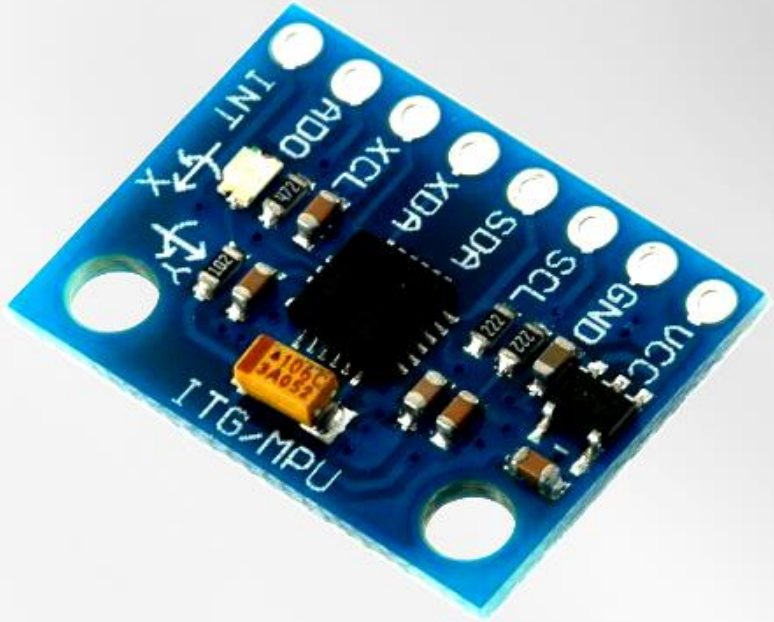
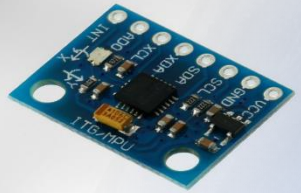
Kommunikation

Telemetrie und Telekommandos

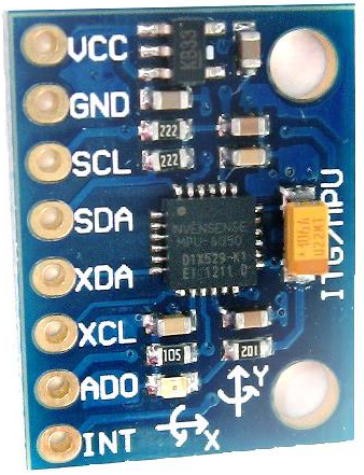


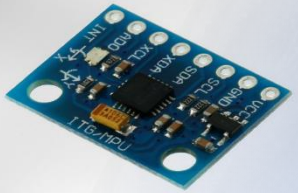


Inhalt

Umfang: ca. 1-2 Zeitstunden

- Klassenbibliothek Qt
- Quatplay Benutzeroberfläche und Verbindungsaufbau
- Telemetrie
- Telekommandierung
- Bearbeitung der Quatplay Benutzeroberfläche
- Aufgaben

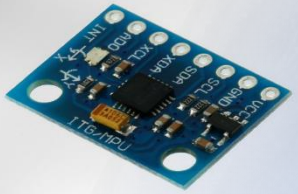




Klassenbibliothek Qt

Qt:

- C++ Klassenbibliothek
- GUI Programmierung
- Qt Creator -> Benutzeroberflächen mit Drag & Drop erstellen
- GPL Lizenz, Open Source, frei verfügbar



