Kompas elektroniczny

Dawid Brząkała

1 Cel projektu

Celem projektu jest stworzenie wizualizacji kompasu w 3D oraz wizualizacji danych pomiarowych w postaci wykresu. Dane będą dostarczane do programu z zewnętrznego układu z mikroprocesorem oraz akcelerometrem i magnetometrem co pozwoli na uzyskanie danych o przechyleniu urządzenia oraz kierunku południka magnetycznego.

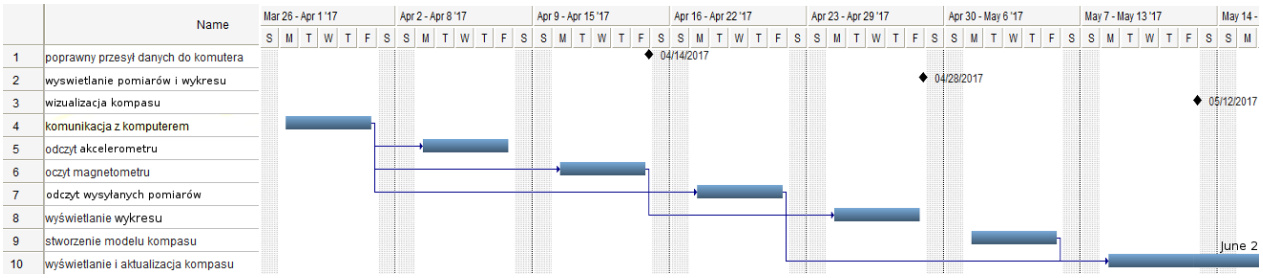
2 Założenia projektu

Układ pomiarowy zostanie zaprogramowany na płytce rozwojowej STM32F3 Discovery wyposażonej w peryferia wymagane do projektu. Odczyt pomiarów, jak i konfiguracja akcelerometru oraz magnetometru realizowana jest poprzez magistralę I^2C. Komunikacja układu z komputerem odbędzie się poprzez kabel USB i protokół RS-232. Dane wysyłane będą bitowo, przeznaczając 4 bajty na każdy pomiar i sumę kontrolną typu float. Ramka z danymi rozpoczęta zostanie przez znak 'X' oraz zakończona sumą kontrolną, będącą sumą wartości przesyłanych pomiarów.

3 Funkcjonalności aplikacji

- Odczyt informacji z czujników realizowany przez układ z mikroprocesorem:
  układ odbiera odczyty z czujnika LSM303DLHC poprzez magistralę I^2C, a następnie odczytana wartość jest skalowana, aby odpowiadała zakresowi pomiaru
- Przesył pomiarów z urządzenia do komputera:
  Przesył przy wykorzystaniu protokołu RS-232. Ramka danych składa się ze znaku rozpoczynającego 'X', sześciu danych pomiarowych oraz sumy kontrolnej typu float przesyłanych bitowo. Suma kontrolna jest sumą uzyskanych pomiarów.
  Program komputerowy natomiast po otrzymaniu danych sprawdza poprawność znaku początkowego ramki oraz sumy kontrolnej. Gdy odebrane dane są poprawne, odczyty czujników kopiowane są to pamięci.
- Wyświetlanie wartości odbieranych pomiarów w postaci tekstowej:
  odczyty odebrane przez port szeregowy są cyklicznie przekazywane do wątku głównego i wyświetlane w widgecie.
- Wyświetlanie wykresu przedstawiającego zmianę azymutu względem czasu:
  wykres jest cyklicznie aktualizowany o wartość obliczoną na podstawie przechowywanych pomiarów.
- Trójwymiarowa wizualizacja kompasu przy użyciu biblioteki openGL, w postaci tarczy z podziałką kątową oraz igły magnetycznej o modyfikowanej orientacji w zależności od odczytanych pomiarów:
  obiekt utworzony w formacie .obj zostaje odpowiednio przetworzony i wczytany do prograu, następnie w regularnych odstępach czasu obliczana jest orientacja układu z mikrokontrolerem i zmieniana zostaje orientacja obiektu 3D.

