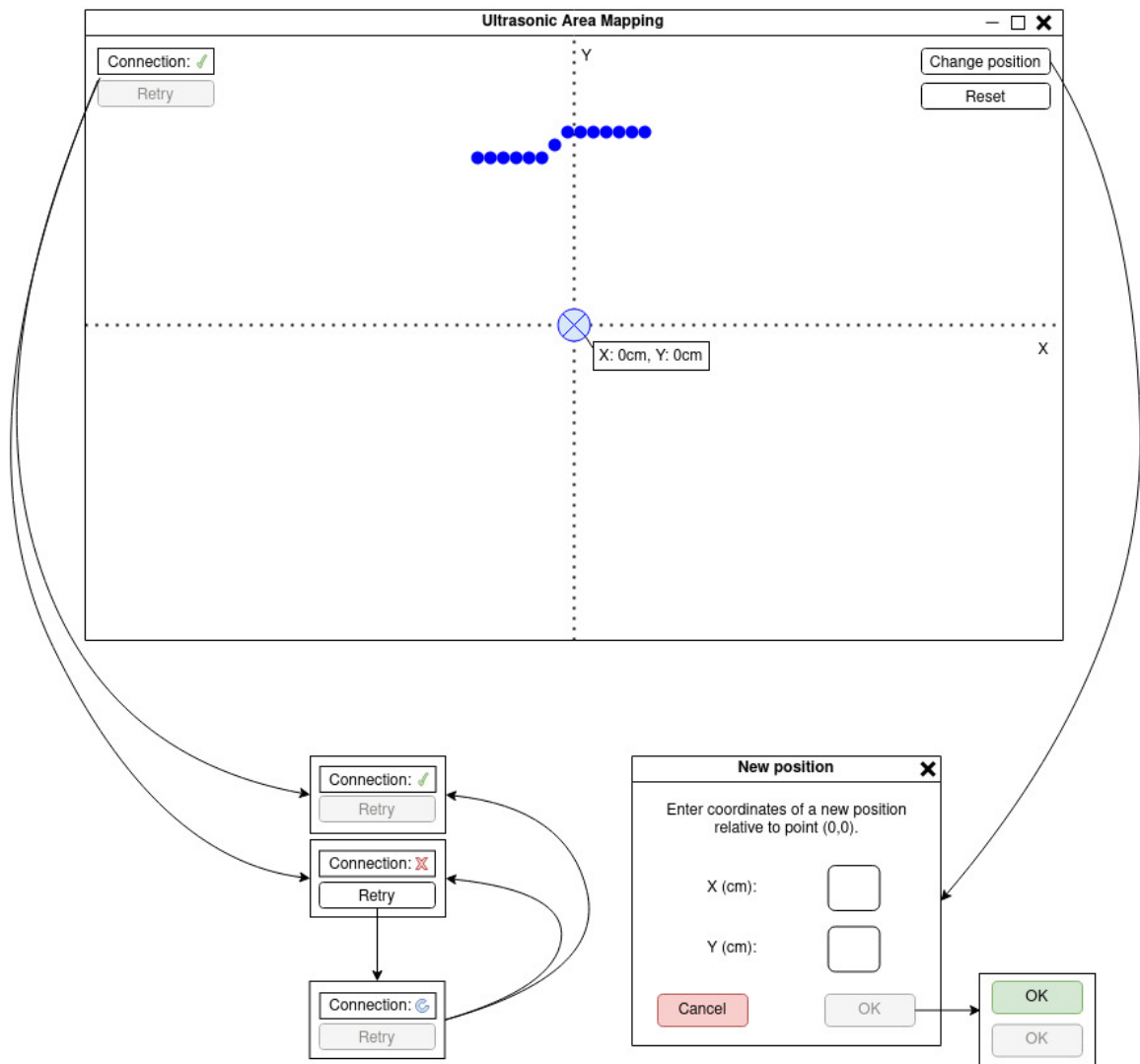
Rysunek 2: Okno powitalne oraz okna połączenia komputera z robotem

5.2 Okno wizualizacji mapowania

W oknie wizualizacji wyświetlane są:

- Aktualna pozycja robota wraz ze współrzędnymi położenia (kolor niebieski)
- Punkty otrzymane z czujników w aktualnej pozycji (kolor niebieski)
- Poprzednie pozycje robota (kolor szary)
- Punkty otrzymane z czujników w poprzednich pozycjach (kolor szary)

- Osie wraz z oznaczeniami
- Stan połączenia
- Przyciski umożliwiające zmianę pozycji robota, przycisk reset oraz przycisk ponownej próby połączenia



Rysunek 3: Okno wizualizacji mapowania oraz okno zmiany pozycji robota

Po uruchomieniu aplikacji, zawsze pierwsza pozycja jest określana jako pozycja początkowa czyli (0,0). Współrzędne pozycji są określane w centymetrach.

Stan połączenia, ponowne łączenie:

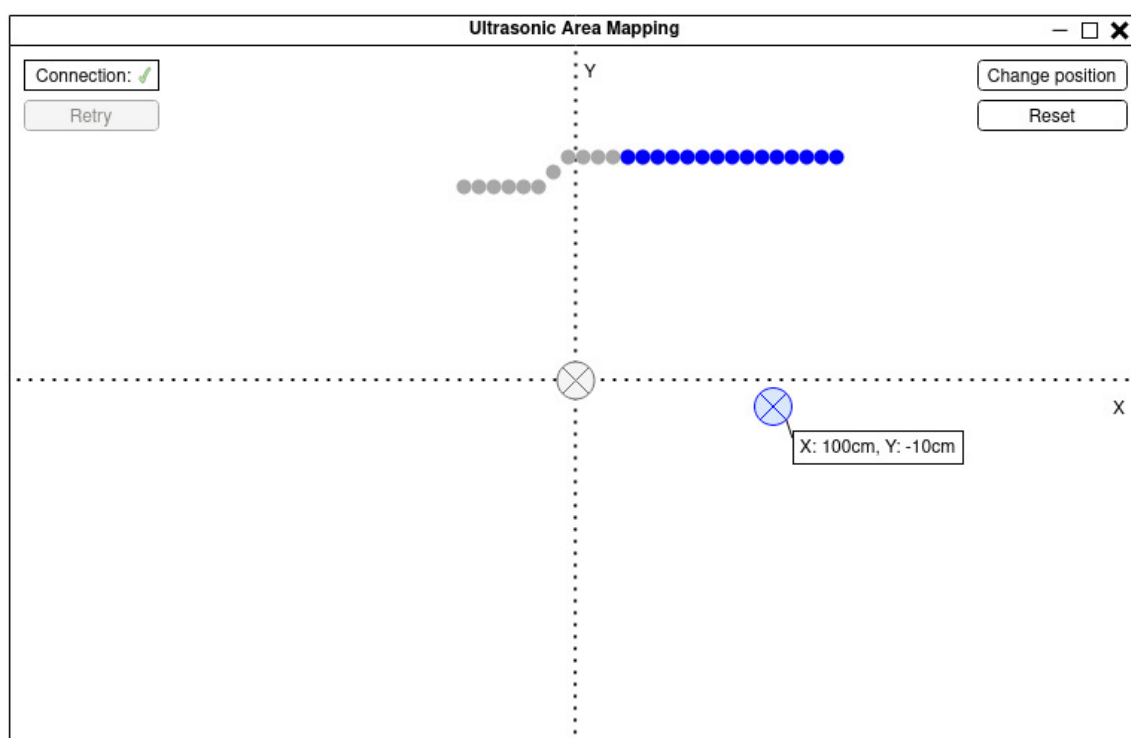
Przy ustanowionym połączeniu **Connection: ✓**, przycisk **'Retry'** jest nieaktywny. Jeśli połączenie zostanie utracone **Connection: ✗**, przycisk **'Retry'** staje się aktywny a po jego przyciśnięciu ponawiane jest ustanawianie połączenia z robotem **Connection: 🔄**.

Zmiana pozycji:

Po kliknięciu przycisku 'Change position' wyświetlane jest okno dodawania nowej pozycji. Aby dodać nową pozycję należy poprawnie wpisać współrzędne (tylko liczby całkowite) względem punktu (0,0) po czym domyślnie nieaktywny przycisk OK staje się aktywny i pozycja robota zostaje zmieniona.

Reset:

Po przyciśnięciu przycisku 'Reset' wszystkie punkty otrzymane z czujników są usuwane a aktualna pozycja robota zostaje ustawiona jako pozycja (0,0).



