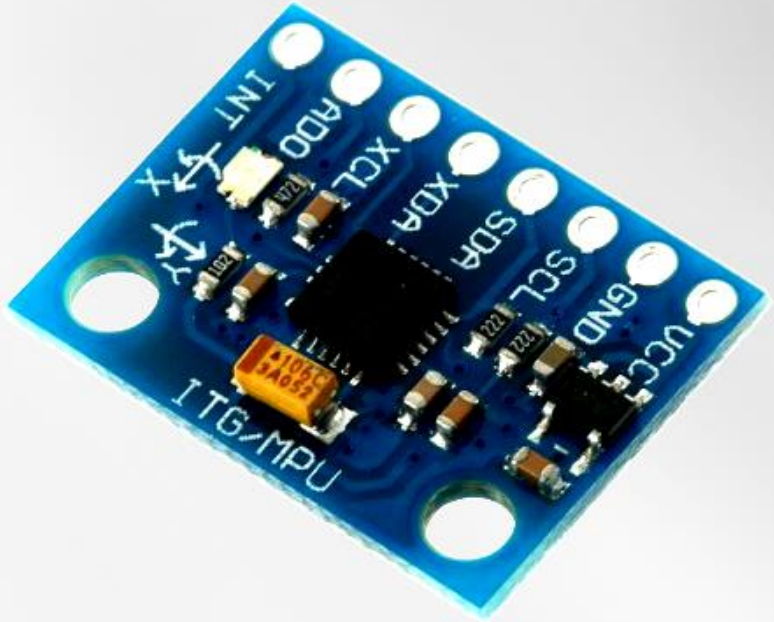
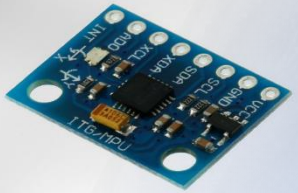
Kommunikation

Telemetrie

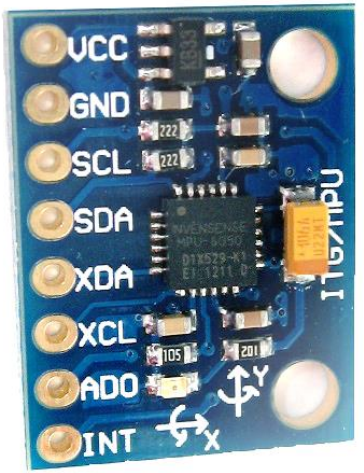


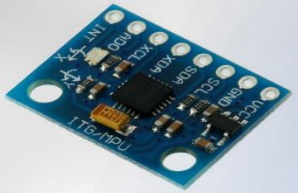


Inhalt

Umfang: ca. 2-4 Zeitstunden

- Klassenbibliothek Qt
- Telemetrie Framework Quatplay
- Quatplay Benutzeroberfläche
- Telemetrie Implementierung
  - Telemetrie senden
  - Telemetrie emfangen
- Aufgaben

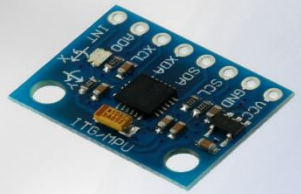




Klassenbibliothek Qt

Qt:

- C++ Klassenbibliothek
- GUI Programmierung
- Qt Creator -> Drag & Drop (einfach, ergonomisch)
- GPL Lizenz, Open Source, frei verfügbar
- Plattformen: Windows, Unix, Max, Android, …
- Erweitert C++ Standard um MOC (meta object compiler):
  - Mehr Funktionen: Signal & Slots
  - Voll kompatibel zu C++ Compiler



Klassenbibliothek Qt

Verwendung Signal & Slots:
1.) Signal und Slots sind Methoden
2.) Bei der Methoden-Deklaration in der Klassendefinition werden Signal und Slots definiert.
3.) Ein Signal mit Slot ist zu verbinden, dann wird jedes Mal der Slot aufgerufen, wenn das Signal aufgerufen wird. Das geht via connect:
Connect(Sender, Signal, Empfänger, Slot)

- Sender: Sendendes Objekt
- Signal: Methode, die bei Aufruf eine andere Methode anstößt
- Empfänger: Empfangenes Objekt
- Slot: Angestoßene Methode

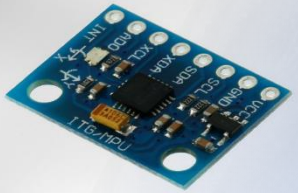
```
signals:
    void quaternion_received();
    void data_received(char *buff, int anzahl_Bytes);

public slots:

private slots:
        void receiveMsg();
```

4.) Im Beispiel ruft die Signal-Funktion quaternion_received() des Objekts QSerialPortobj->conn_usart die Slot-Funktion QUAT_Update() des Objekts (this) auf, in der dieser Code ausgeführt wird.

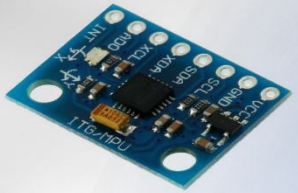
```
connect(QSerialPortobj->conn_uart,SIGNAL(quaternion_received()),this,SLOT(QUAT_Update()));
```



Telemetrie Framework Quatplay

Quatplay:

- Telemetrie Framework
- Qt basierend
- Enthält 5 fertige Module:
  - Glwidget -> 3D Anzeige (Flugzeug)
  - Serial Driver -> Serielle Kommunikation und Parser
  - myGraph -> Realisiert einen Graphen
  - mycurve -> Realisiert „Kurven“
    Pro Graph können drei „Kurven“ generiert werden
  - Quaternion -> Quaternionen
- Wie QwtPlot hinzufügen? Copy + Paste