Qt – Projekt öffnen und starten

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

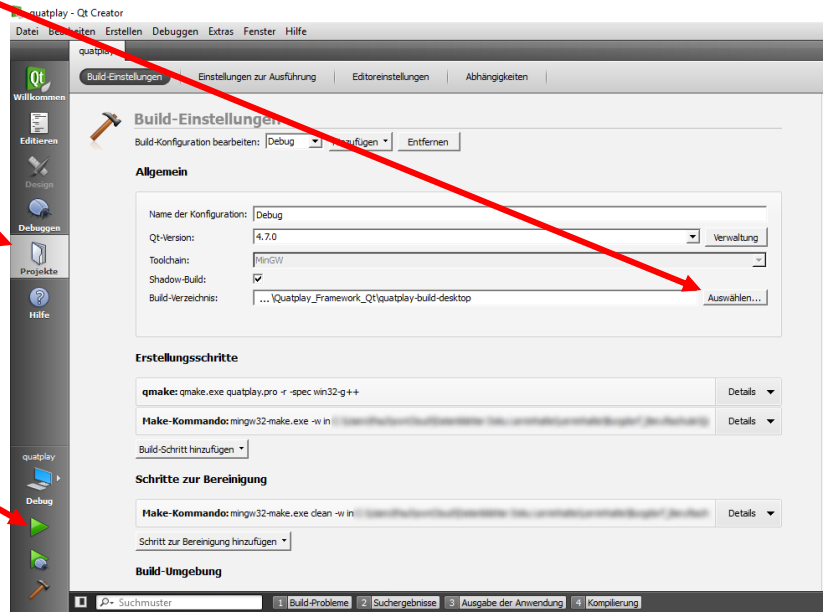
   Datei -> Datei oder Projekt öffnen -> Quatplay_Framework_Qt -> quatplay -> quatplay.pro

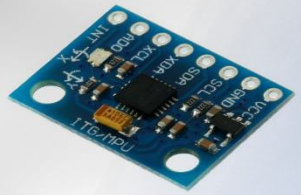
2. **Build-Pfad festlegen:**

Projekte -> Build Verzeichnis -> Auswählen -> Quatplay_Framework_Qt\quatplay-build-desktop

3. **Ausführen**

WICHTIG:
Der Pfad darf keine
Leerzeichen enthalten!