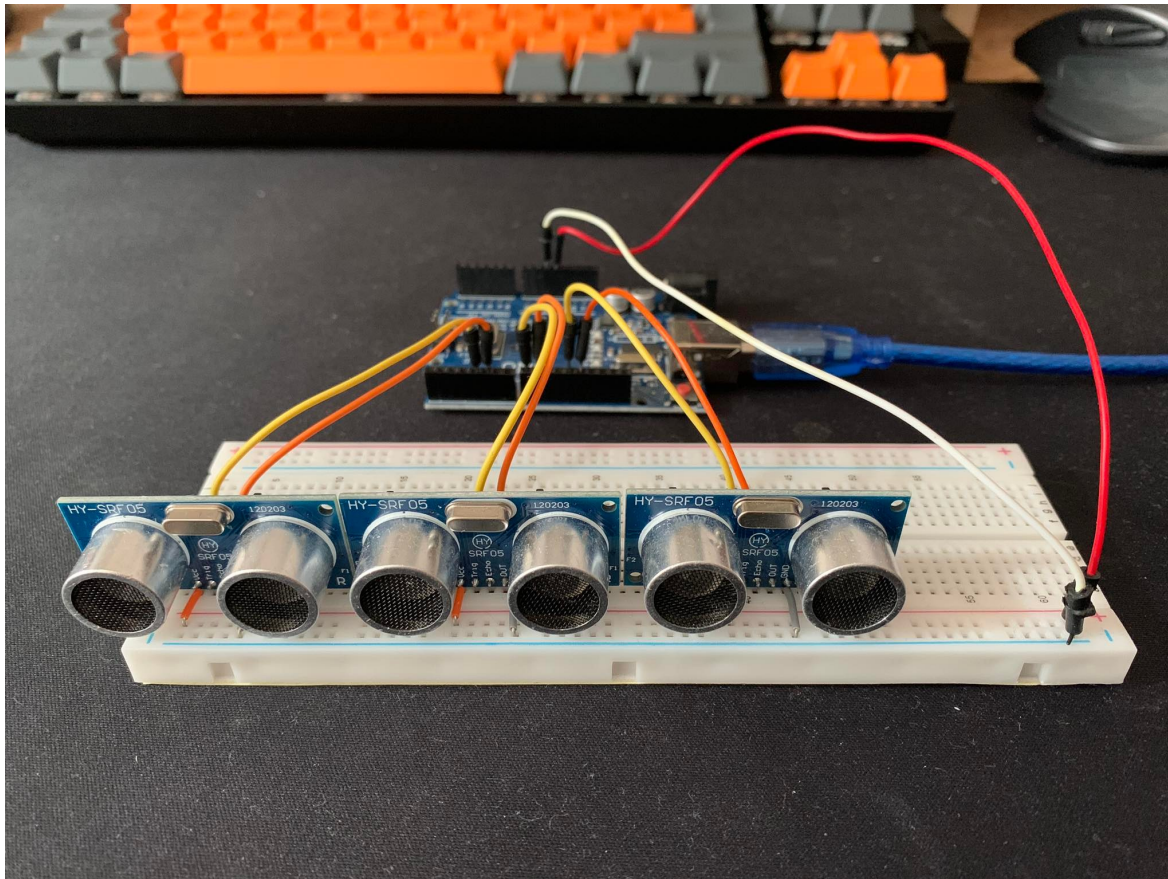
Rysunek 4: Okno wizualizacji mapowania oraz okno zmiany pozycji robota

Projekt interfejsu został wykonany za pomocą serwisu Draw.IO

6 Wstępne rezultaty

6.1 Układ elektroniczny

Złożony został układ elektroniczny składający się z trzech czujników ultradźwiękowych typu HY-SRF05 i Arduino Uno.



Rysunek 5: Układ elektroniczny

6.1.1 Kod programu

Napisany został program umożliwiający obsługę czujników, odczyt długości impulsu wejściowego z czujnika a na jego podstawie obliczenie odległości czujnika od przeszkody. Po wyznaczeniu odległości przeszkody od każdego z czujników, obliczana jest suma kontrolna wysyłanej paczki danych a następnie poprawnie sformatowana ramka jest wysyłana przez port seryjny.

```
1 #include "CRC16.h"
2 #include "CRC.h"
3
4 class Sensor {
5     public:
6         uint8_t SENSOR_ID;
7         uint8_t TRIG_PIN;
8         uint8_t ECHO_PIN;
9         Sensor(uint8_t SENSOR_ID, uint8_t TRIG_PIN, uint8_t ECHO_PIN);
10 };
```

```

11
12 Sensor::Sensor(uint8_t SENSOR_ID, uint8_t TRIG_PIN, uint8_t ECHO_PIN) {
13     Sensor::SENSOR_ID = SENSOR_ID;
14     Sensor::TRIG_PIN = TRIG_PIN;
15     Sensor::ECHO_PIN = ECHO_PIN;
16 }
17
18 Sensor list[] = {Sensor(0, 4, 5), Sensor(1, 8, 9),
19                 Sensor(2, 12, 13)};
20 CRC16 crc;
21
22 long readFromSensor(int TRIG_PIN, int ECHO_PIN) {
23     digitalWrite(TRIG_PIN, LOW);
24     delayMicroseconds(2);
25     digitalWrite(TRIG_PIN, HIGH);
26     delayMicroseconds(10);
27     digitalWrite(TRIG_PIN, LOW);
28     return pulseIn(ECHO_PIN, HIGH);
29 }
30
31 void setup() {
32     for (Sensor sensor : list) {
33         pinMode(sensor.TRIG_PIN, OUTPUT);
34         pinMode(sensor.ECHO_PIN, INPUT);
35     }
36     Serial.begin(9600);
37     crc.setPolynome(0x1021);
38 }
39
40 void loop() {
41     long duration, distanceCm;
42     char numstr[21];
43     String data = "";
44
45     for (Sensor sensor : list) {
46         duration = readFromSensor(sensor.TRIG_PIN, sensor.ECHO_PIN);
47         distanceCm = duration / 29.1 / 2;
48         if (distanceCm <= 0) {
49             Serial.println("Out of range");
50         } else {
51             sprintf(numstr, "%d %lu ", sensor.SENSOR_ID, distanceCm);
52             data = data + numstr;
53         }
54         delay(100);
55     }
56     for (char i: data) {
57         crc.add(i);
58     }
59     Serial.print(data);
60     Serial.print("\t");
61     Serial.println(crc.getCRC(), HEX);
62     delay(1000);
63     crc.reset();
64 }

```

6.1.2 Format ramki

Przesyłana ramka z danymi ma następujący format:

ID_0 ODL_0 ID_1 ODL_1 ID_2 ODL_2 SUMA_KONTROLNA

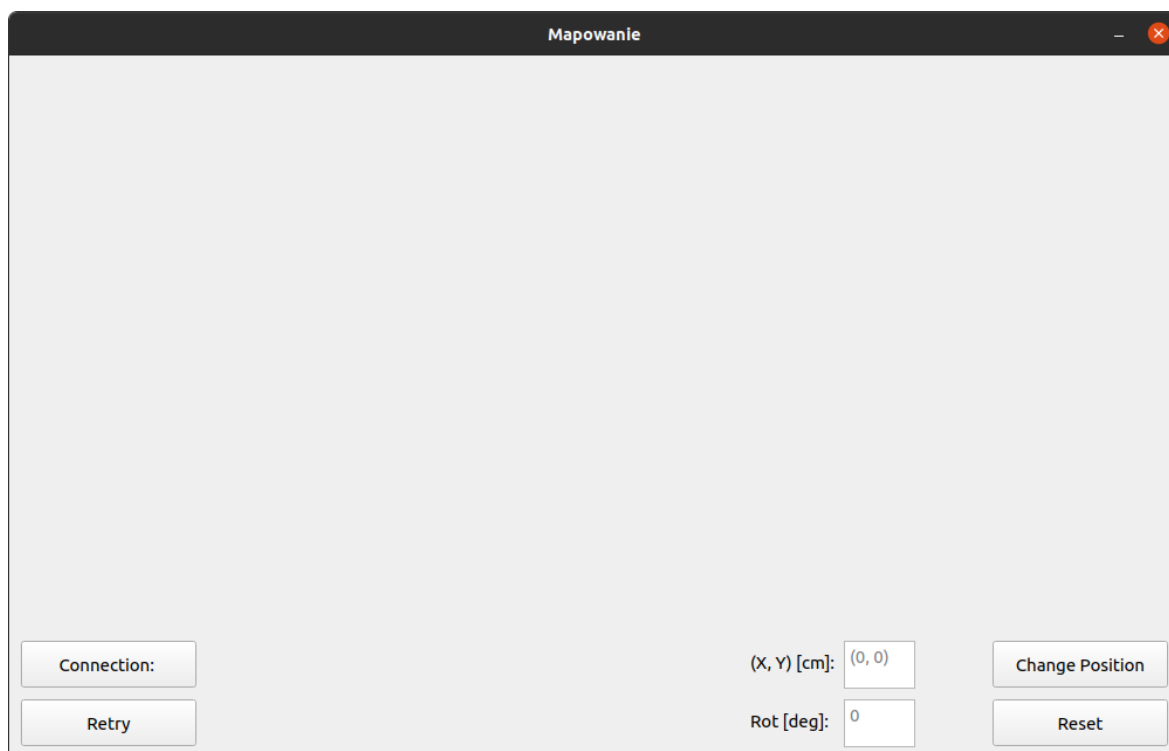
Gdzie:

- ID_* - ID czujnika
- ODL_* - Odległość przeszkody od czujnika w centymetrach
- SUMA_KONTROLNA - Suma kontrolna w systemie CRC16 wyliczona z przesyłanych danych