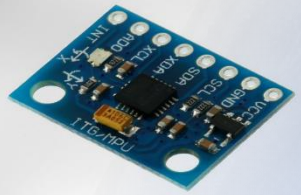
Telemetrie Framework Quatplay

myGraph:

- QwtPlot (wohin malen) wird dem Konstruktor übergeben
- add_value(x, y, ID-Kurve) -> fügt neues Datum hinzu
- clear_all -> löscht alle Daten
- update_graph() -> aktualisiert Graphen

myCurve:

- Ringpuffer, Werte werden intern doppelt gespeichert zum Effizienten simultanen lesen + schreiben
- update_graph (default) -> aktualisiert Graphen: Default-Werte möglich
- add_value(x,y) -> fügt Wert hinzu
- Clear_all() -> löscht Daten



Telemetrie Framework Quatplay

1. **Projekt *Quatplay_Framework_Qt* öffnen:**

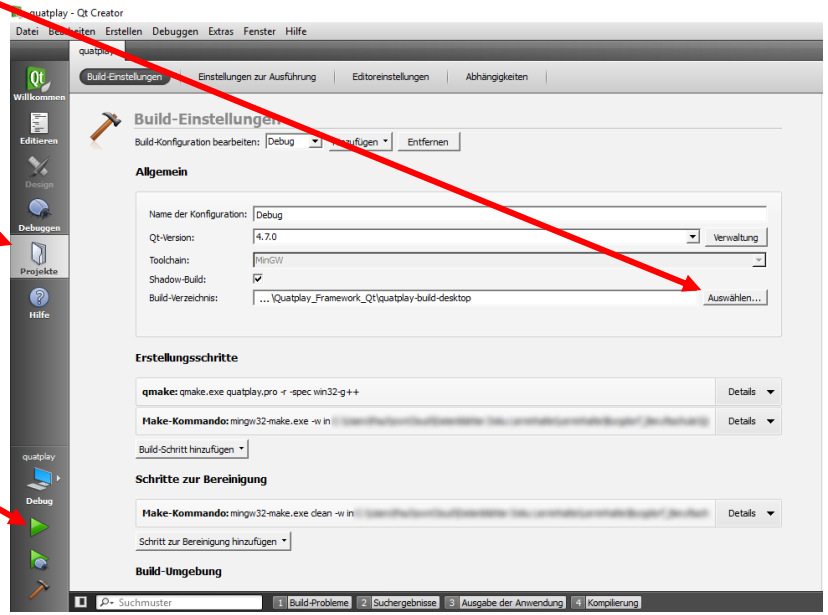
   Datei -> Datei oder Projekt öffnen -> QCSF_Framework -> quatplay -> quatplay.pro

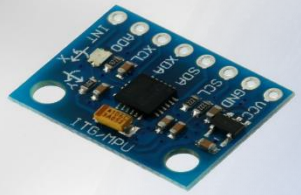
2. **Build-Pfad festlegen:**

Projekte -> Build Verzeichnis -> Auswählen -> QCSF_Framework -> quatplay-build-desktop

3. **Ausführen**

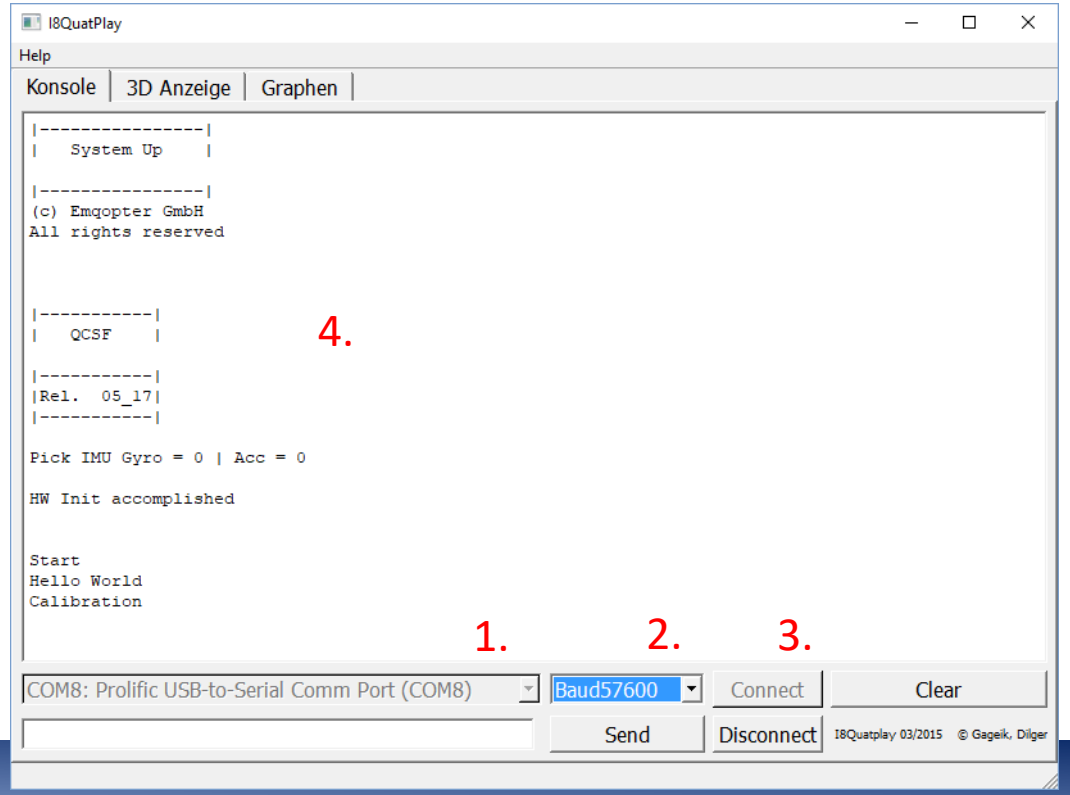
WICHTIG:
Der Pfad darf keine
Leerzeichen enthalten!

