Wizualizacja danych sonaru

Dorian Janiak

22.04.2015

1 Krótki opis

Projekt zakłada stworzenie aplikacji komputerowej, która będzie wizualizowała otrzymywane dane z sonaru ultradźwiękowego. Aplikacja ma próbować łączyć dane w taki sposób, aby móc z nich stworzyć zarys otoczenia. Drugą część projektu stanowi symulacja, której zadaniem jest z odpowiednio przygotowanych obiektów geometrycznych wygenerować dane potrzebne do wykonania wizualizacji (ma być symulacją sonaru). Dane mają zostać przesłane przy pomocy wybranego protokołu komunikacyjnego.

2 Cel

Celem projektu jest zapoznanie się ze środowiskiem Qt oraz sposobami wizualizacji danych. Jednym z problemów, który zostanie również poruszony w ramach pracy będzie sposób komunikacji modułów i przesył danych. Jednym z ważnych powodów podjęcia się realizacji tego tematu jest chęć głębszego poznania biblioteki OpenGL.

3 Rozszerzony opis

3.1 Komputer PC

Projekt składa się z dwóch części. Pierwsza część obejmuje stworzenie aplikacji w Qt, która pozwoli na przedstawienie danych sensorycznych z sonaru. Sonar i robot mobilny zostaną stworzone w ramach osobnego projektu (kurs: Roboty mobilne(1)). Aplikacja stworzona w środowisku Qt pozwala również na sterowanie robotem. Dane są wysyłane do robota przy użyciu interfejsu Bluetooth. Robot reaguje na to zmieniając swoją pozycję. Następnie z aplikacji Qt wysyła się żądanie wykonania skanowania terenu. Robot przy pomocy czujnika ultradźwiękowego rozpoczyna skanowanie terenu, zapisując dane w swojej pamięci. Gdy skończy - przesyła dane pomiarowe przez interfejs Bluetooth do komputera z aplikacją. Dane zostają wczytane i zwizualizowane. Należy przy tym pamiętać, że robot nie znajduje się dokładnie w tym miejscu, w którym oczekiwalibyśmy. Dane zostają połączone przez aplikację i wyrysowany zostaje kolejny kawałek mapy otoczenia. Gdyby omiatanie terenu odbywało się również wertykalnie, a nie tylko horyzontalnie możnaby wymagać większej dokładności wizualizacji.

3.2 Urządzenie

Drugą częśc aplikacji stanowi symulator robota z sonarem. Początkowym założeniem było stworzenie symulatora uruchamianego na tym samym komputerze co główna aplikacja, jednak po rozpoznaniu tematu i ponownym oszacowaniu dostępnego czasu projekt symulatora został ograniczony do projektu aplikacji na urządzenie z systemem Android. Aplikacja będzie wczytywać obiekty z plików OBJ oraz ze specjalnych plików konfiguracyjnych położenie poszczególnych obiektów na scenie (proponowany format: JSON). Następnie będzie omiatała otoczenie z pewną dokładnością. Wyniki zostaną

wysłane poprzez interfejs Bluetooth. Posłużą one do zwizualizowania efektów symulacji w oknie aplikacji z pierwszej części. Ostatnim etapem rozwoju symulatora będzie stworzenie ciągłej komunikacji między symulatorem i aplikacją wizualizującą, tak aby można było sterować robotem umieszczonym w symulatorze.

4 Funkcjonalność skończona

Poniżej zamieszczona została lista funkcjonalności, którą udało się ukończyć dotychczas. Lista zawiera również nieplanowany wcześniej punkt. Został on zrealizowany w celu ułatwienia późniejszego debugowania i wyszukiwania problemów w trakcie komunikacji z urządzeniem. Klasa (MessageController) została specjalnie stworzona w taki sposób, aby dziedzicząc ją była w stanie obsłużyć zarówno dane pochodzące z pliku dyskowego jak i dane pochodzące z transmisji Bluetooth.

4.1 Aplikacja na komputerze PC

- Rysowanie na scenie 3D mapy otoczenia.
- Wczytywanie danych pomiarowych sonaru z pliku (w przypadku symulacji).
- Łączenie nowo odczytanej z poprzednio odczytaną mapą terenu.
- Możliwość sterowania widokiem 3D.
- *(Nieplanowane wcześniej)* Wyświetlanie logów dotyczących przesyłanych komunikatów oraz interpretacja występujących błędów.