Koniec ramki oznaczany jest znakiem nowej linii

6.2 Aplikacja

6.2.1 Aktualny wygląd aplikacji



Rysunek 6: Okno główne



Rysunek 7: Okno zmiany pozycji

6.3 Dokumentacja

6.3.1 Przykładowa strona z dokumentacji Doxygen

The screenshot shows a Doxygen-generated documentation page for a class named **Sensor**. The page title is "Mapowanie obszaru" (Area Mapping). Below the title is a subtitle: "Wizualizacja mapowania obszaru za pomocą czujników ultradźwiękowych" (Visualization of area mapping using ultrasonic sensors). The page has a navigation bar with links: "Strona główna", "Klasy", and "Pliki". A search bar is also present. The main content area is titled "Dokumentacja klasy Sensor" and includes sections for "Metody publiczne" (Public Methods), "Atrybuty publiczne" (Public Attributes), "Opis szczegółowy" (Detailed Description), and "Dokumentacja konstruktora i destruktor" (Constructor and Destructor Documentation). The "Metody publiczne" section shows a constructor: **Sensor** (uint8_t **SENSOR_ID**, uint8_t **TRIG_PIN**, uint8_t **ECHO_PIN**) with a description "Konstruktor. Więcej...". The "Atrybuty publiczne" section lists three attributes: uint8_t **SENSOR_ID**, uint8_t **TRIG_PIN**, and uint8_t **ECHO_PIN**. The "Opis szczegółowy" section contains a paragraph: "Klasa czujnik posiada pola przechowujące informacje o ID czujnika oraz pinów mikrokontrolera używanych przez czujnika". The "Dokumentacja konstruktora i destruktor" section shows the constructor signature: **Sensor()** and its definition: **Sensor::Sensor** (uint8_t **SENSOR_ID**, ...).

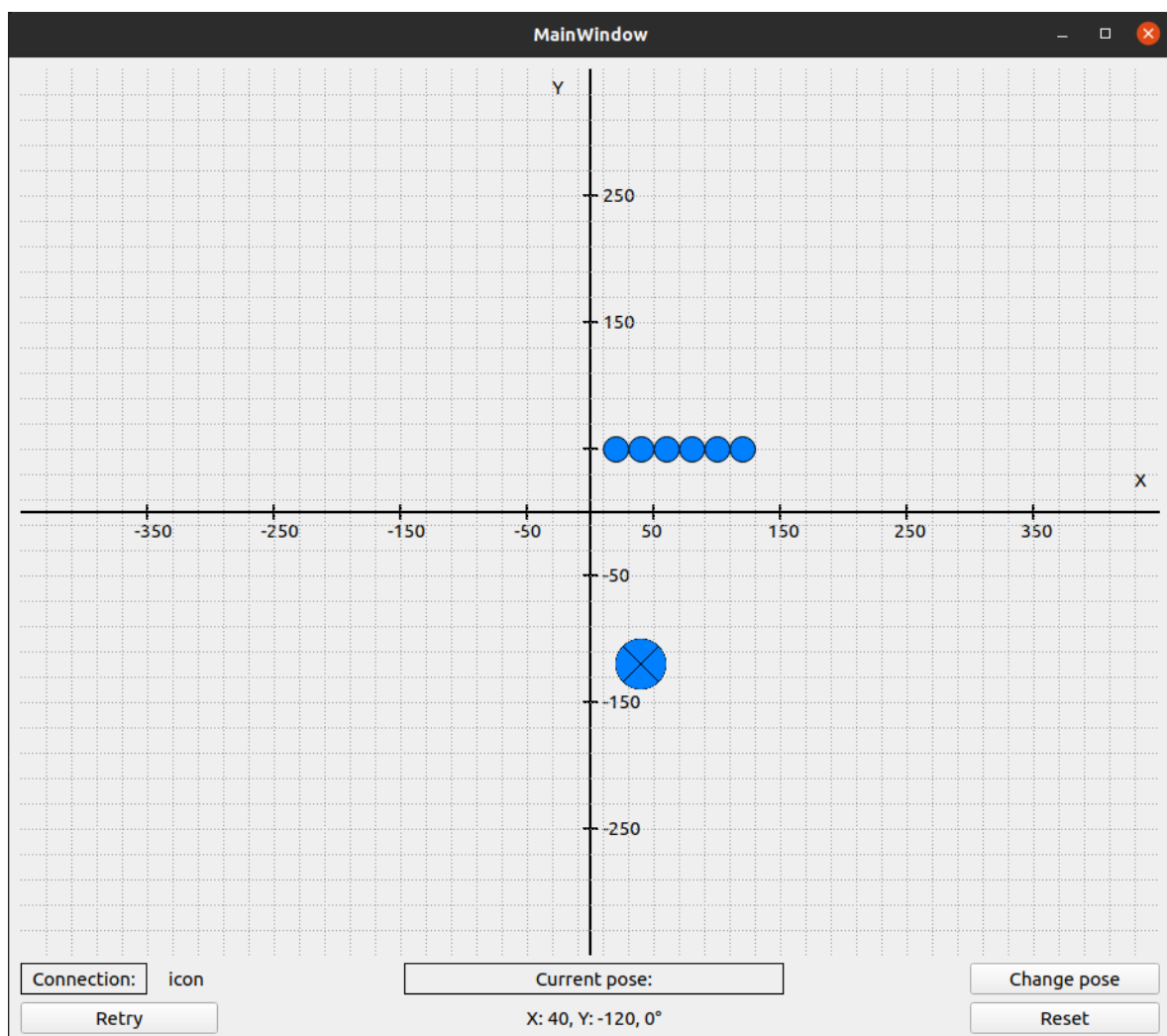
Rysunek 8: Przykład dokumentacji

7 Rezultaty zaawansowane

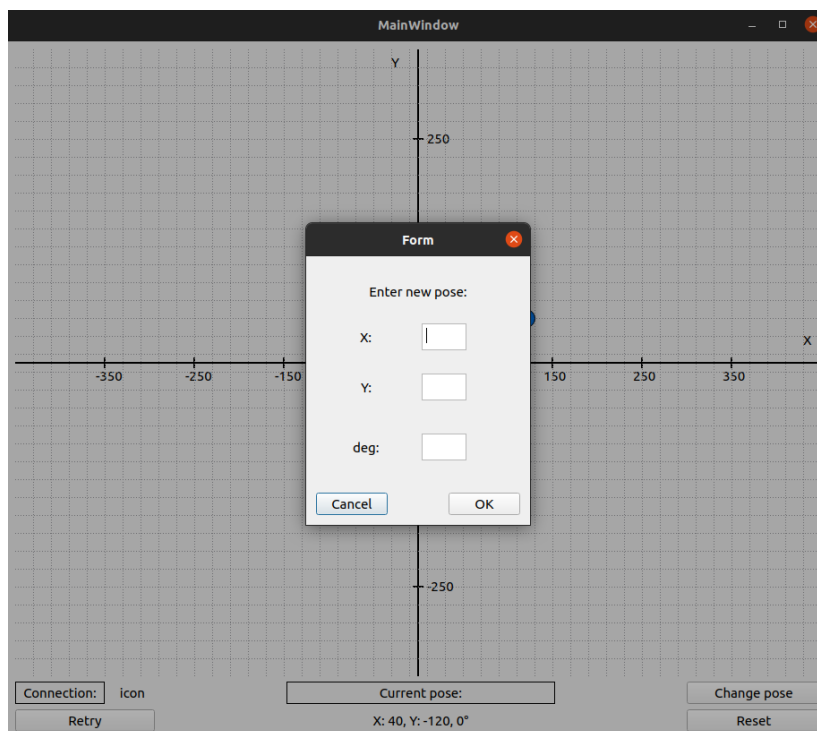
7.1 Aplikacja

Aplikacja została przebudowana od nowa. Dodane zostały klasy punktów, urządzenia, klasa widgetu mapy, w którym rysowany jest układ współrzędnych oraz punkty i pozycje urządzenia. Zrobiony został nowy UI dla okna głównego 9 oraz okna dialogowego zmiany pozycji 10. Wyświetlana jest aktualna pozycja urządzenia, podpięte są przyciski resetujące mapę oraz przycisk wyświetlający okno dialogowe 10. Dodana została funkcjonalność zmiany pozycji 11. Po zmianie pozycji, poprzednie pozycje są wyświetlane na szaro 12. Reset mapy polega na usunięciu wszystkich punktów i ustawieniu aktualnej pozycji jako domyślnej 13.