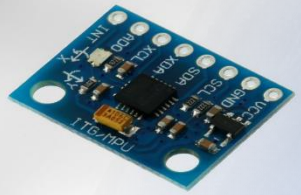




Telemetrie Framework Quatplay

1. COM – Port auf entsprechenden Port des USB – RS232 Converter einstellen
2. Baudrate auf *Baud57600* einstellen
3. Mit Klick auf *Connect* Verbindung aufbauen
4. Nachrichten empfangen

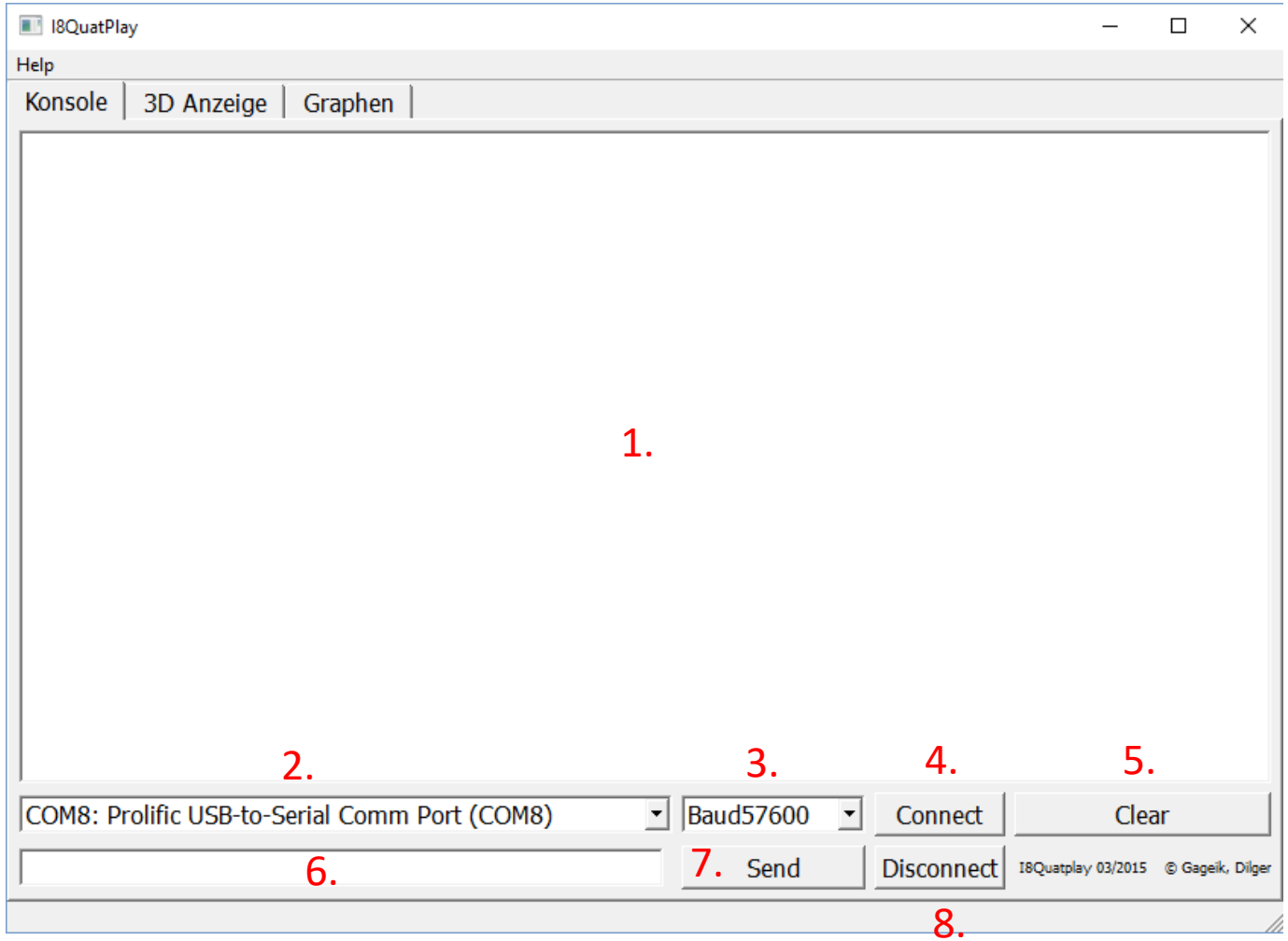


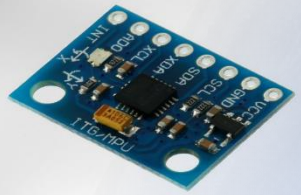


Quatplay - Benutzeroberfläche

Startansicht (Reiter *Konsole*)

1. Konsole
2. Com-Port-Auswahl
3. Baudraten-Auswahl
4. Verbindungsaufbau
5. Konsole löschen
6. Textfeld für Telekommandos
7. Telekommandos senden
8. Verbindung trennen