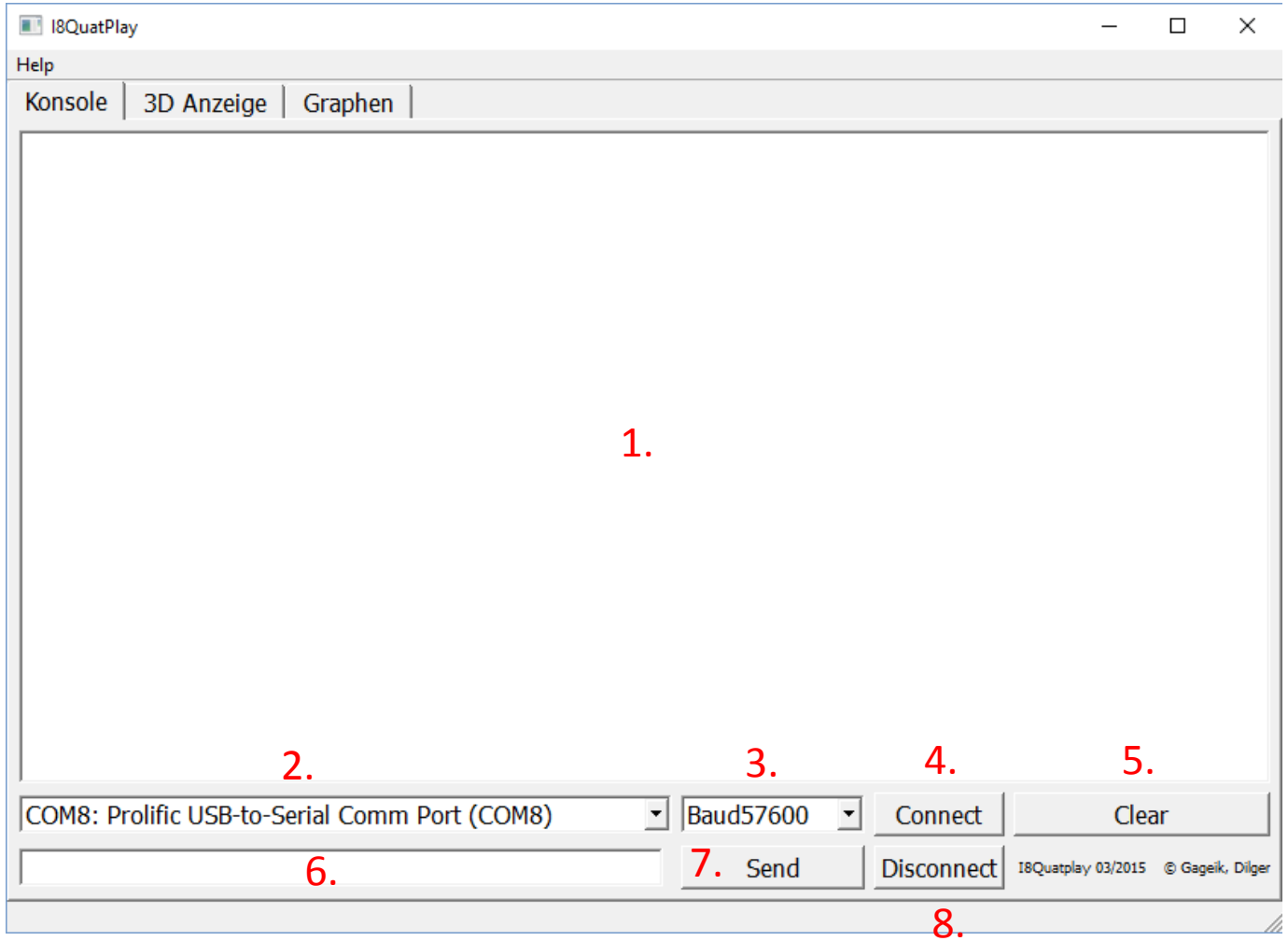


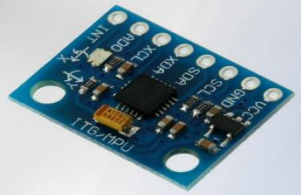


Quatplay - Benutzeroberfläche

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

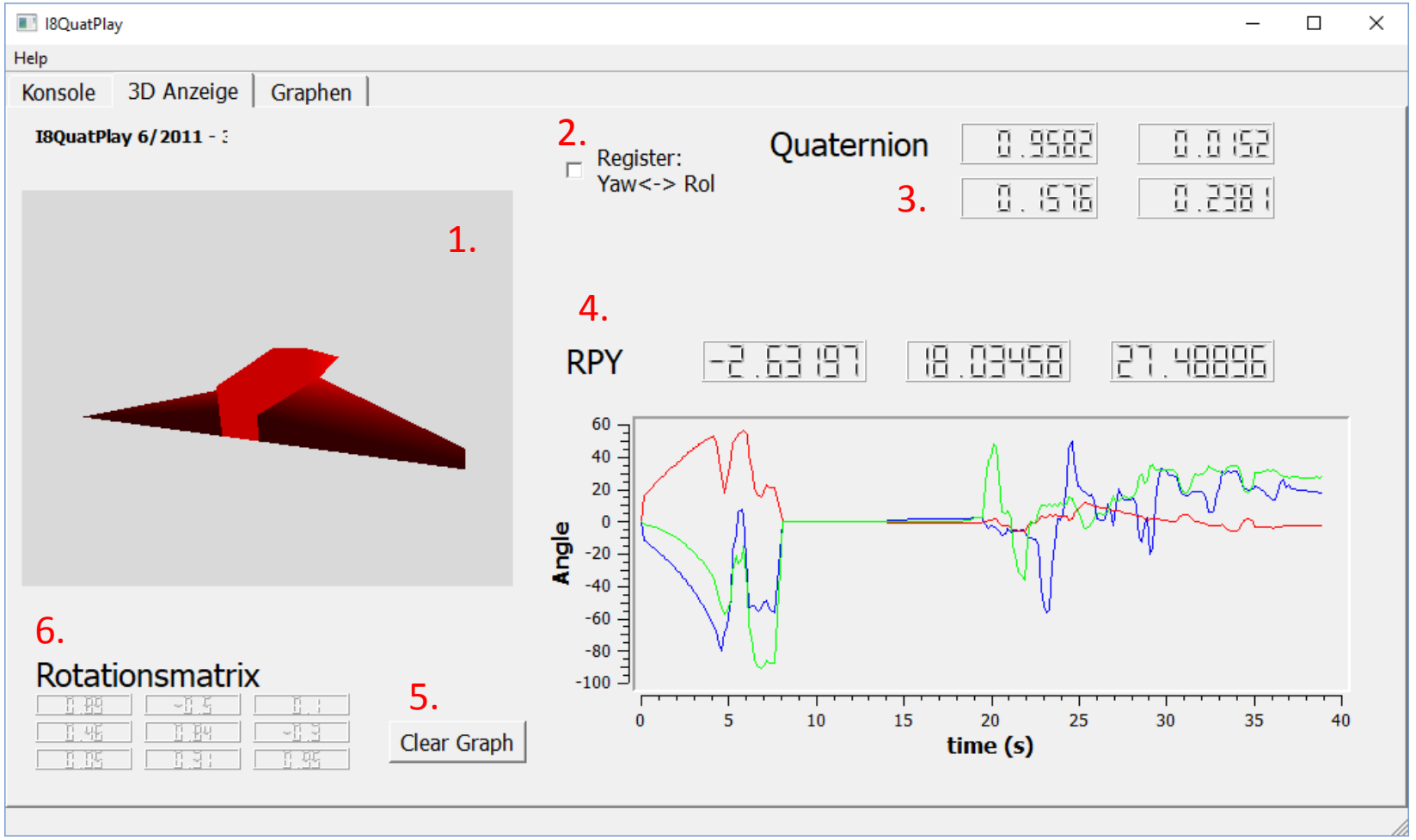