4 Harmonogram



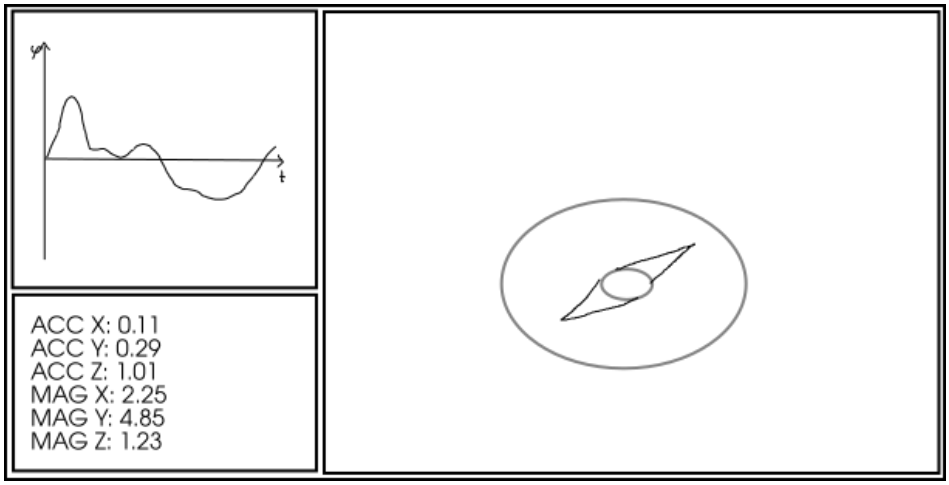
5 Kamienie milowe

- Poprawny przesył danych do komputera
- Wyświetlanie pomiarów oraz wykresu
- Wizualizacja kompasu

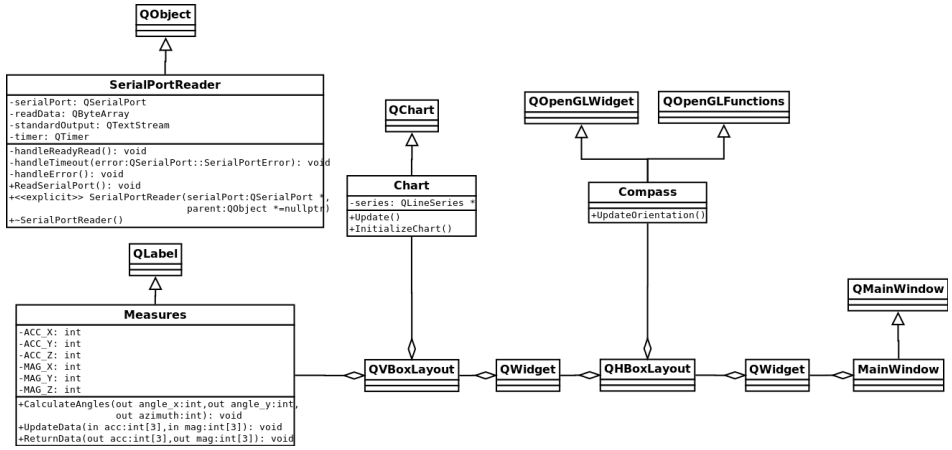
6 Zrealizowane zadania

- Komunikacja z komputerem
- Odczyt akcelerometru
- Odczyt magnetometru
- Odczyt wysłanych pomiarów
- Wyświetlanie wykresu
- stworzenie modelu kompasu