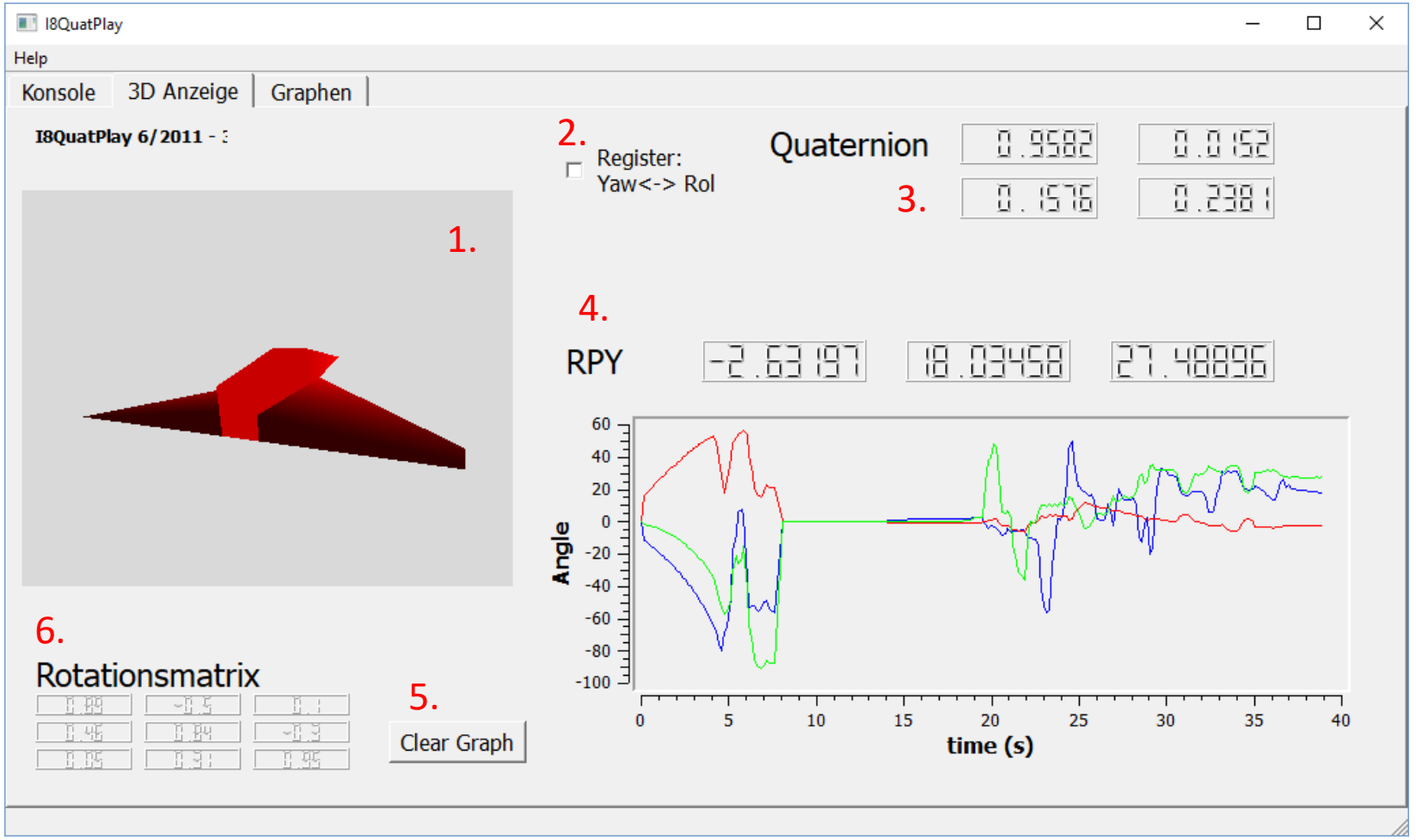


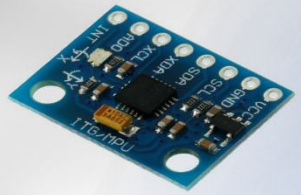


Quatplay - Benutzeroberfläche

3D Anzeige

1. Fluglagenanzeige
2. Vertauscht Yaw und Roll
3. Quaternionen
4. Roll, Pitch & Yaw
5. Graphen löschen
6. Rotationsmatrix