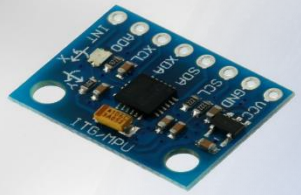


Quatplay - Benutzeroberfläche

3D Anzeige

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

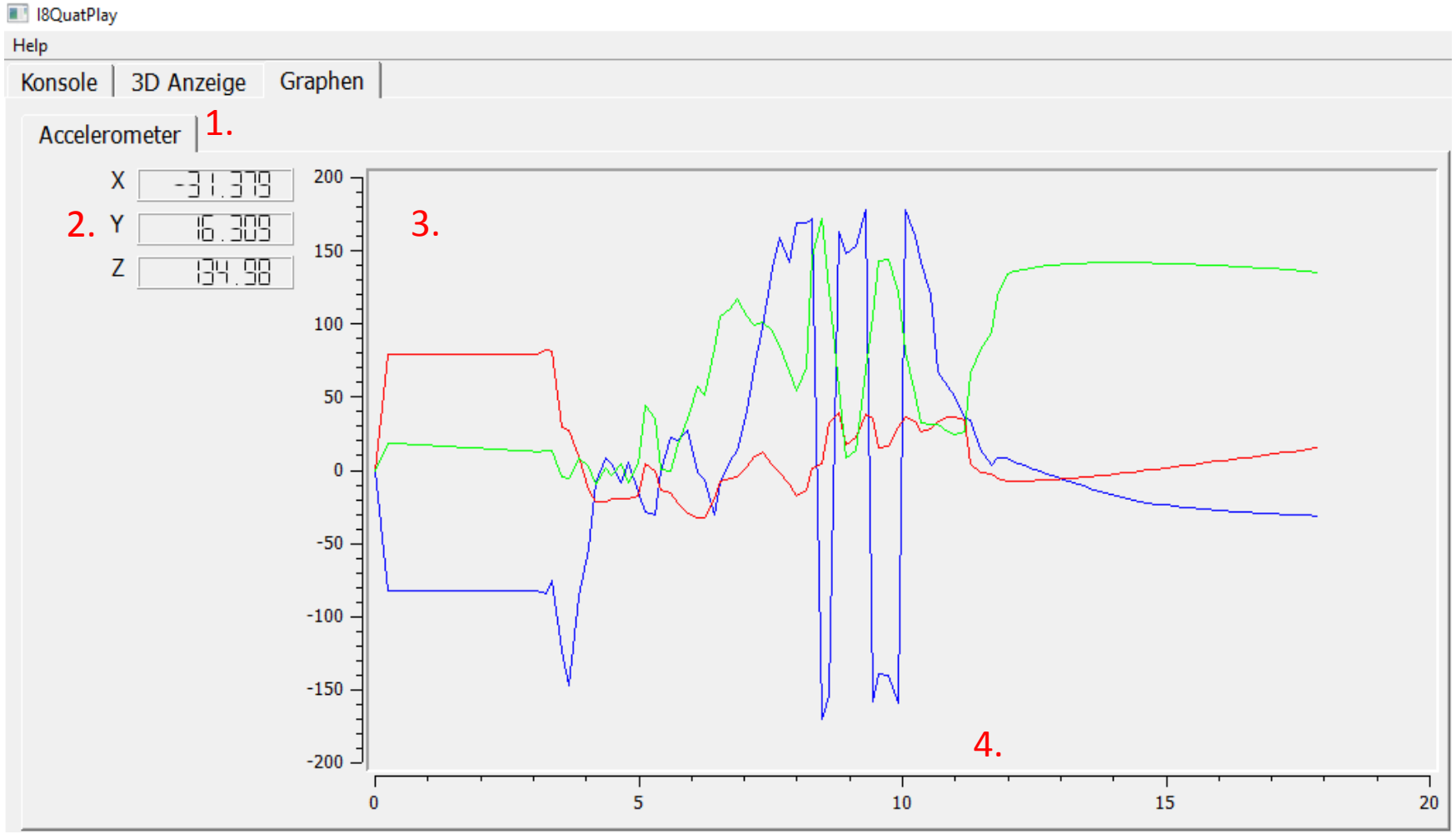


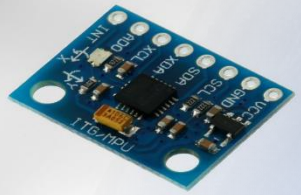