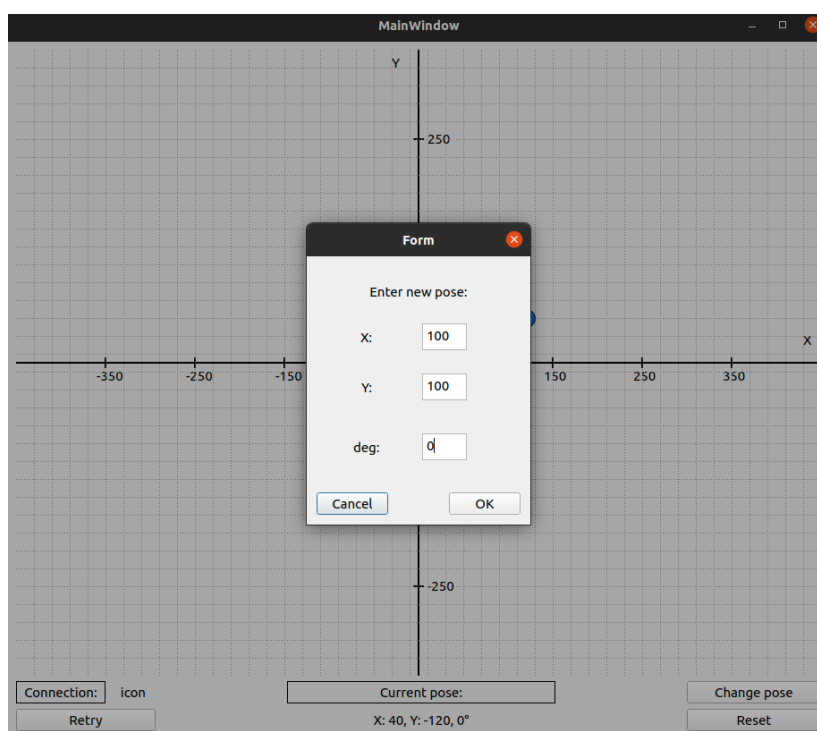
7.1.1 Wygląd aplikacji



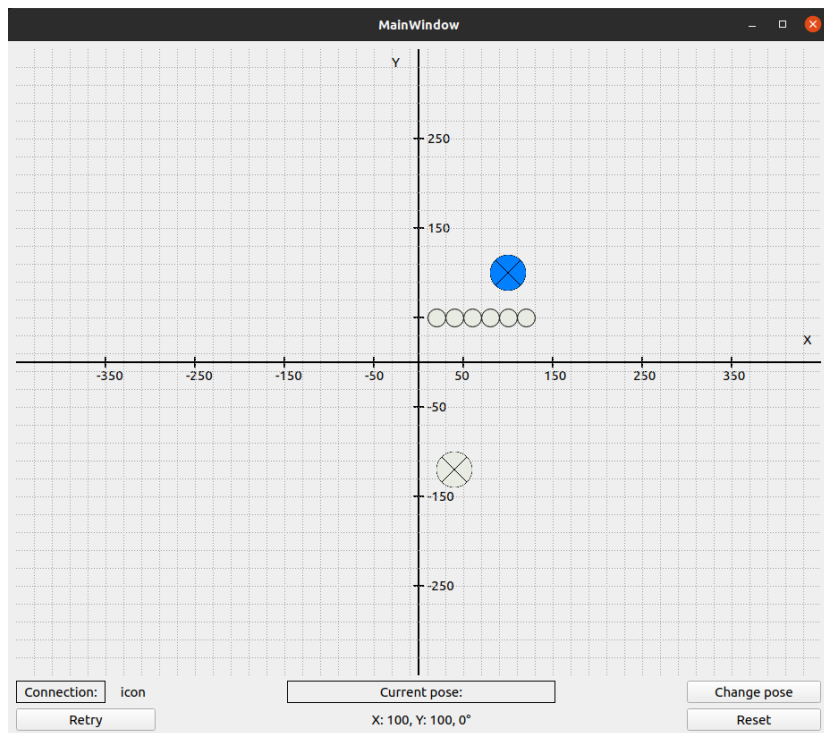
Rysunek 9: Okno wizualizacji mapowania



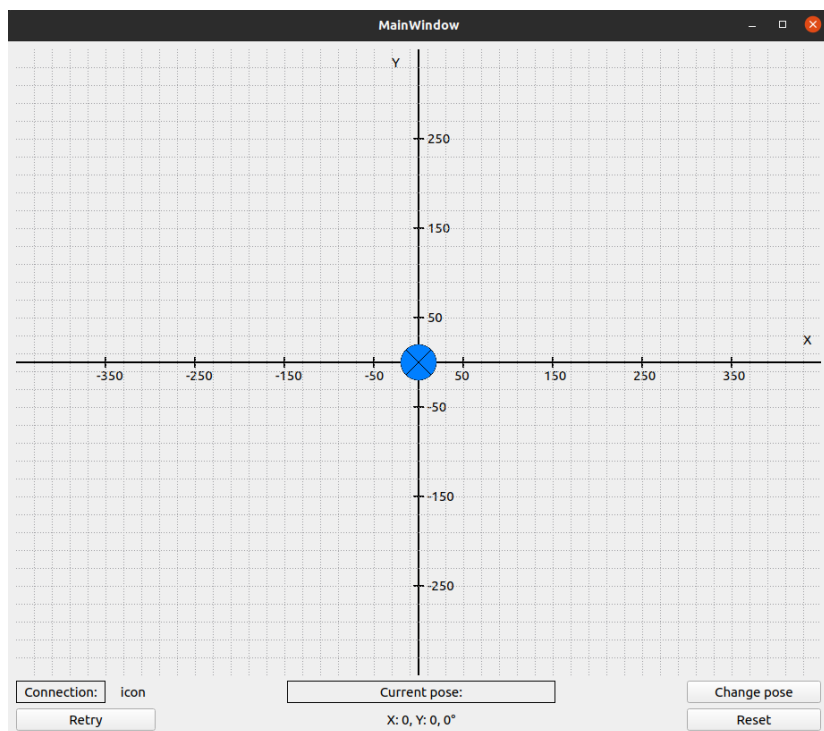
Rysunek 10: Okno dialogowe zmiany pozycji



Rysunek 11: Okno dialogowe zmiany pozycji



Rysunek 12: Okno dialogowe zmiany pozycji



Rysunek 13: Okno dialogowe zmiany pozycji

7.2 Dokumentacja

Dodała zostana pełna dokumentacja. 14 15 16 17

Mapowanie obszaru

Wizualizacja mapowania obszaru za pomocą czujników ultradźwiękowych

[Strona główna](#) | [Klasy](#) | [Pliki](#) |

Metody publiczne | Atrybuty publiczne | Lista wszystkich składowych

Dokumentacja klasy Sensor

Metody publiczne

Sensor (uint8_t SENSOR_ID, uint8_t TRIG_PIN, uint8_t ECHO_PIN)
Konstruktor. Więcej...

Atrybuty publiczne

uint8_t SENSOR_ID
uint8_t TRIG_PIN
uint8_t ECHO_PIN

Opis szczegółowy

Klasa czujnik posiada pola przechowujące informacje o ID czujnika oraz pinów mikrokontrolera używanych przez czujnika

Dokumentacja konstruktora i destruktor

◆ Sensor()

Sensor::Sensor (uint8_t SENSOR_ID,

Rysunek 14: Przykładowe zdjęcie dokumentacji

Dokumentacja funkcji składowych

◆ drawGrid()

void Map::drawGrid (QPainter & painter,
int mapWidth,
int mapHeight
)

private

Metoda rysująca układ współrzędnych.

Za pomocą tej metody rysowany jest układ współrzędnych

Parametry

painter

referencja do obiektu typu QPainter umożliwiającego rysowanie

mapWidth

szerokość widgetu w którym rysowana jest mapa

mapHeight

wysokość widgetu w którym rysowana jest mapa

Definicja w linii 35 pliku map.cpp.

```
35 {  
36     // Translate the coordinate system so that point 0 is in the center of the widget  
37     painter.translate(mapWidth / 2, mapHeight / 2);  
38  
39     // Set the pen color and width  
40     painter.setPen(QPen(Qt::black, 2));  
41  
42     // Draw axis  
43     painter.drawLine(-mapWidth / 2, 0, mapWidth / 2, 0);  
44     painter.drawLine(0, -mapHeight / 2, 0, mapHeight / 2);  
45  
46     // Draw ticks and labels  
47     int tickDistance = 100;  
48     for (int x = -mapWidth / 2 + tickDistance; x < mapWidth / 2; x += tickDistance)  
49     {  
50         painter.drawLine(x, -5, x, 5);  
51         painter.drawText(x - 10, 20, QString::number(x));  
52     }  
53     for (int y = -mapHeight / 2 + tickDistance; y < mapHeight / 2; y += tickDistance)  
54     {  
55         painter.drawLine(-5, y, 5, y);  
56         painter.drawText(10, y + 5, QString::number(-y));  
57     }  
58  
59     // Draw labels  
60     painter.drawText(mapWidth / 2 - 20, - 20, "X");  
61     painter.drawText(- 30 , - mapHeight / 2 + 20, "Y");  
62  
63     // Draw grid
```

Rysunek 15: Przykładowe zdjęcie dokumentacji

Opis szczegółowy

Klasa widgetu mapy.

W tej klasie bazując na widgecie rysowany jest układ współrzędnych. Przechowywane są też informacje o zmapowanych punktach, aktualnej pozycji urządzenia oraz o poprzednich pozycjach urządzenia

Definicja w linii 27 pliku `map.h`.

Dokumentacja konstruktora i destruktor

◆ Map()

Map::Map (QWidget * parent = nullptr)

Konstruktor klasy.