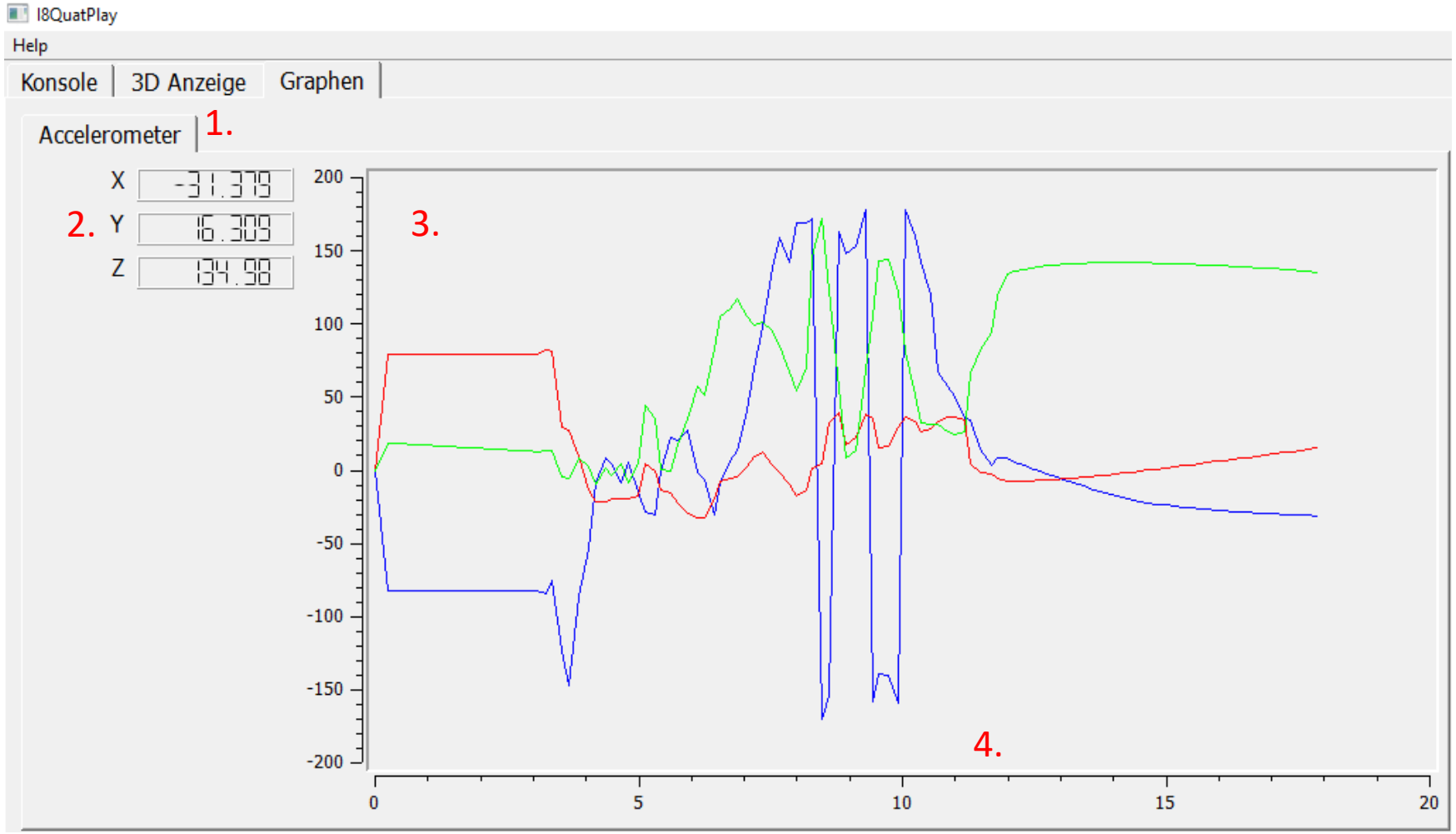


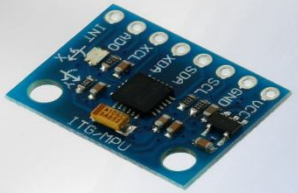


Quatplay - Benutzeroberfläche

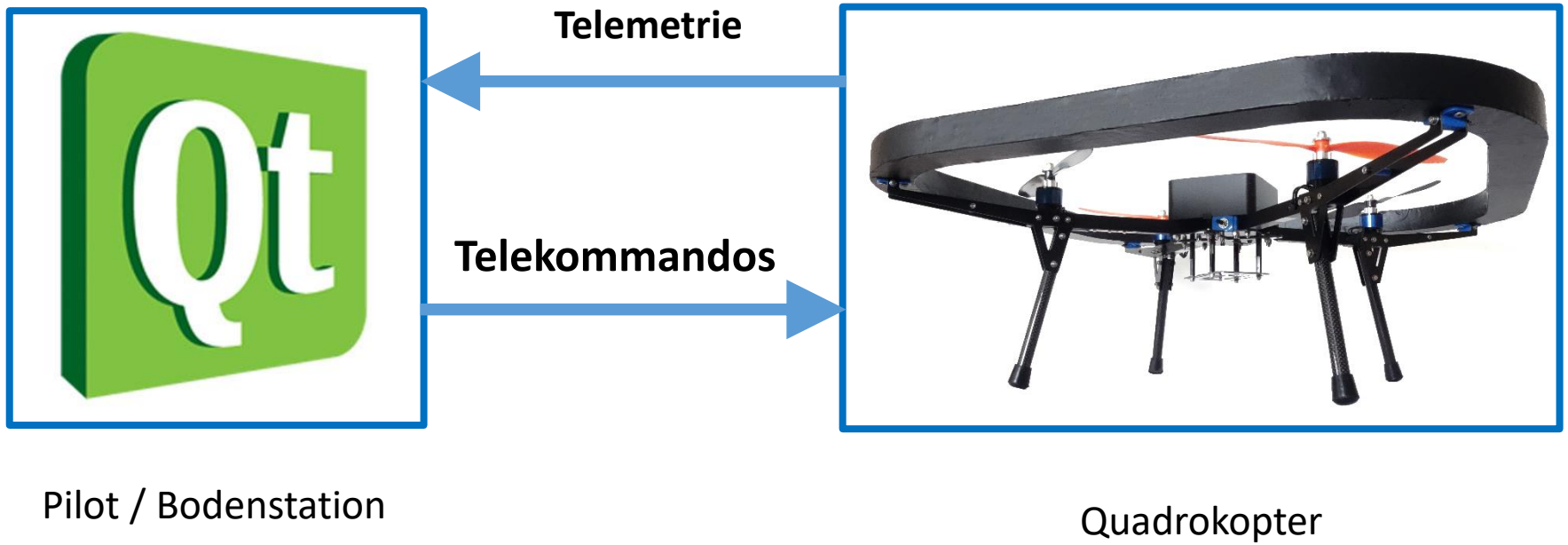
Graphen

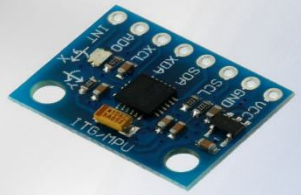
1. Reiter für versch. Graphen
2. Zahlenanzeige für 3 Werte
3. Graphanzeige für 3 Werte
4. Zeitachse in Sekunden





Telemetrie Implementierung





Telemetrie Implementierung

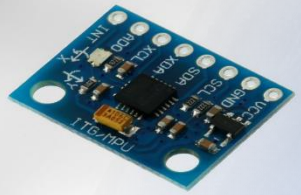
In der AVR – Software werden Datenpakete („Frames“) angelegt und versendet, die auf der PC-Seite mit Qt ausgelesen und dargestellt werden können.