5 Funkcjonalność planowana

Poniżej zamieszczone zostały listy funkcjonalności, które zamierzam wykonać w ramach projektu.

5.1 Aplikacja na komputerze PC

- Rysowanie na scenie 3D robota.
- Udostępnienie pilota pozwalającego na sterowanie robotem (lub symulacją).
- Wysyłanie żądania przemieszczenia robota (poprzez interfejs komunikacyjny).
- Wysyłanie żądania wykonania skanowania terenu (poprzez interfejs komunikacyjny).
- Odbieranie danych z sonaru robota (poprzez interfejs komunikacyjny).

5.2 Symulator na urządzeniu

- Ładowanie plików konfiguracyjncyh (zawierają informacje o położeniu obiektów 3D na symulowanej scenie)
- Ładowanie plików OBJ
- Prosta symulacja sonaru ultradźwiękowego
- Wysłanie danych do aplikacji wizualizującej poprzez interfejs komunikacyjny (Bluetooth)
- Odbieranie żądania przemieszczenia robota na scenie
- Odbieranie żądania wykonania symulacji i przesłania wyników

6 Funkcjonalność zawieszona

Funkcjonalność wymieniona w tym punkcie została zawieszona ze względu na ograniczenia czasowe oraz stosunkowo małą jej przydatność w programie. Symulator stworzony na komputerze PC przy poniższych założeniach nie jest w stanie dorównać symulatorowi zainsalowanemu na urządzeniu zewnętrznym, z którym program będzie musiał komunikować się poprzez interfejs Bluetooth. Czas, który pierwotnie miał być poświęcony na jej stworzenie zostanie przeznaczony na stworzenie symulatora na telefonie z systemem Android.

6.1 Symulator na komputerze PC

- Ładowanie plików konfiguracyjnych (zawierają informacje o położeniu obiektów 3D na symulowanej scenie)
- Ładowanie plików OBJ
- Symulacja sonaru ultradźwiękowego
- Zapisanie danych do z góry ustalonego pliku wynikowego (np. out.txt)

7 Zaktualizowana lista kamieni milowych

- K1 - zakłada działającą podstawową aplikację komputerową. Aplikacja jest w stanie rysować wszystkie potrzebne do symulacji obiekty 3D. Udostępnia również prosty pilot. Jest w stanie załadować prosty plik symulacyjny. Pozwala na sterowanie widokiem 3D.
- K2 - zakłada podstawową formę komunikacji między urządzeniem opartym o system Android, a aplikacją komputerową. Nie zakłada pełnej interakcji, ale urządzenie jest w stanie przesłać takie podstawowe informacje jak pozycja robota oraz wynik pomiaru danych. Niekoniecznie obsługiwana jest jeszcze pozycja zadawana przez aplikację komputerową.
- K3 - zakłada stworzone: aplikację oraz symulator. Obie są w stanie się ze sobą komunikować oraz dodatkowo aplikacja komputerowa komunikuje się z robotem.

8 Harmonogram

Od:	**Do:**	**Harmonogram tworzenia głównej aplikacji**	**Harmonogram tworzenia symulatora**	**Kamień milowy**	**Status**
23.03	29.03	Stworzenie szkieletu aplikacji QT z ładowaniem okna 3D.			Skończone
30.03	05.04	Ładowanie danych symulacyjnych z pliku i proste rysowanie mapy w oknie 3D.			Skończone
06.04	12.04		Ładowanie plików OBJ i konfiguracji sceny.		Nierozpoczęte
13.04	19.04		Logika symulatora - omiatanie terenu i eksport wyników do pliku.	K1	Nierozpoczęte

20.04	26.04	Stworzenie prostej aplikacji w QT obsługującej Bluetooth.	Stworzenie prostej aplikacji obsługującej Bluetooth na Androidzie.		
27.04	03.05		Przeniesienie logiki do aplikacji na Androidzie.		
04.05	10.05	Stworzenie komunikacji przez Bluetooth	Przeniesienie logiki do aplikacji na Androidzie	K2	
11.05	17.05	Dostosowanie aplikacji do robota.			
18.05	24.05	System poruszania robotem.	Stworzenie reakcji symulatora na sterowanie robotem.		
25.05	31.05	Sklejanie mapy 3D oraz wizualizacja ścieżki ruchu robota.			Częściowo skończone
01.06	07.06	Możliwość poruszania widokiem 3D			Skończone
08.06	14.06	Stosowanie poprawek.	Stosowanie poprawek	K3	