W konstruktorze definiowane są wszystkie obiekty typu QPixmap

Definicja w linii 3 pliku `map.cpp`.

```
3      : QWidget(parent) {
4
5      dev_pos_pixmap = new QPixmap(":/device_pos.svg");
6      *dev_pos_pixmap = dev_pos_pixmap->scaled(dev_pos_pixmap->width() / 2, dev_pos_pixmap->height() / 2);
7      curr_dev_pose = new DevicePoint(*dev_pos_pixmap, {40, -120, 0});
8
9      point_pixmap = new QPixmap(":/point.svg");
10     for (int i = 0; i < 6; i++) {
11         curr_points.push_back(DevicePoint(*point_pixmap, {(i + 1) * 20, 50}));
12     }
13
14     point_prev_pixmap = new QPixmap(":/point_prev.svg");
15     dev_pos_prev_pixmap = new QPixmap(":/device_pos_prev.svg");
16
17     *dev_pos_prev_pixmap = dev_pos_prev_pixmap->scaled(dev_pos_prev_pixmap->width() / 2,
18     dev_pos_prev_pixmap->height() / 2);
19 }
```

Rysunek 16: Przykładowe zdjęcie dokumentacji

Dokumentacja klasy DevicePoint

Metody publiczne | Atrybuty prywatne | Lista wszystkich składowych

Klasa urządzenia jako punktu na mapie. Więcej...

#include <devicepoint.h>

Metody publiczne

| | |
|----------------------|---|
| | DevicePoint (const QPixmap &pixmap, std::array< int, 3 > pose) |
| | Konstruktor klasy. Więcej... |
| std::array< int, 3 > | getPose () |
| const QPixmap & | getPixmap () |
| | Metoda pobierająca pixmap. Więcej... |
| void | setPose (std::array< int, 3 > newPose) |
| | Metoda ustawiająca tablice współrzędnych. Więcej... |
| void | setPixmap (const QPixmap &newPixmap) |
| | Metoda ustawiająca pixmap. Więcej... |

Atrybuty prywatne

| | |
|----------------------|-----------------------------|
| QPixmap | pixmap |
| | Prywatna zmienna. Więcej... |
| std::array< int, 3 > | pose |
| | Prywatna zmienna. Więcej... |

Opis szczegółowy

Klasa urządzenia jako punktu na mapie.

Klasa zawierająca informacje o współrzędnych, w których urządzenie się znajduje, jego orientacji oraz pixmap determinujący czy pozycja jest aktualna.

Definicja w linii 15 pliku `devicepoint.h`.

Rysunek 17: Przykładowe zdjęcie dokumentacji

7.3 Kod źródłowy

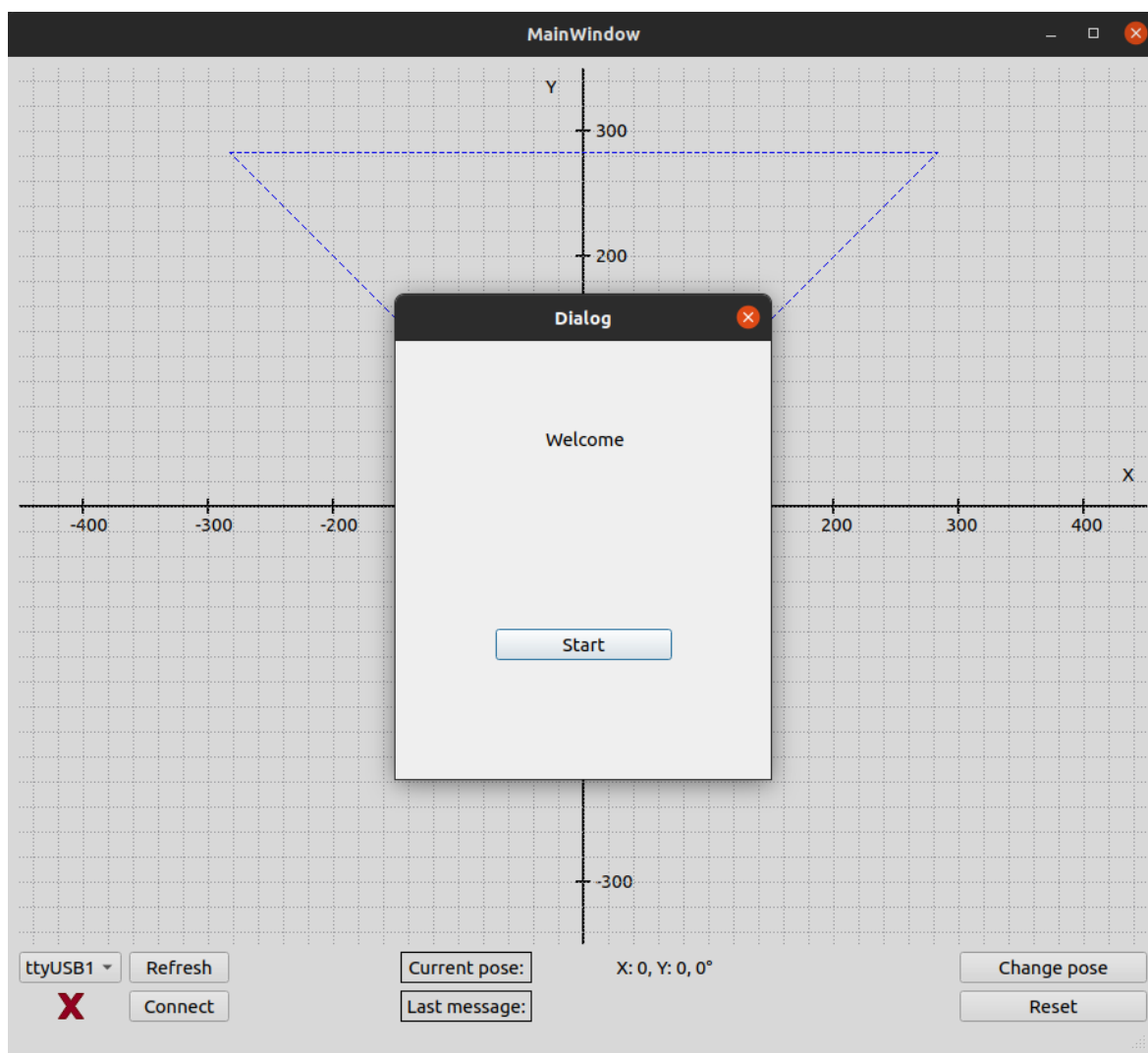
Kod źródłowy jest udostępniony w serwisie GitHub.

8 Rezultaty prawie końcowe

8.1 Aplikacja

Stworzono okno powitalne wyświetlające się przy otwarciu aplikacji, po którego zamknięciu przechodzimy do okna głównego 18. Dodana została wizualna oraz tekstowa sygnalizacja połączenia z urządzeniem 19 20 oraz rysowanie obszaru obejmowanego przez czujniki 22 23. Połączenie z urządzeniem odbywa się przez port seryjny, na który zostaje wysłana wiadomość. Jeśli urządzenie jest podłączone poprawnie to ta wiadomość zostanie odebrana i wysłana zostanie informacja zwrotna potwierdzająca połączenie. Pola edycji tekstu służące do zmiany pozycji zostały zmodyfikowane aby jako dane wejściowe akceptowane były tylko liczby, łącznik jako znak minus oraz klawisze `Backspace` i `Delete` 21.