Quatplay - Benutzeroberfläche

Graphen

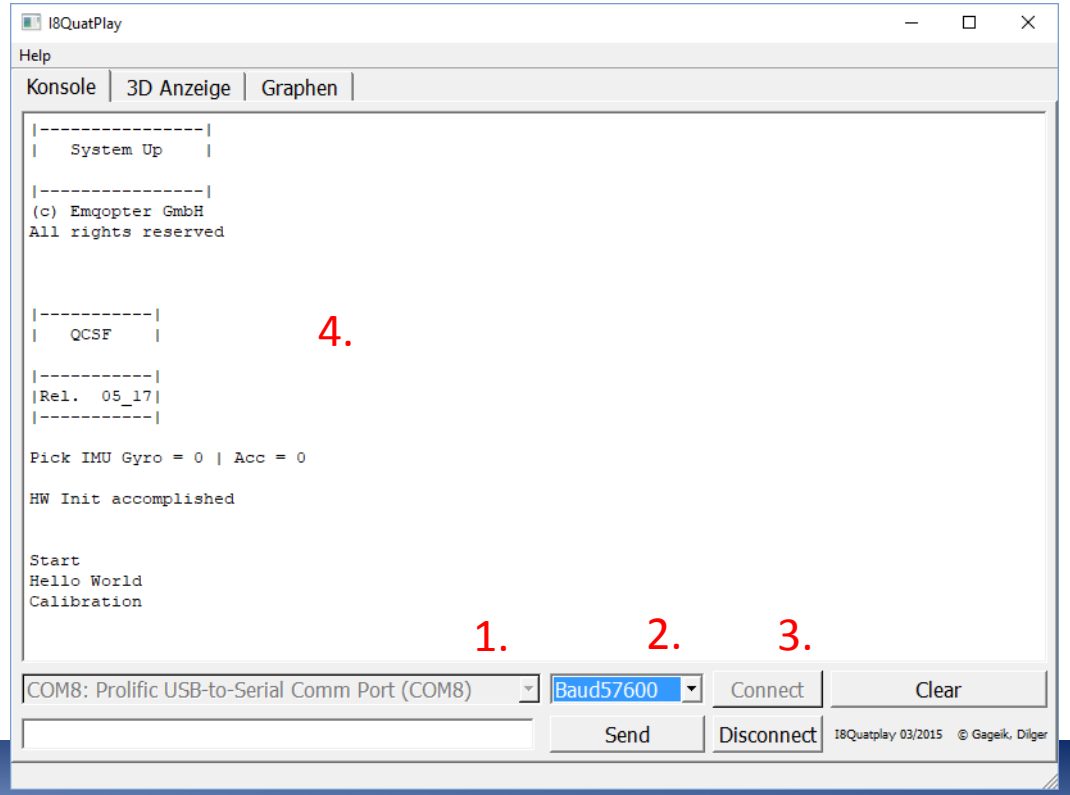
1. Reiter für versch. Graphen
2. Zahlenanzeige für 3 Werte
3. Graphanzeige für 3 Werte
4. Zeitachse in Sekunden