Allgemeine Vorgehensweise:

1. Daten auf AVR – Seite zu Frames zusammenfügen
2. Frames entsprechend eines Protokolls zu einer Nachricht formulieren
3. Erstellte Nachricht über USART versenden
4. Nachricht auf Qt – Seite empfangen
5. Anhand des Protokolls die Nachricht interpretieren und Frames erstellen
6. Daten aus Frames darstellen

AVR

Qt



Telemetrie senden

1. Daten auf AVR Seite zu Frames zusammenfügen

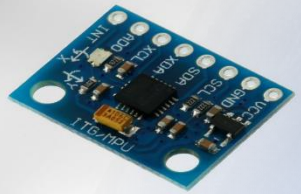
- *protocol.h* stellt *Frame – struct* bereit:

```
17 typedef struct {
18     char data[MAX_FRAME_LENGTH];
19     int data_pos;
20 } Frame;
```

- Frame initialisieren mit `void` **`startFrame`**`(`**`Frame*`**` f)` in *telemetry.c*

```
22         // Legt den Telemetrie Frame an.
23         Frame f; // USART Frame entsprechend definiertem Protokoll
24         startFrame(&f);
```

- Daten können mit `void` **`addValuesToFrame(Frame*`**` f, `**`float*`**` values, `**`int`**` numOfValues)` *telemetry.c* zu bestehendem Frame hinzugefügt werden.



Telemetrie senden

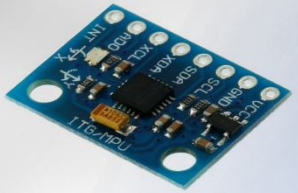
2. Frames entsprechend eines Protokolls zu einer Nachricht formulieren

- `void` **`addValuesToFrame(Frame*`** `f,` **`float*`** `values,` **`int`** `numOfValues)`

implementiert bereits die nötige Vorgehensweise, um die Daten entsprechend eines festgelegten Protokolls zu einer Nachricht zusammen zu fügen.

3. Erstellte Nachricht über USART versenden

- Mit `void` **`endAndSendFrame(Frame*`** `f)` in *telemetry.c* wird der Frame abgeschlossen und über USART verschickt.



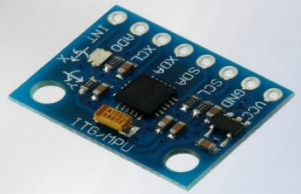
Telemetrie senden

Die gesamte Vorgehensweise wird in `void` **`mySendMessage`**`(`**`int`** `messageType)` in *telemetry.c* programmiert. Die Funktionalität ist dazu für jeden Frametyp entsprechend zu erweitern.

`int` `messageType` kennzeichnet den Frametyp – je nachdem, welche Daten verschickt werden sollen (Quaternionen, Gyrometerdaten, Accelerometerdaten,…). Diese Daten werden in der Datei *protocol.h* per define angelegt:

```
39// Telemetrie: IDs
40#define      TM_QUATERNION        0
41#define      TM_RPY               1
42#define      TM_MESSAGES          2
```

Hier wird für jeden Frametyp entsprechend ein Eintrag angelegt.



Telemetrie senden

Für jeden `int messageType` werden in `void mySendMessage(int messageType)` die Daten zum Frame hinzugefügt.

Das Hinzufügen der Daten wird in den jeweiligen Funktionen ausimplementiert.

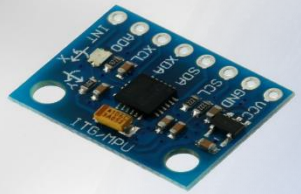
```
switch (messageType) {
default:
    break;

case QUATERNIONS:
    add_Quaternion_Values(&f);
    break;

case GYROMETER:
    my_add_Gyrometer_Frame(&f);
    break;

}
```

```
void add_Quaternion_Values(Frame* f) {
    float values[16]; // Array of values to be sent.
    values[0] = (float) imu.q0;
    values[1] = (float) imu.q1;
    values[2] = (float) imu.q2;
    values[3] = (float) imu.q3;
    addValuesToFrame(f, values, 4); // Add 4 values to Frame frame.
}
```



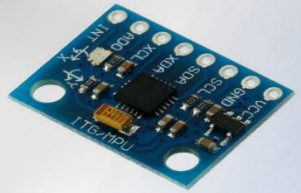
Telemetrie senden

`void mySendMessage(Frame* f, int messageType):`
Hier wird ein Frame für die Übertragung der Daten von messageType mit Daten gefüllt.

`void startFrame(Frame* f):`
Übernimmt das Anlegen eines neuen Frames und setzt die Anfangszeichen für den Frame entsprechend Protokoll.

`void addValuesToFrame(Frame* f, float* values, int numOfValues):`
Fügt dem Frame f entsprechend dem Protokoll #numOfValues Daten aus values hinzu.

`void endAndSendFrame(Frame* f):`
Schließt den Frame f entsprechend dem Protokoll ab und verschickt ihn.



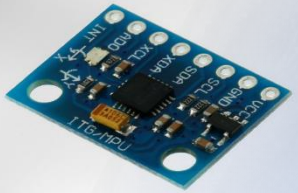
Telemetrie senden

Bedingung für erfolgreiche Kommunikation:

Sender einer Nachricht muss eine Sprache sprechen, die der Empfänger versteht.

- Einführung des Kommunikationsprotokolls in Datei *protocol.h*
- Selbe Datei in AVR – Software und Qt – Software!
- **Enthält:**
  - Start-, Trenn- und Endzeichen
  - Telekommando- und Telemetrie-Types

-> Festlegung der Sprache für Telemetrie und Telekommandos
-> Änderungen im Protokoll müssen in beiden Softwaren berücksichtigt werden!