9 Zaktualizowany harmonogram

Od:	**Do:**	**Harmonogram tworzenia głównej aplikacji**	**Harmonogram tworzenia symulatora**	**Kamień milowy**	**Status**
23.03	29.03	Stworzenie szkieletu aplikacji QT z ładowaniem okna 3D.			Skończone
30.03	05.04	Przygotowanie diagramu klas oraz diagramu przypadków użycia.			Skończone
06.04	12.04	Rysowanie siatki, możliwość poruszania widokiem 3D. Parsowanie pliku symulacyjnego.			Skończone
13.04	19.04	Rysowanie mapy otoczenia oraz obsługa błędów i formatu wiadomości. Wyświetlanie logów komunikacji.			Skończone
20.04	26.04	Rysowanie robota w oknie 3D oraz obsługa zmiany jego położenia.		K1	
27.04	03.05		Zapoznanie z Android SDK.		
04.05	10.05	Stworzenie komunikacji przez Bluetooth. Parowanie urządzeń.	Stworzenie prostej komunikacji przez Bluetooth. Parowanie urządzeń.		
11.05	17.05	Obsługa przesyłanych wiadomości poprzez Bluetooth.	Stworzenie symulatora sonaru. Ładowanie plików OBJ.	K2	

18.05	24.05	Obsługa przesyłanych wiadomości (z telefonu oraz robota)	Dopracowanie symulatora zgodnie ze sposobem działania robota.		
25.05	31.05	Poprawki obsługi modułu Bluetooth oraz synchronizacji widoku.	Dopracowanie symulatora zgodnie ze sposobem działania robota.		
01.06	07.06	Dostosowanie aplikacji do możliwości robota.			
08.06	14.06	Stosowanie poprawek.	Stosowanie poprawek	K3	