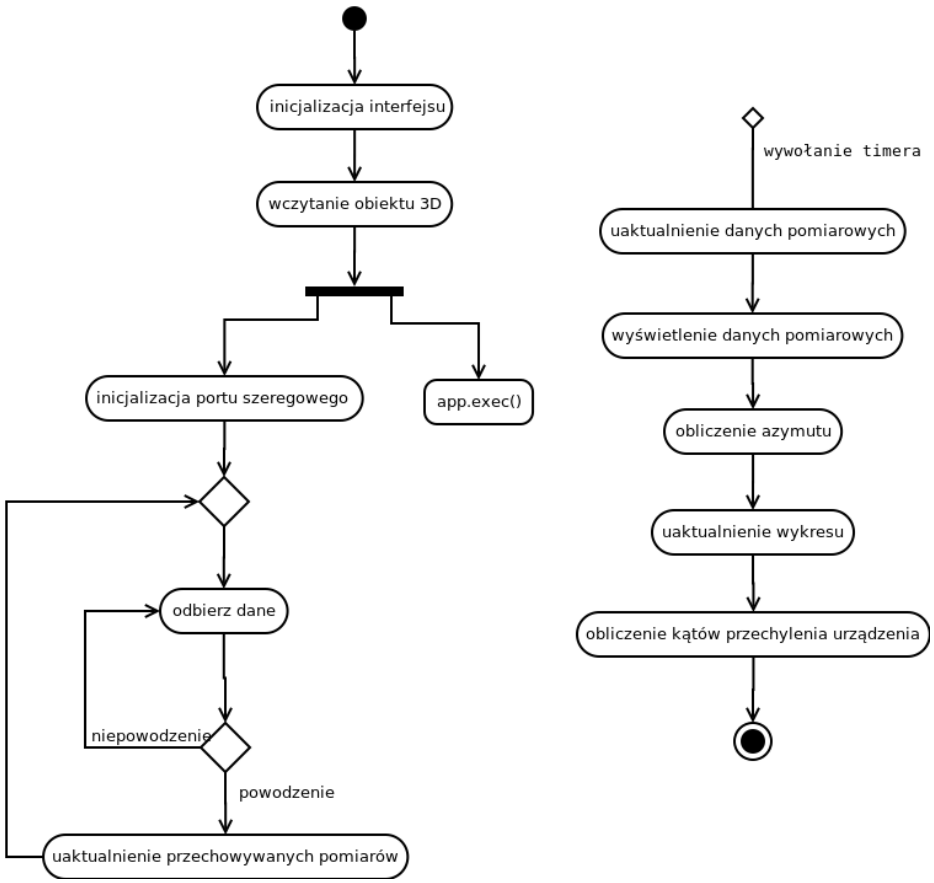
7 Szkic interfejsu graficznego programu



8 Diagram UML



9 Schemat blokowy



10 Dokumentacja Doxygen

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

Rozdział 2

Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy Chart

klasa obsługująca wykres

```
#include <chart.h>
```

Diagram dziedziczenia dla Chart

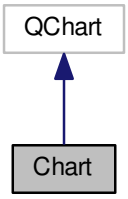
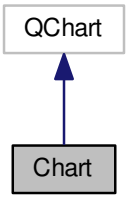


Diagram współpracy dla Chart:



Sloty publiczne

- void Update ()

Metody publiczne

- **Chart** (QGraphicsItem *parent=0)

3.1.1 Opis szczegółowy

klasa obsługująca wykres

klasa umożliwia aktualizację wykresu

3.1.2 Dokumentacja funkcji składowych

3.1.2.1 void Chart::Update ( ) [slot]

slot umożliwiający aktualizację wykresu w czasie

Dokumentacja dla tej klasy została wygenerowana z plików:

- chart.h
- chart.cpp

3.2 Dokumentacja klasy Compass

Klasa obsługująca obiekt 3D.