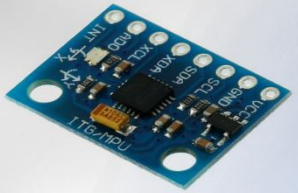




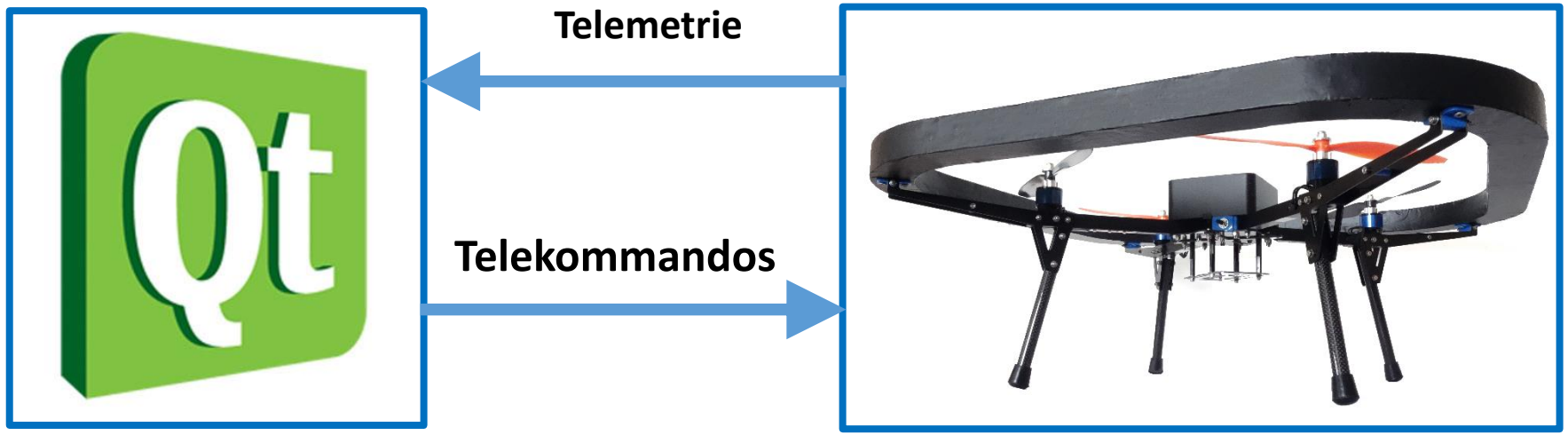
Verbindungsaufbau mit Quatplay

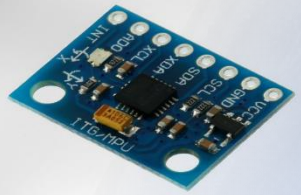
1. COM – Port auf entsprechenden Port des USB – RS232 Converter einstellen
2. Baudrate auf *Baud57600* einstellen
3. Mit Klick auf *Connect* Verbindung aufbauen
4. Nachrichten empfangen





Telemetrie und Telekommandos





Telemetrie (1)

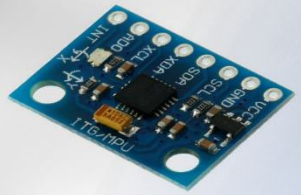
In der AVR – Software werden Datenpakete („Frames“) angelegt und versendet, die auf der PC-Seite mit Qt ausgelesen und dargestellt werden können.