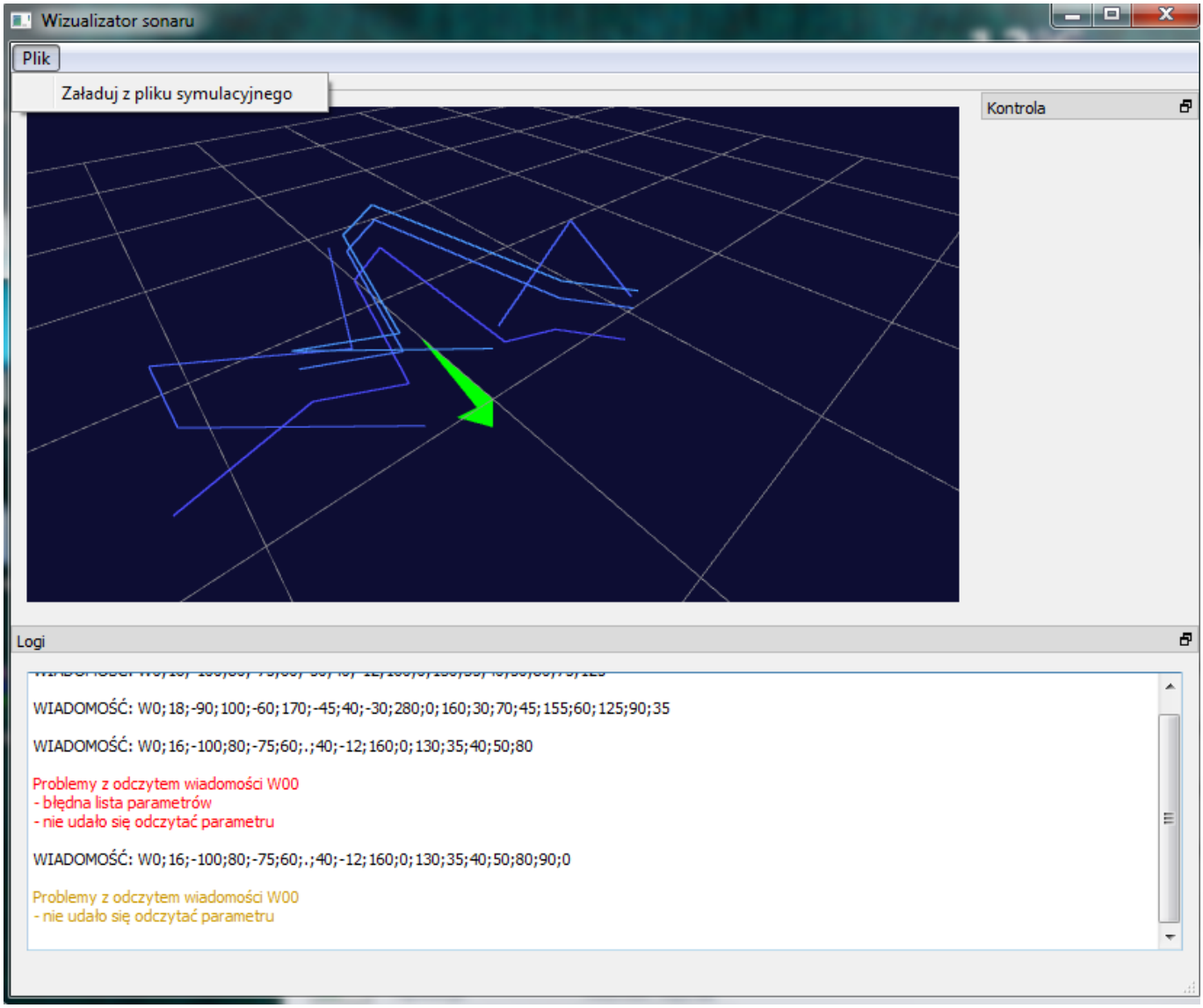
10 Wygląd aplikacji

Poniżej prezentuję aktualnie uzyskany wygląd głównej aplikacji. W ostatecznej wersji aplikacji zostanie on wzbogacony o przyciski w dokowanym widżecie "kontroler", które pozwolą na sterowanie pozycją robota. W górnej części okna znajduje się menu, w którym znajdą się między innymi takie zakładki jak:

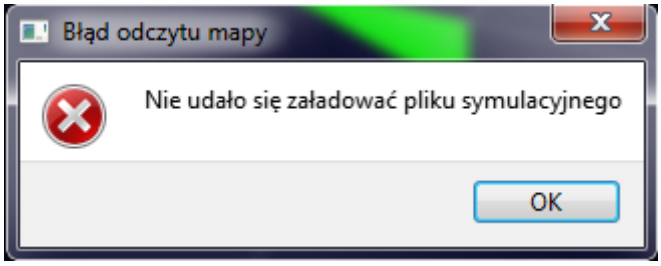
- "Plik- pozwoli na ładowanie z pliku symulacji oraz w razie decyzji dalszego rozwoju aplikacji na zapis i otwieranie różnego formatu plików
- "Bluetooth- pozwoli na parowanie urządzenia z aplikacją
- "Pomoc- będzie otwierało okno z informacjami o autorze.

Główną część okna stanowi widok 3D, w którym przy użyciu myszy komputerowej można sterować kątem kamery oraz jej przybliżeniem. W oknie tym będzie rysowana mapa 3D. Na screenie widać poszczególne jej składowe (kolorowe linie) z tym, że wszystkie zostały wyrysowane względem tego samego punktu środkowego (0,0,0,1) we współrzędnych jednorodnych. Poniżej głównego okna znajduje się dokowany widżet, w którym zapisywane będą logi z aktualnie odbywającej się komunikacji. Zostały wyróżnione różnego typu komunikaty:

- kolor czarny - zwykła informacja
- kolor pomarańczowy - ostrzeżenie
- kolor czerwony - błąd (ale nie krytyczny)