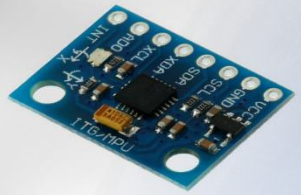


Telemetrie empfangen

4. Nachricht auf Qt – Seite empfangen

Grundsätzliche Funktionsweise (*Signals* und *Slots*):

Module in Qt stellen *Slots* (Funktionen) bereit, die aufgerufen werden, wenn bestimmte *Signals* (Ereignisse) getriggert werden – vorausgesetzt, Signals und Slots sind verbunden.



Telemetrie empfangen

4. Nachricht auf Qt – Seite empfangen

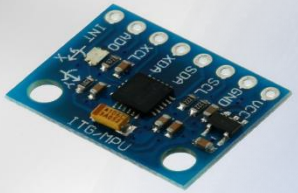
Slot *receiveMsgFrames()* in *qserialconnection_uart.h* zum Empfangen von Nachrichten

```
44    private slots:
45            void receiveMsgFrames();
```

wird in *qserialconnection_uart.cpp* durch die Funktion `bool` **`connect`**`(…)`
mit einem Signal *timeout()* des Objektes *timer_usart* verbunden:

```
12        // Frage alle 50ms nach neuen Daten am USART
13        timer_usart.setInterval(50);
14        connect(&timer_usart,SIGNAL(timeout()),this,SLOT(receiveMsgFrames()));
```

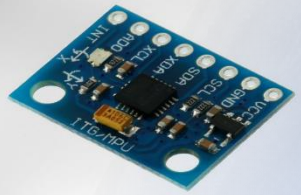
-> Sobald *timeout()* getriggert wird, wird die Funktion *receiveMsgFrames()* ausgeführt.



Telemetrie empfangen

4. Nachricht auf Qt – Seite empfangen

- *receiveMsgFrames()* prüft, ob neue Nachrichten am USART angekommen sind
- Falls ja, wird *processUARTFrames()* in *QSerial.cpp* aufgerufen, wo die Nachricht verarbeitet wird.



Telemetrie empfangen

5. Anhand des Protokolls die Nachricht interpretieren und Frames erstellen

```
8   extern Quaternion Quat;
9   extern PacketData receivedPacket_Quat;          1.
10  extern PacketData receivedPacket_RPY;

119     if(frameStart != -1) {
120         msg = msg.right(frameStart - 1);
121         while(msg.length() > 1) {                2.
122             QString data = entnehmePaket();      3.
123
124             if (data != FEHLERSTRING){
125
126                 PacketData receivedPacket = this->parsePacket(data);
127
128              4. if(receivedPacket.typ == TM_QUATERNION) {
129                     receivedPacket_Quat  = receivedPacket,
130                 6. frameReceived_Quat();
131              5. } else if (receivedPacket.typ == TM_RPY) {
132                     receivedPacket_RPY = receivedPacket;
133                 6. frameReceived_RPY();
134                 }
135             }
136         }
137         msg = "";
138     }else{
139         // Framestart nicht gefunden -> Message über Console ausgeben.
140         console_output(msg);
141         msg = "";
142     }
```

1. *receivedFrame_Quat,receivedFrame_RPY* und Quat sind globale Variablen aus mainwindow.cpp, um die Daten bis zur Darstellung im Graphen zwischen zu speichern.

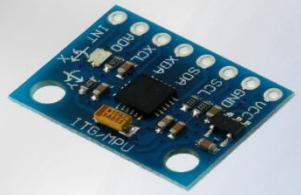
2. Solange noch unverarbeitete Nachrichten vorhanden sind, werden Pakete gesucht und gemäß Protokoll extrahiert

3. Es wird immer ein Packet entnommen und verarbeitet, bevor das nächste entnommen wird

4. Mit *PacketData.typ* erhält man den Pakettyp

5. Die Interpretation der Daten erfolgt abhängig vom Pakettyp.

6. *frameReceived_Quat()* und *frameReceived_RPY()* sind Signals, die das Ankommen eines neuen Frames des jeweiligen Typs signalisieren.



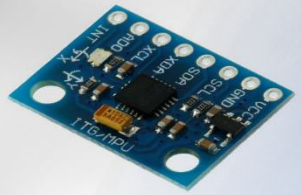
Telemetrie empfangen

5. Anhand des Protokolls die Nachricht interpretieren und Frames erstellen

Hinweise zu *PacketData* **in** *QSerial.h*:

- **Beinhaltet:**
  - *typ* vgl. TelemetryTypes in *protocol.h*
  - *values* Datenwerte
  - *value_count* *Anzahl der Datenwerte*

```
19  struct PacketData {
20      float typ;
21      float * values;
22      int value_count;
23  };
```



Telemetrie empfangen

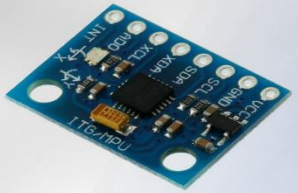
6. Daten aus Frames darstellen

- Framedaten sind für jeden Typ separat in *FrameData* – Strukturen in *mainwindow.cpp* abgelegt

```
34    // Framedaten
35    PacketData receivedPacket_Quat;    // Framedaten zu Quaternionen
36    PacketData receivedPacket_RPY;     // Framedaten zum Accelerometer
```

- Erinnerung: Strukturen wurden in *processUARTFrames()* in *QSerial.cpp* mit Daten gefüllt.
- Signals für das Ankommen neuer Framedaten in *QSerial.h* angelegt:

```
48        signals:
49        void frameReceived_Quat();   // Neue Quaternionendaten angekommen.
50        void frameReceived_RPY();    // Neue RPY - Daten angekommen.
```