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 (1)

1. Daten auf AVR Seite zu Frames zusammenfügen

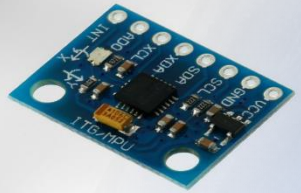
- *protocol.h* stellt *Frame – struct* bereit:

```
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 (1)

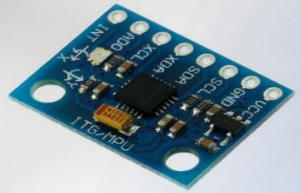
2. Frames entsprechend eines Protokolls zu einer Nachricht formulieren

- `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.



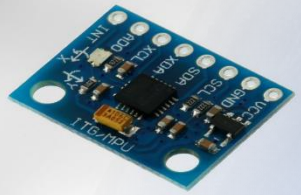
Telemetrie (1)

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 (1)

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.

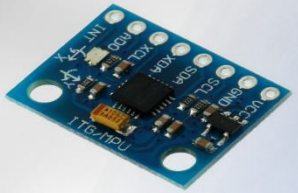
```
switch (messageType) {
default:
    break;

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.
}
```



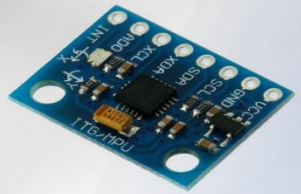
Funktionsübersicht telemetry.c

`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.



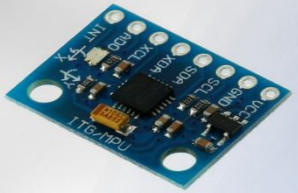
Das Kommunikationsprotokoll

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!

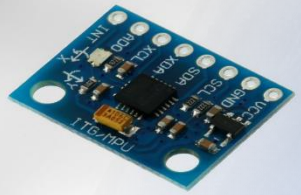


Telemetrie (2)

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 (2)

4. Nachricht auf Qt – Seite empfangen

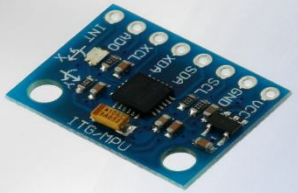
Slot *receiveMsgFrames()* in *qserialconnection_uart.h* zum Empfangen von Nachrichten

```
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()));
```

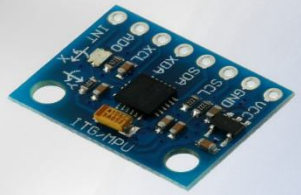
-> Sobald *timeout()* getriggert wird, wird die Funktion *receiveMsgFrames()* ausgeführt.



Telemetrie (2)

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 (2)

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.

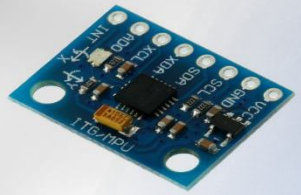
2. Solange noch unverarbeitete Nachrichten vorhanden sind, werden Pakete gesucht und gemäß Protokoll extrahiert

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.



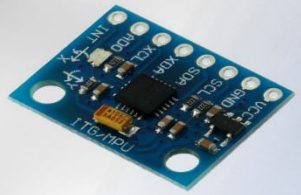
Telemetrie (2)

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 (2)

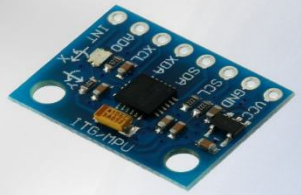
6. Daten aus Frames darstellen

- 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.
```



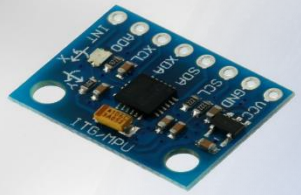
Telemetrie (2)

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 (2)

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.