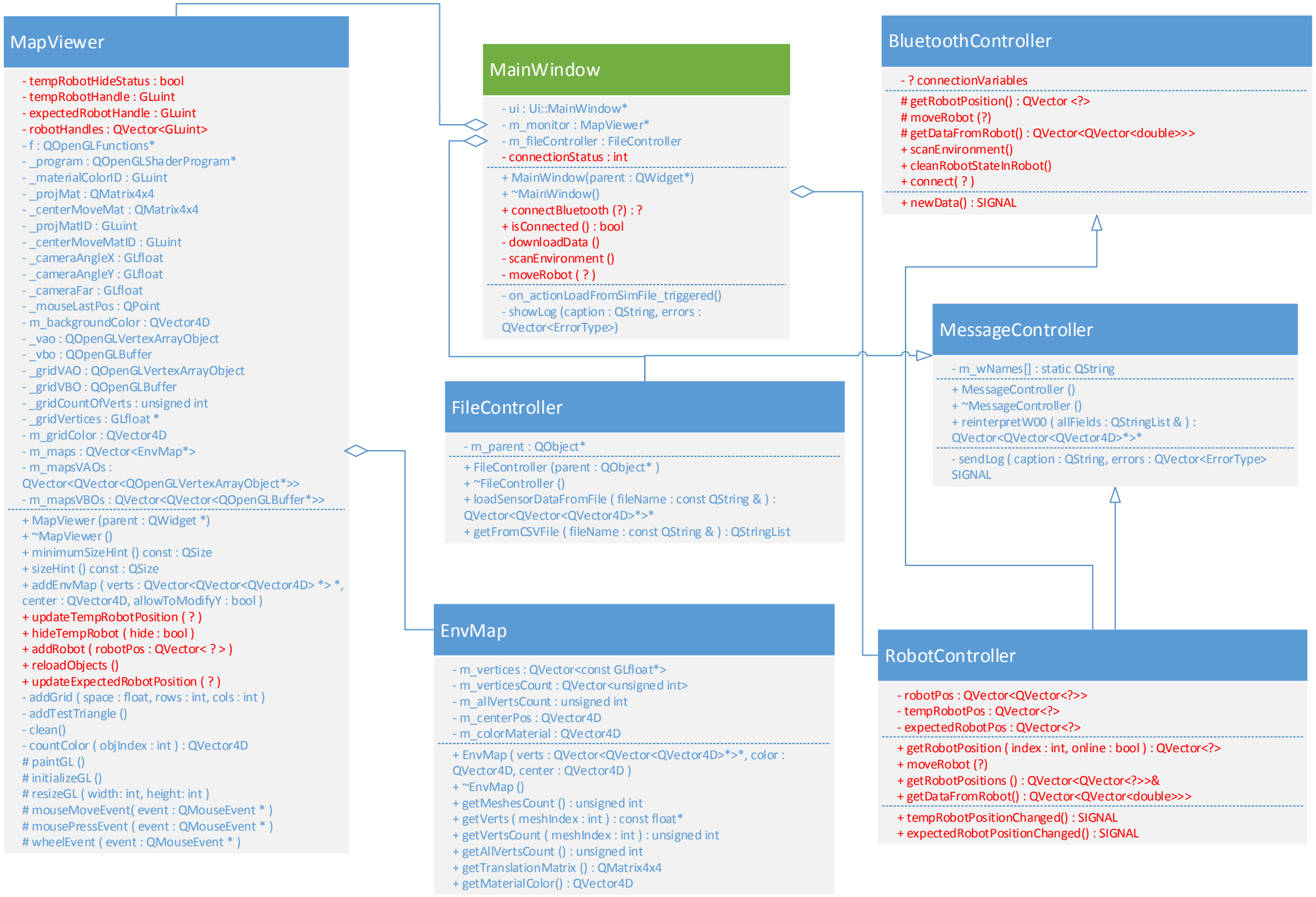


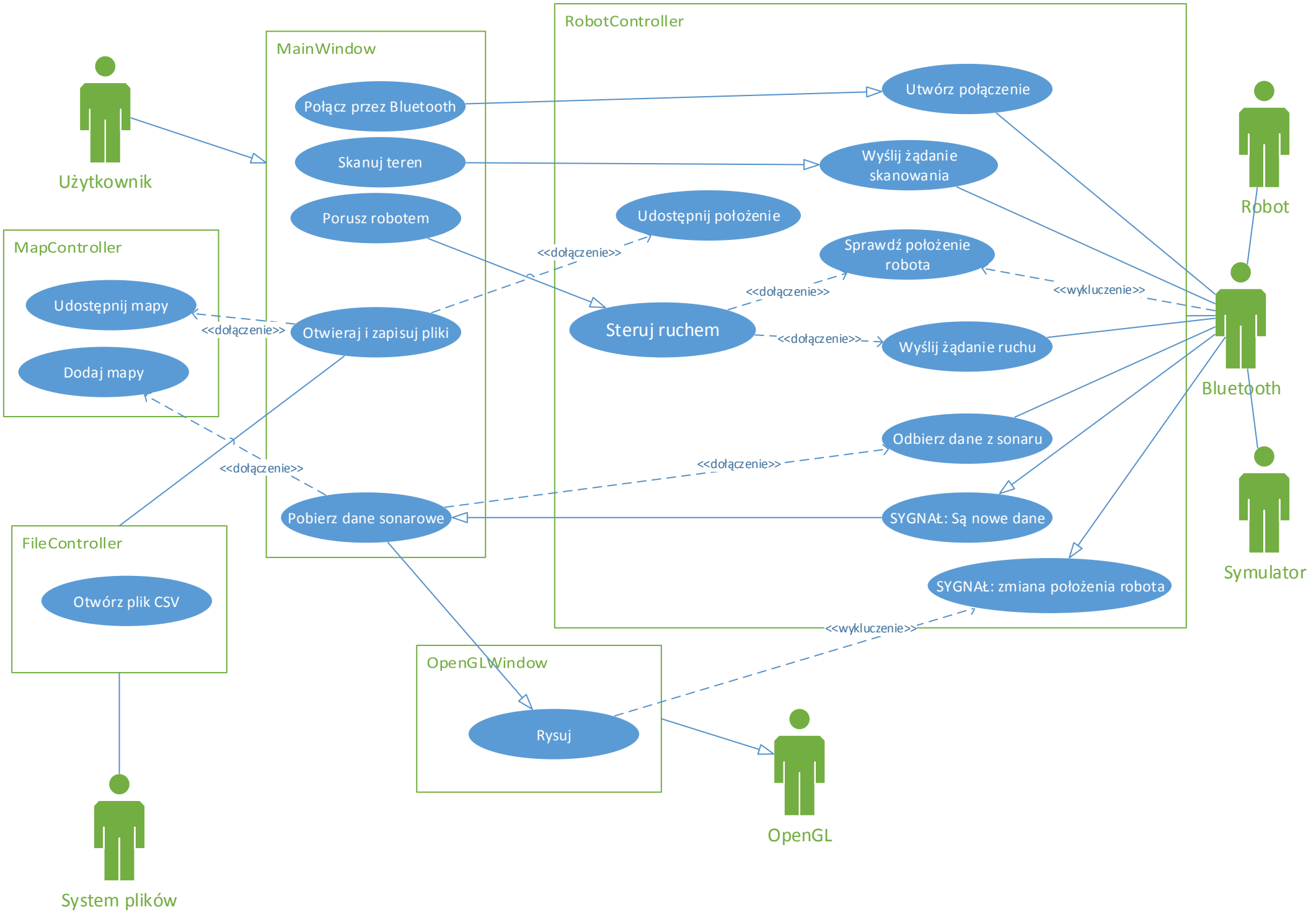
Poważniejsze błędy, wymagające uwagi użytkownika są raportowane okienkiem błędu.



11 Diagramy

Metody oraz pola poniższego diagramu zaznaczone kolorem czerwonym oznaczają elementy, których jeszcze nie stworzyłem. Pozostałe są już stworzone. Główną klasą jest klasa MainWindow, która odpowiada głównemu oknu aplikacji. Przechowuje ona obiekty pozostałych klas. MessageController odpowiada za interpretowanie i przygotowywanie wiadomości potrzebnych do komunikacji z urządzeniami oraz odczytem danych z plików. Będzie ona dziedziczona przez klasy FileController (obsługuje operacje plikowe) oraz RobotController (zarządza robotem). W przypadku tej drugiej klasy odziedziczy ona również po klasie BluetoothController. Zapewni to klasie RobotController komplet funkcji potrzebnych do sterowania robotem. Klasa MapViewer dziedziczy od klasy QOpenGLWidget i odpowiada za sterowanie widokiem 3D.