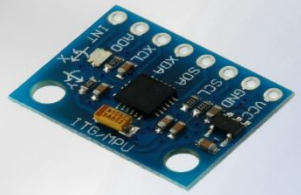
Telemetrie empfangen

6. Daten aus Frames darstellen

- In *mainwindow.cpp* werden die *Signals* für ankommende Daten mit den Funktionen zur Darstellung der Daten in Graphen verbunden:

```
87          // ************* FRAMES & GRAPHEN ******************
88
89          connect(QSerialPortobj,SIGNAL(frameReceived_Quat()),this,SLOT(frameUpdate_Quat()));
90          connect(QSerialPortobj,SIGNAL(frameReceived_RPY()),this,SLOT(frameUpdate_RPY()));
```

→ Kommen neue Quaternionendaten an, wird *frameUpdate_Quat()* in *mainwindow.cpp* aufgerufen.
→ Kommen neue RPY Daten an, wird *frameUpdate_RPY()* in *mainwindow.cpp* aufgerufen.



Telemetrie empfangen

6. Daten aus Frames darstellen

- In der Funktion *frameUpdate_RPY()* in *mainwindow.cpp* werden die in *receivedFrame_RPY* zwischen gespeicherten RPY – Daten an den Graphen rpy_graph und die LCD Anzeigen übertragen.

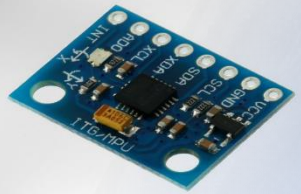
```
200   // Stellt die ankommenden RPY - Daten dar
201   void MainWindow::frameUpdate_RPY() {
202         // Daten in Graph schreiben
203         rpy_graph->add_value_on_time(receivedPacket_RPY.values[0], 1);
204         rpy_graph->add_value_on_time(receivedPacket_RPY.values[1], 2);
205         rpy_graph->add_value_on_time(receivedPacket_RPY.values[2], 3);
206         // Daten in LCDs anzeigen
207         ui->lcdNumber_RPY_X->display(receivedPacket_RPY.values[0]);
208         ui->lcdNumber_RPY_Y->display(receivedPacket_RPY.values[1]);
209         ui->lcdNumber_RPY_Z->display(receivedPacket_RPY.values[2]);
```

Initialisierung der Graphen
in *mainwindow.cpp:*

```
38    // Graphen
39    myGraph* rpy_graph;
40    myGraph* quat_graph;

92        rpy_graph = new myGraph(ui->qwtPlot_Graph_RPY);
93        quat_graph = new myGraph(ui->qwtPlot_Quat);
```

myGraph bekommt beim Anlegen das UI – Objekt mit übergeben!

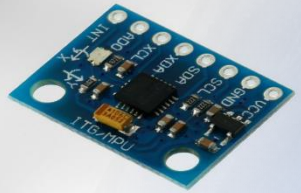


Telemetrie empfangen

6. Daten aus Frames darstellen

- **myGraph - Modul:**
  - Enthält bis zu 3 unabhängige Kurven
  - Relevante Funktionen:

`myGraph(QwtPlot *p)`	-> Konstruktor mit ***QwtPlot*** *p als Zeichenebene
`void add_value_on_time(y,Kurven_ID)`	-> fügt neues Datenpaar zur Kurve mit Kurven_ID hinzu
`void clear_all()`	-> löscht alle Daten des Graphen



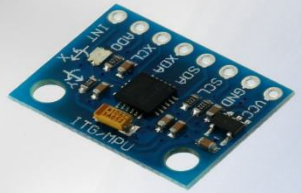
Aufgaben

Notwendige Hardware:

- EMQ3000
- Mini USB Kabel für Strom und zum Flashen
- Mini USB Kabel für Kommunikation: RS232 auf PC (RS232 oder USB)
- QCS / QCSF

Notwendige Software:

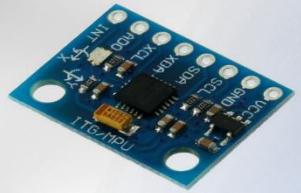
- AVR Studio 32 installiert (mit Tool Chain und Flip Treiber)
- Qt SDK Version 4.8.1 oder 4.7.4.
- Quatplay Qt Framework (Code)



Aufgaben

Aufgabe 1:
Stellen Sie eine Kommunikationsverbindung mit dem Qt-Steuerprogramm her. Testen Sie die Übertragung mit den bereits implementierten Funktionen aus Kapitel 2.

Aufgabe 2:
Nutzen Sie die Funktion `void` **`my_send_telemetry`**`()` in der Datei *telemetry.c*, um die Quaternionen des QCS per USART an das Qt - Steuerprogramm zu verschicken. Das Verschicken von Nachrichtenpaketen erfolgt dazu über `void` **`mySendMessage(int`** `messageType`**`).`**
Verwenden Sie als Parameter *messageType* den Eintrag *TM_QUATERNION* entsprechend dem define aus in der Datei *protocol.h*. Wenn die Aufgabe richtig gelöst wurde, ist unter dem Tab *3D Anzeige* ein Flugzeugmodell zu sehen, das entsprechend den Bewegungen der IMU reagiert.



Aufgaben

Aufgabe 3:

Legen Sie in der AVR Software, analog zum Beispiel der Quaternionen, einen Frame an, der die RPY – Daten des QCS per Telemetrie an Qt sendet. Modifizieren sie dazu die Funktion `void mySendMessage(int messageType).` Verwenden und implementieren Sie dazu die Funktion `void my_add_RPY_Values(Frame * f)`. Die RPY – Daten erhalten Sie über das externe struct `imu_data imu.`

Sie sollten nun im Reiter *Graphen/RPY* die Lage des QCS zu den Achsen Roll, Pitch und Yaw in einem Diagramm dargestellt bekommen.