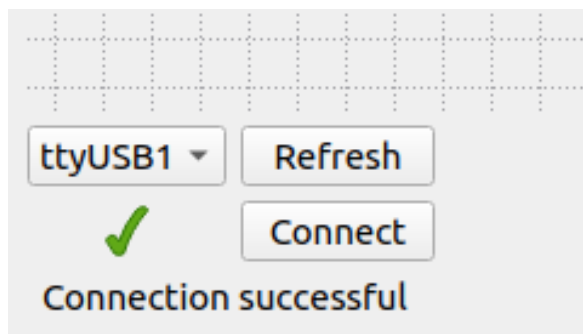
8.1.1 Wygląd aplikacji



Rysunek 18: Okno powitalne



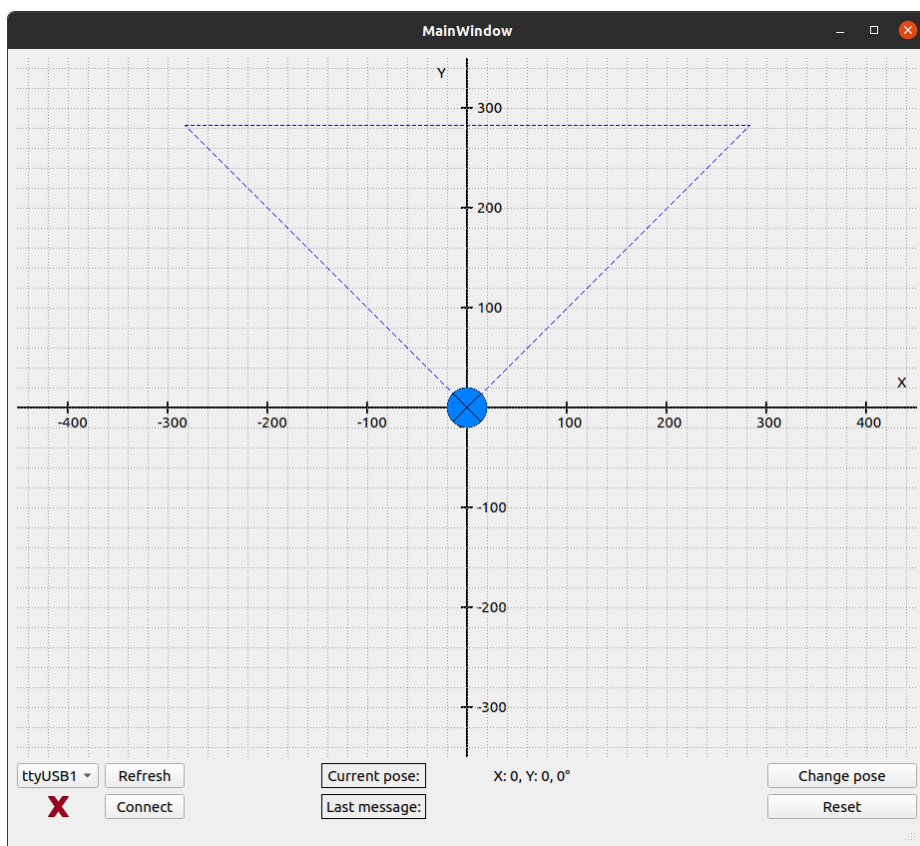
Rysunek 19: Sygnalizacja połączenia



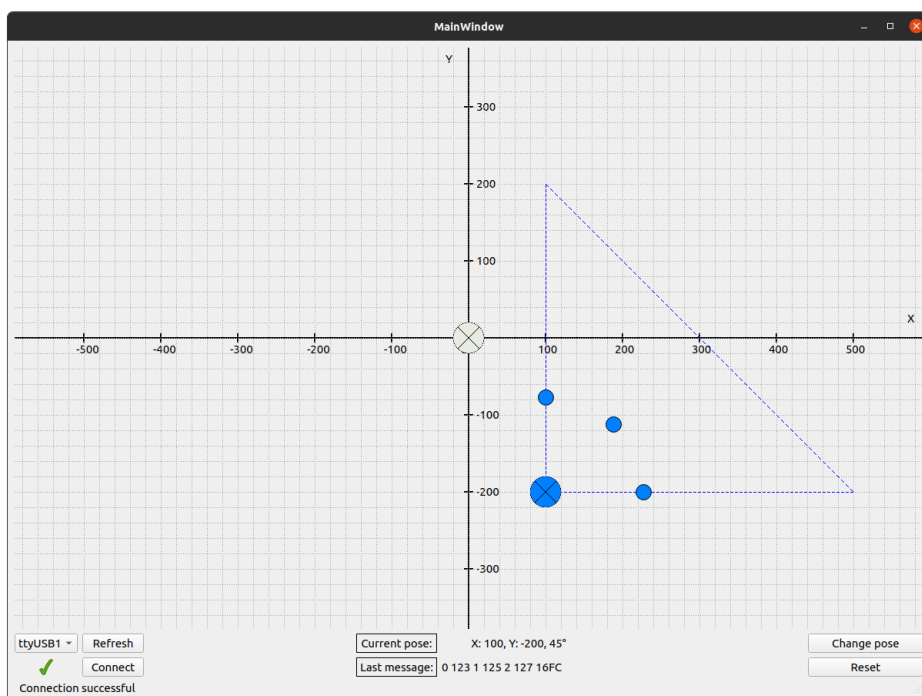
Rysunek 20: Sygnalizacja połączenia

A screenshot of a dialog box titled 'Form' with a close button (X) in the top right corner. The dialog has a light gray background. Inside, the text 'Enter new pose:' is centered. Below this, there are three input fields. The first is labeled 'X:' and contains the value '-100'. The second is labeled 'Y:' and contains the value '321'. The third is labeled 'deg:' and contains the value '90'. At the bottom of the dialog, there are two buttons: 'Cancel' on the left and 'OK' on the right.

Rysunek 21: Okno zmiany pozycji



Rysunek 22: Okno główne



Rysunek 23: Okno główne

8.2 Dokumentacja

Dokumentacja jest kompletna i uzupełniana na bieżąco w czasie tworzenia aplikacji 24 25.

```
#ifndef OBSTACLEPOINT_H
#define OBSTACLEPOINT_H

#include <array>

#include <QWidget>
#include <QLabel>
#include <QPixmap>

/** Klasa zmapowanego punktu przeszkody
 *!
 * Klasa zawierająca informacje o współrzędnych zmapowanego punktu przeszkody
 * oraz pixmap determinujący rodzaj punktu (pobrane z aktualnej pozycji urządzenia
 * czy z jednej z poprzednich)
 */
class ObstaclePoint{
public:

    /** Konstruktor klasy
    *!
    * Przy wywołaniu konstruktora przypisywane są współrzędne
    * punktu oraz pixmap
    */
    ObstaclePoint(const QPixmap& pixmap, std::array<int, 2> position);

    /** Metoda pobierająca tablicę zawierającą informacje o współrzędnych
    *!
    * \return Tablica współrzędnych
    */
    std::array<int, 2> getPos();
};
```

Rysunek 24: Przykład dokumentowania kodu

Dokumentacja funkcji składowych

◆ keyPressedEvent()

void PoseTextEdit::keyPressEvent (QKeyEvent * event) override protected

Metoda nadpisująca funkcję keyPressEvent.

Za pomocą tej metody określone są znaki akceptowalne przez pole tekstowe oraz długość wpisanego wyrażenia

Parametry

event wskaźnik na obiekt typu QKeyEvent

Definicja w linii 5 pliku `posetextedit.cpp`.

```
5 {
6
7     if (event->key() == Qt::Key_Backspace || event->key() == Qt::Key_Delete) {
8         QTextEdit::keyPressEvent(event);
9         return;
10    }
11
12    QString text = toPlainText();
13    if (text.length() >= 4 && event->key() != Qt::Key_Backspace) {
14        event->ignore();
15        return;
16    }
17
18    static QRegularExpression regex("[0-9 \\-]");
19    if (!regex.match(event->text()).hasMatch()) {
20        event->ignore();
21        return;
22    }
23    QTextEdit::keyPressEvent(event);
24 }
25 }
```

Dokumentacja dla tej klasy została wygenerowana z plików:

- Mapowanie2/`posetextedit.h`
- Mapowanie2/`posetextedit.cpp`

Wygenerowano przez **doxygen** 1.8.17

Rysunek 25: Przykład dokumentacji

8.3 Kod źródłowy

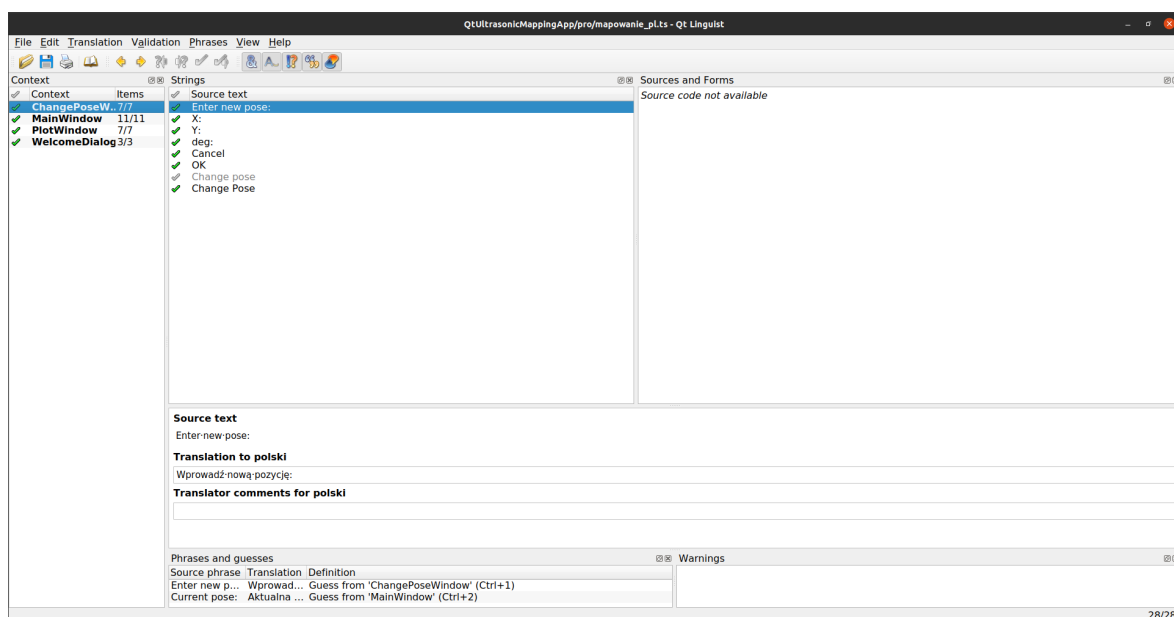
Kod źródłowy jest udostępniony w serwisie GitHub.