MapViewer
- tempRobotHideStatus : bool
- tempRobotHandle : GLuint
- expectedRobotHandle : GLuint
- robotHandles : QVector<GLuint>
- f : QOpenGLFunctions*
- _program : QOpenGLShaderProgram*
- _materialColorID : GLuint
- _projMat : QMatrix4x4
- _centerMoveMat : QMatrix4x4
- _projMatID : GLuint
- _centerMoveMatID : GLuint
- _cameraAngleX : GLfloat
- _cameraAngleY : GLfloat
- _cameraFar : GLfloat
- _mouseLastPos : QPoint
- m_backgroundColor : QVector4D
- _vao : QOpenGLVertexArrayObject
- _vbo : QOpenGLBuffer
- _gridVAO : QOpenGLVertexArrayObject
- _gridVBO : QOpenGLBuffer
- _gridCountOfVerts : unsigned int
- _gridVertices : GLfloat *
- m_gridColor : QVector4D
- m_maps : QVector<EnvMap*>
- m_mapsVAOs : QVector<QVector<QOpenGLVertexArrayObject*>>
- m_mapsVBOs : QVector<QVector<QOpenGLBuffer*>>
+ MapViewer (parent : QWidget *)
+ ~MapViewer ()
+ minimumSizeHint () const : QSize
+ sizeHint () const : QSize
+ addEnvMap (verts : QVector<QVector<QVector4D> *> *, center : QVector4D, allowToModifyY : bool)
+ updateTempRobotPosition (?)
+ hideTempRobot (hide : bool)
+ addRobot (robotPos : QVector< ? >)
+ reloadObjects ()
+ updateExpectedRobotPosition (?)
- addGrid (space : float, rows : int, cols : int)
- addTestTriangle ()
- clean()
- countColor (objIndex : int) : QVector4D
paintGL ()
initializeGL ()
resizeGL (width: int, height: int)
mouseMoveEvent(event : QMouseEvent *)
mousePressEvent (event : QMouseEvent *)
wheelEvent (event : QMouseEvent *)
MainWindow
- ui : Ui::MainWindow*
- m_monitor : MapViewer*
- m_fileController : FileController
- connectionStatus : int
+ MainWindow(parent : QWidget*)
+ ~MainWindow()
+ connectBluetooth (?) : ?
+ isConnected () : bool
- downloadData ()
- scanEnvironment ()
- moveRobot (?)
- on_actionLoadFromSimFile_triggered()
- showLog (caption : QString, errors : QVector<ErrorType>)
FileController
- m_parent : QObject*
+ FileController (parent : QObject*)
+ ~FileController ()
+ loadSensorDataFromFile (fileName : const QString &) : QVector<QVector<QVector4D>*>*
+ getFromCSVFile (fileName : const QString &) : QStringList
EnvMap
- m_vertices : QVector<const GLfloat*>
- m_verticesCount : QVector<unsigned int>
- m_allVertsCount : unsigned int
- m_centerPos : QVector4D
- m_colorMaterial : QVector4D
+ EnvMap (verts : QVector<QVector<QVector4D>*>*, color : QVector4D, center : QVector4D)
+ ~EnvMap ()
+ getMeshesCount () : unsigned int
+ getVerts (meshIndex : int) : const float*
+ getVertsCount (meshIndex : int) : unsigned int
+ getAllVertsCount () : unsigned int
+ getTranslationMatrix () : QMatrix4x4
+ getMaterialColor() : QVector4D
BluetoothController
- ? connectionVariables
getRobotPosition() : QVector <?>
moveRobot (?)
getDataFromRobot() : QVector<QVector<double>>>
+ scanEnvironment()
+ cleanRobotStateInRobot()
+ connect(?)
+ newData() : SIGNAL
MessageController
- m_wNames[] : static QString
+ MessageController ()
+ ~MessageController ()
+ reinterpretW00 (allFields : QStringList &) : QVector<QVector<QVector4D>*>*
- sendLog (caption : QString, errors : QVector<ErrorType> SIGNAL
RobotController
- robotPos : QVector<QVector<?>>
- tempRobotPos : QVector<?>
- expectedRobotPos : QVector<?>
+ getRobotPosition (index : int, online : bool) : QVector<?>
+ moveRobot (?)
+ getRobotPositions () : QVector<QVector<?>>&
+ getDataFromRobot() : QVector<QVector<double>>>
+ tempRobotPositionChanged() : SIGNAL
+ expectedRobotPositionChanged() : SIGNAL


RobotController
MainWindow
Użytkownik
Połącz przez Bluetooth
Skanuj teren
Porusz robotem
Utwórz połączenie
Wyślij żądanie skanowania
Udostępnij położenie
Sprawdź położenie robota
Steruj ruchem
Wyślij żądanie ruchu
Odbierz dane z sonaru
SYGNAŁ: Są nowe dane
SYGNAŁ: zmiana położenia robota
Pobierz dane sonarowe
<<dołączenie>>
<<wykluczenie>>
Robot
Bluetooth
Symulator
MapController
Udostępnij mapy
Dodaj mapy
Otwieraj i zapisuj pliki
FileController
Otwórz plik CSV
System plików
OpenGLWindow
Rysuj
OpenGL