`#include <compass.h>`

Diagram dziedziczenia dla Compass

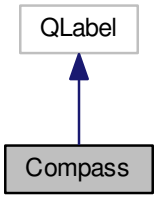
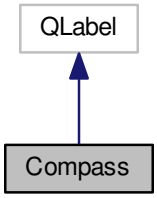


Diagram współpracy dla Compass:



Metody publiczne

- **Compass** (QWidget ∗parent=0)

3.2.1 Opis szczegółowy

Klasa obsługująca obiekt 3D.

klasa odpowada za wczytanie obiektu 3D oraz jego wyświetlanie oraz zmianę orientacji

Dokumentacja dla tej klasy została wygenerowana z plików:

- compass.h
- compass.cpp

3.3 Dokumentacja unii Data

struktura przechowująca odczyty

```
#include <measures.h>
```

Atrybuty publiczne

- float **data** [6]
- struct {
  float ACC_X
  odczyt akcelerometru dla osi x
  float ACC_Y
  odczyt akcelerometru dla osi y
  float ACC_Z
  odczyt akcelerometru dla osi z
  float MAG_X
  odczyt magnetometru dla osi x
  float MAG_Z
  odczyt magnetometru dla osi z
  float MAG_Y
  odczyt magnetometru dla osi y
  };

3.3.1 Opis szczegółowy

struktura przechowująca odczyty

Dokumentacja dla tej unii została wygenerowana z pliku:

- measures.h

3.4 Dokumentacja unii data_t

struktura przechowująca odbieraną ramkę

```
#include <types.h>
```

Atrybuty publiczne

- struct {
  uint8_t flag
  znak początkowy ramki
  datatype_t ACC_Data [3]
  odczyty akcelerometru
  datatype_t MAG_Data [3]
  odczyty magnetometru
  datatype_t checksum
  suma kontrolna
  };
- struct {
  uint8_t **f**
  datatype_t **data** [7]
  };
- uint8_t **buffer** [1+7 ∗sizeof(datatype_t)]

3.4.1 Opis szczegółowy

struktura przechowująca odbieraną ramkę

Dokumentacja dla tej unii została wygenerowana z pliku:

- types.h

3.5 Dokumentacja klasy MainWindow

klasa okna głównego

`#include <mainwindow.h>`

Diagram dziedziczenia dla MainWindow

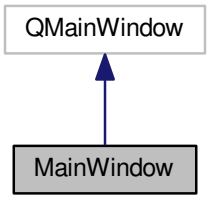
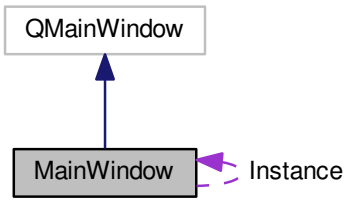


Diagram współpracy dla MainWindow:



Metody publiczne

- MainWindow (float ∗source, QWidget ∗parent=0)

  konstruktor inicjalizujący interfejs programu
- Measures ∗ getMeasures () const

  akceesor zmiennej measures

Statyczne atrybuty publiczne

- static MainWindow ∗ Instance

  zmienna statyczna będąca wskaźnikiem na instancję okna głównego

3.5.1 Opis szczegółowy

klasa okna głównego

klasa inicjalizuje interfejs programu oraz łączy sygnały i sloty dla aktualizacji pomiarów, wykresu oraz obiektu 3D

3.5.2 Dokumentacja konstruktora i destruktora

3.5.2.1 MainWindow::MainWindow (float * *source*, QWidget * *parent* = 0) [explicit]

konstruktor inicjalizujący interfejs programu

Parametry

in	*source*	wskaźnik na dane modyfikowane przez port szeregowy

3.5.3 Dokumentacja funkcji składowych

3.5.3.1 Measures * MainWindow::getMeasures ( ) const

akceesor zmiennej measures

Zwraca

wskaźnik na widget Measures

Dokumentacja dla tej klasy została wygenerowana z plików:

- mainwindow.h
- mainwindow.cpp

3.6 Dokumentacja klasy Measures

klasa wyświetlająca i przetwarzająca pomiary

`#include <measures.h>`

Diagram dziedziczenia dla Measures

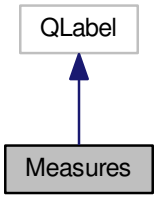
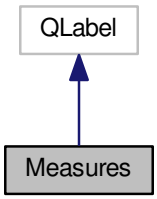