9 Rezultaty końcowe

9.1 Zmiany w aplikacji

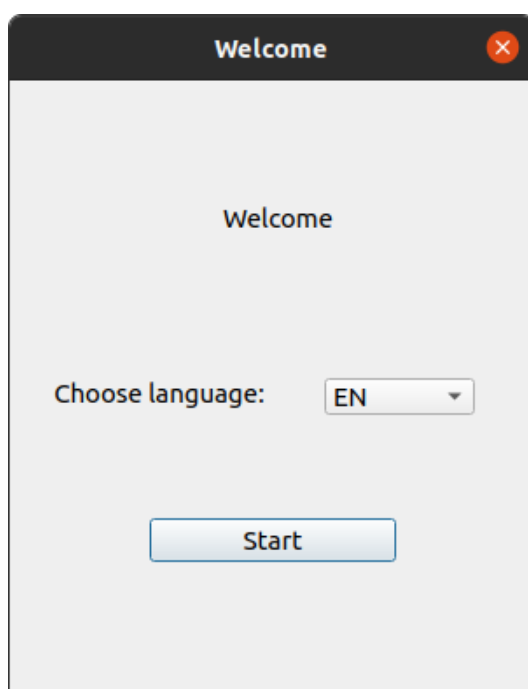
Dodane zostało okno, w którym w czasie rzeczywistym rysowany jest wykres wskazań z czujników 28. Wykorzystano do tego bibliotekę `QCustomPlot`. Okno z wykresem można wywołać z poziomu okna głównego przez odpowiedni przycisk. Za pomocą Qt Luingist dodano tłumaczenie aplikacji 26. Wszystkie wyświetlane teksty zostały zostały sformatowane i przetłumaczone. Zmiana języka jest możliwa poprzez wybranie odpowiedniej pozycji z listy w oknie powitalnym 27 lub oknie głównym aplikacji. Po wybraniu języka z listy, wszystkie wyświetlane teksty są tłumaczone na wybrany język 29 30. Aplikacja jest teraz dostępna w dwóch wersjach językowych: polskiej i angielskiej.

9.1.1 Wygląd aplikacji

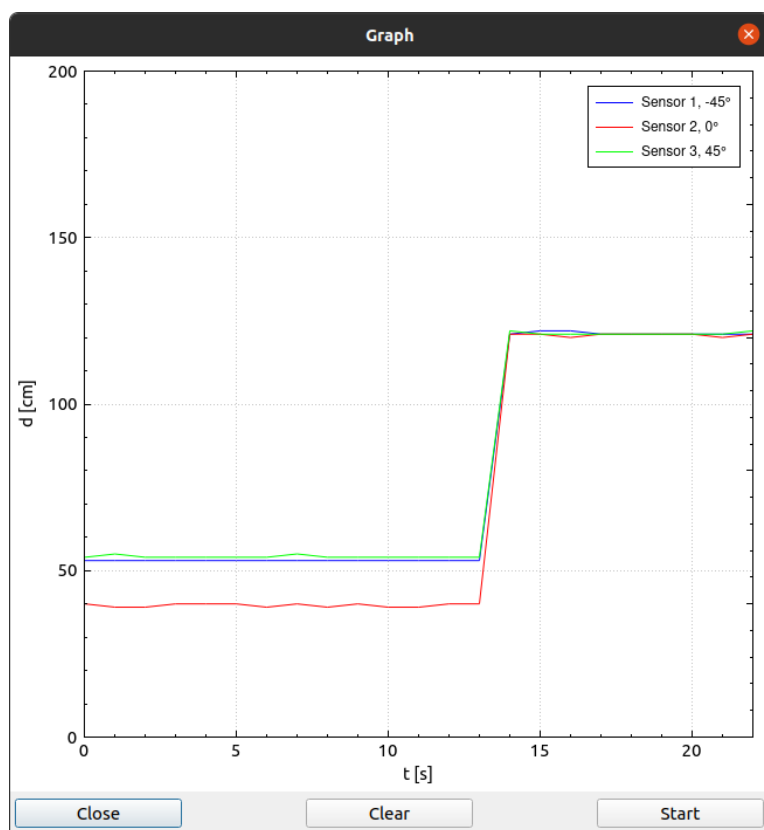


Rysunek 26: Qt Linguist

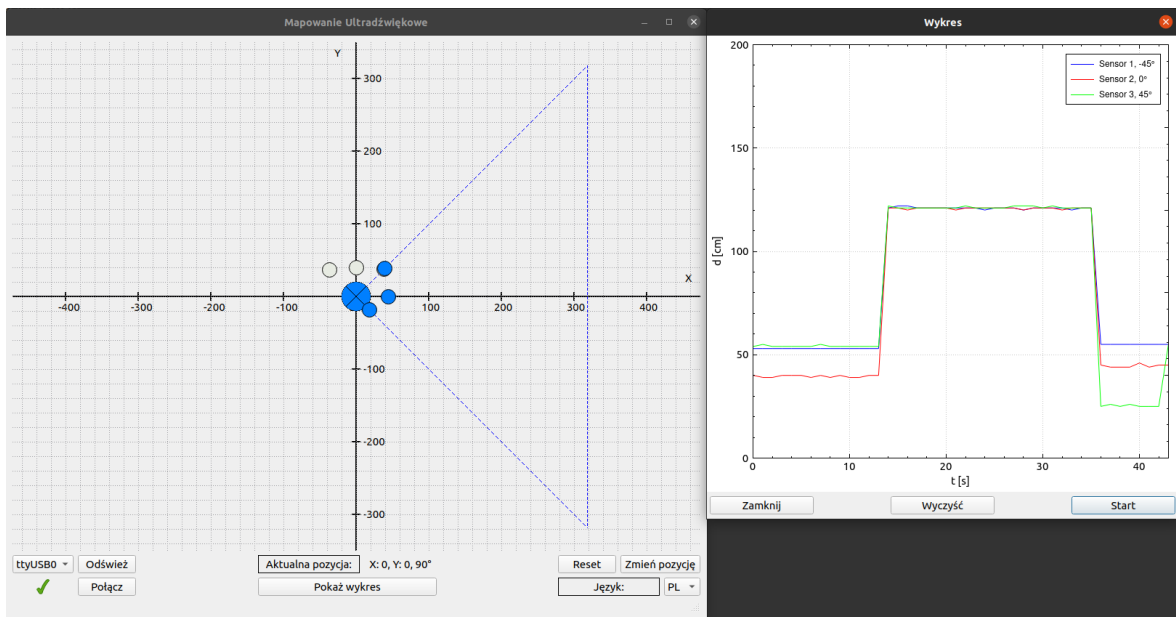
Na poniższych rysunkach przedstawiony jest finalny wygląd aplikacji:



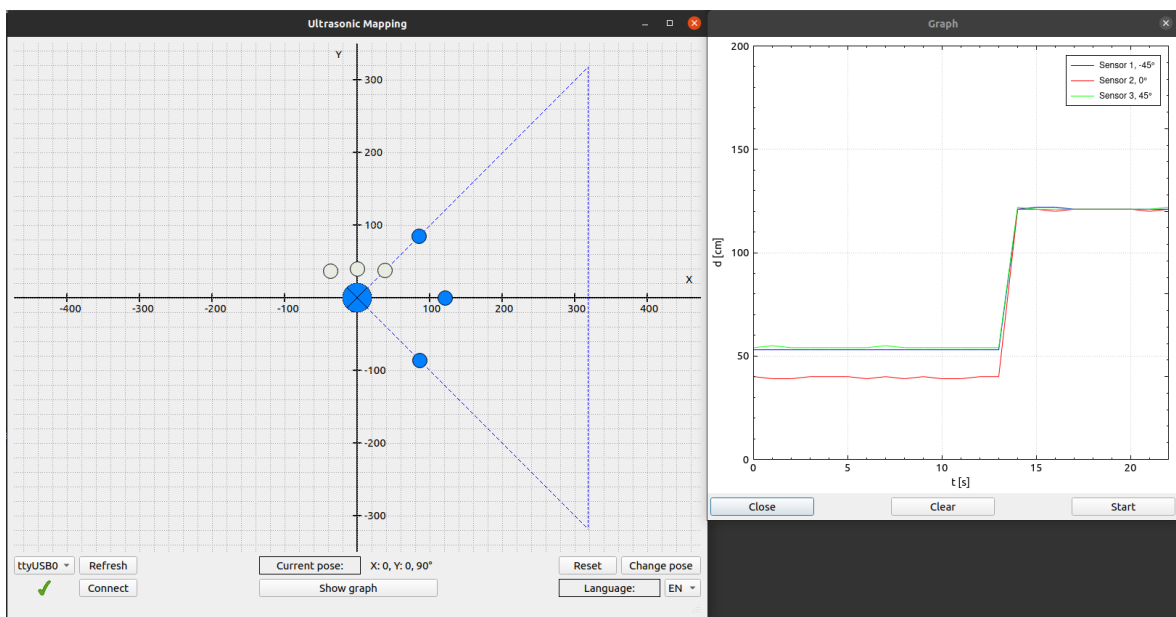
Rysunek 27: Okno powitalne z wyborem języka