Diagram współpracy dla Measures:



Sloty publiczne

- void UpdateData ()

Metody publiczne

- Measures (float ∗source, QWidget ∗parent=0)

  konstruktor klasy Measures
- void CalculateAngles (float ∗roll, float ∗pitch, float ∗heading)

  funkcja oblicza orientację urządzenia na podstawie przechowywanych pomiarów
- void ReturnData (float ∗data)

  akcesor danych pomiarowych

3.6.1 Opis szczegółowy

klasa wyświetlająca i przetwarzająca pomiary

klasa wyświetla otrzymane poiary oraz pozwala na ich przetwarzanie

3.6.2 Dokumentacja konstruktora i destruktora

3.6.2.1 Measures::Measures (float ∗ *source,* QWidget ∗ *parent* = 0)

konstruktor klasy Measures

Parametry

in	*source*	zródło danych wejściowych

3.6.3 Dokumentacja funkcji składowych

3.6.3.1 void Measures::CalculateAngles (float * *roll,* float * *pitch,* float * *heading*)

funkcja oblicza orientację urządzenia na podstawie przechowywanych pomiarów

Parametry

out	*roll*	obrót wokół osi x
out	*pitch*	obrót wokół osi y
out	*heading*	obrót igły magnetycznej

3.6.3.2 void Measures::ReturnData (float * *data*)

akcesor danych pomiarowych

Parametry

out	*data*	wskaźnik na przechowywane dane pomiarowe czujników

3.6.3.3 void Measures::UpdateData ( ) [slot]

slot umożliwiający aktualizację danych pomiarowych

Dokumentacja dla tej klasy została wygenerowana z plików:

- measures.h
- measures.cpp

3.7 Dokumentacja klasy SerialPort

klasa portu szeregowego

`#include <serialport.h>`

Metody publiczne

- SerialPort (const char *device, int baudrate=9600, const char *mode="8N1")
  konstruktor konfigurujący port szeregowy
- int Connect (const char *device, int baudrate, const char *mode)
  funkcja umożliwia połączenie do portu szeregowego
- void Disconnect ()
  funkcja odłącza program od portu szeregowego
- int GetArray (unsigned char *buffer, int len)
  funkcja pozwala na odbieranie danych przesyłanych poprzez port szeregowy

3.7.1 Opis szczegółowy

klasa portu szeregowego

klasa pozwala na połączenie się z portem szeregowym oraz odczytywanie i wysyłanie danych

3.7.2 Dokumentacja konstruktora i destruktora

3.7.2.1 SerialPort::SerialPort (const char * *device*, int *baudrate* = 9600, const char * *mode* = "8N1")

konstruktor konfigurujący port szeregowy

Parametry

in	*device*	nazwa portu szeregowego
in	*baudrate*	prędkość transmisji danych
in	*mode*	parametry konfiguracyjne portu

3.7.3 Dokumentacja funkcji składowych

3.7.3.1 int SerialPort::Connect (const char * *device*, int *baudrate*, const char * *mode*)

funkcja umożliwia połączenie do portu szeregowego

Parametry

in	*device*	nazwa portu szeregowego
in	*baudrate*	prędkość transmisji danych
in	*mode*	parametry konfiguracyjne portu

3.7.3.2 int SerialPort::GetArray (unsigned char * *buffer*, int *len*)

funkcja pozwala na odbieranie danych przesyłanych poprzez port szeregowy

Parametry

out	*buffer*	dane odbierane przez port
in	*len*	ilość bajtów które chcemy odebrać

Zwraca

ilość odebranych bajtów, -1 jeśli odbiór danych się nie powiódł

Dokumentacja dla tej klasy została wygenerowana z plików:

- serialport.h
- serialport.cpp