Rysunek 28: Okno z wykresem wskazań czujników



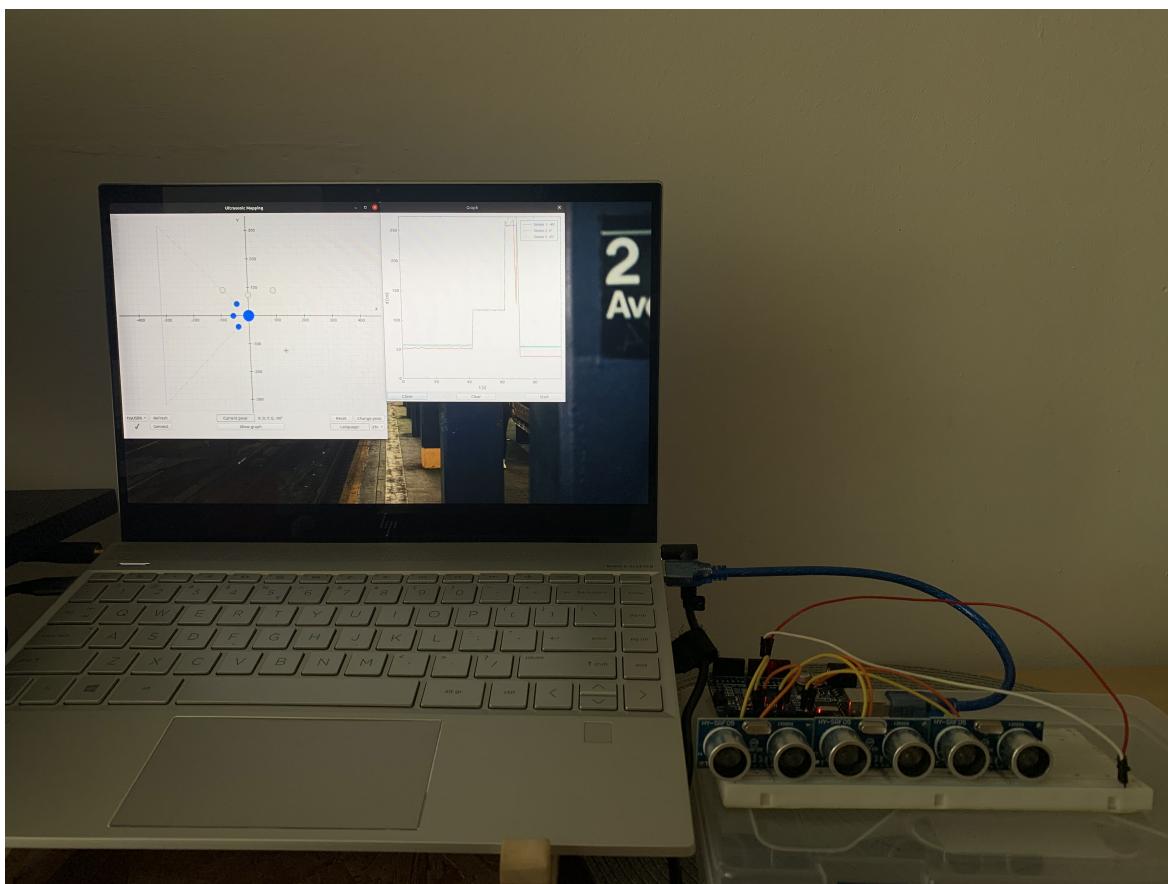
Rysunek 29: Aplikacja w języku polskim



Rysunek 30: Aplikacja w języku angielskim

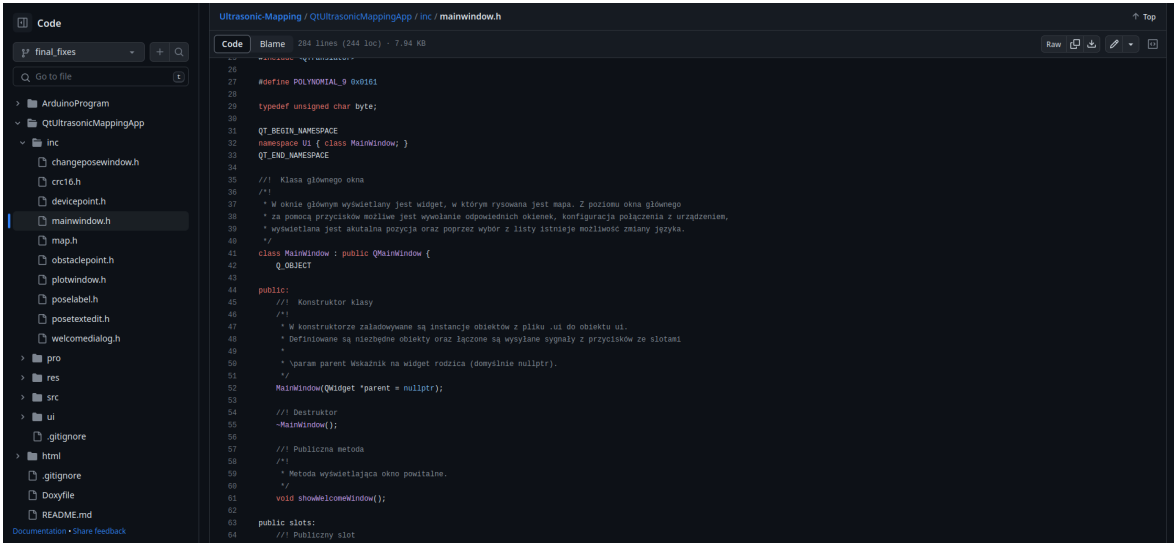
9.2 Efekt końcowy

Założenia projektowe zostały osiągnięte. Rezultatem końcowym jest aplikacja zdolna do wizualizacji mapowania za pomocą czujników ultradźwiękowych, które wysyłają odczyty za pośrednictwem Arduino Uno poprzez port seryjny i odczytywane są z poziomu aplikacji. Odebrane dane są następnie odpowiednio formatowane i obliczana jest ich pozycja w przestrzeni dwuwymiarowej. W czasie rzeczywistym rysowane są wskazania z każdego z czujników a aplikacja jest dostępna w dwóch wersjach językowych: polskiej i angielskiej. Kod źródłowy projektu jest w pełni udokumentowany a dokumentacja została wygenerowana za pomocą narzędzia **Doxygen**. Na poniższym zdjęciu 31 widać działającą aplikację, która wizualizuje dane odczytane z czujników oraz rysuje na ich podstawie wykres w czasie rzeczywistym.



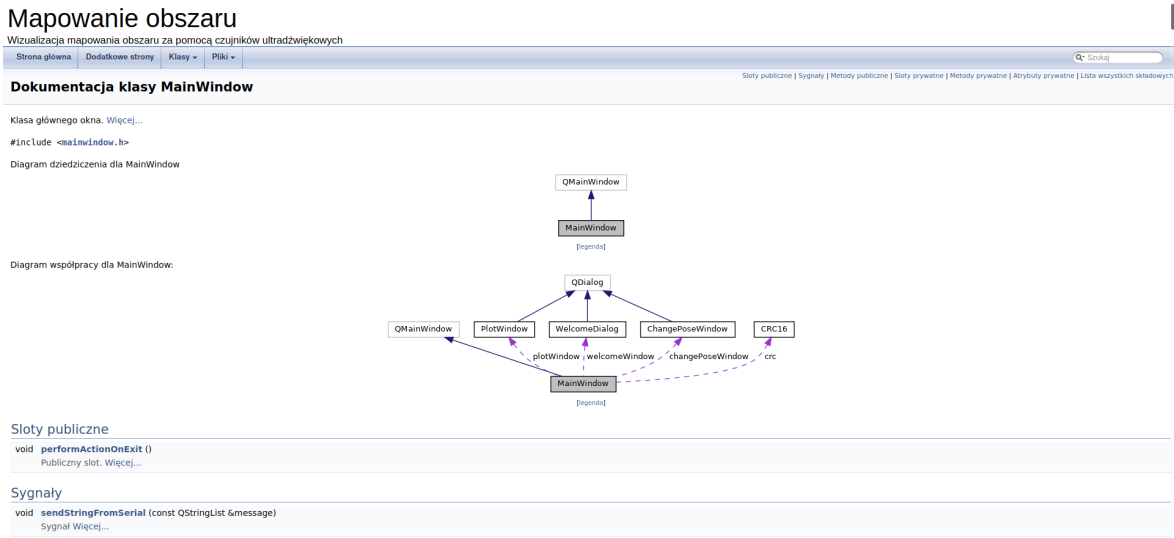
Rysunek 31: Prezentacja aplikacji komunikującej się z podłączonym urządzeniem.

Pełny kod źródłowy 32 jest udostępniony w formie repozytorium Git.



Rysunek 32: Repozytorium Git

Kod został udokumentowany w całości korzystając z Doxygen 33.



Rysunek 33: Dokumentacja Doxygen