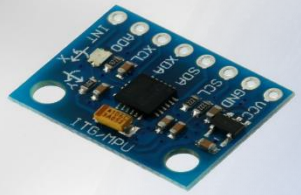
```
200   // Stellt die ankommenden RPY - Daten dar
201   void MainWindow::frameUpdate_RPY() {
202         // Daten in Graph schreiben
203         rpy_graph->add_value_on_time(receivedPacket_RPY.values[0], 1);
204         rpy_graph->add_value_on_time(receivedPacket_RPY.values[1], 2);
205         rpy_graph->add_value_on_time(receivedPacket_RPY.values[2], 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!

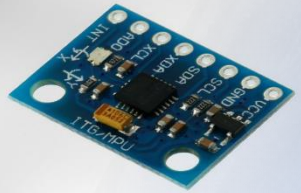


Telemetrie (2)

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

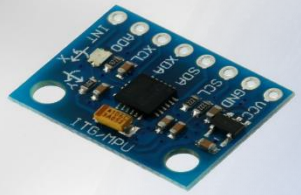


Telekommandierung

Das Qt – Programm wird verwendet, um den Quadrokopter zu steuern / kontrollieren

Allgemeine Vorgehensweise:

1. Daten entsprechend eines Protokolls zu einer Nachricht formulieren
2. Erstellte Nachricht über USART versenden
3. Nachricht auf AVR – Seite empfangen
4. Anhand des Protokolls die Nachricht interpretieren und Daten extrahieren
5. Daten verwenden, um Quadrokopter zu steuern / kontrollieren



Telekommandierung

1. Daten entsprechend eines Protokolls zu einer Nachricht formulieren + 2. Nachricht senden

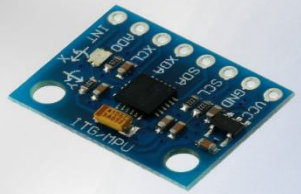
- `void` **`createFrameMessageAndSend`**`(`**`int`** `type, `**`int`** `data_length, `**`float`** `data[])`
  in ***mainwindow.c*** erstellt entsprechend dem Kommunikationsprotokoll einen Frame und sendet ihn ab.

`int` `type` — wird von der Enumeration *TelecommandTypes* aus ***protocol.h*** *entsprechend der gewünschten Nachricht ausgewählt:*

```
35  // Telecommandos: IDs
36  #define         TC_NOTHING                  -1
37  #define         TC_LED                      0
38  #define         TC_TRANS                    1
39  #define         TC_AUTOFLUG                 2
40  #define         TC_MOTOR                    3
41  #define         TC_IMU                      4
```

`int` `data_lenght` — Anzahl der Daten, die in diesem Frame mitgesendet werden.

`float` `data []` — Daten, die in diesem Frame mitgesendet werden.



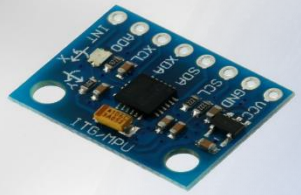
Telekommandierung

3. Nachricht auf AVR – Seite empfangen. + 4. Daten extrahieren

- `int` **`extract_type_and_data_from_telecommand_frame`**`()`
  - In ***telecommand.c***
  - ließt vom USART
  - extrahiert einen ankommenden Telekommando-Frame
  - schreibt die relevanten Daten in my_*data*
  - gibt den Frametyp des empfangenen Frames zurück.

5. Daten verwenden, um Quadrokopter zu steuern / kontrollieren

- `void` **`my_read_telekommand()`**
  - In ***telecommand.c***
  - wird alle TC_UPDATE_RATE Millisekunden aufgerufen
  - Auslesen und Verarbeiten der Telekommandierung.

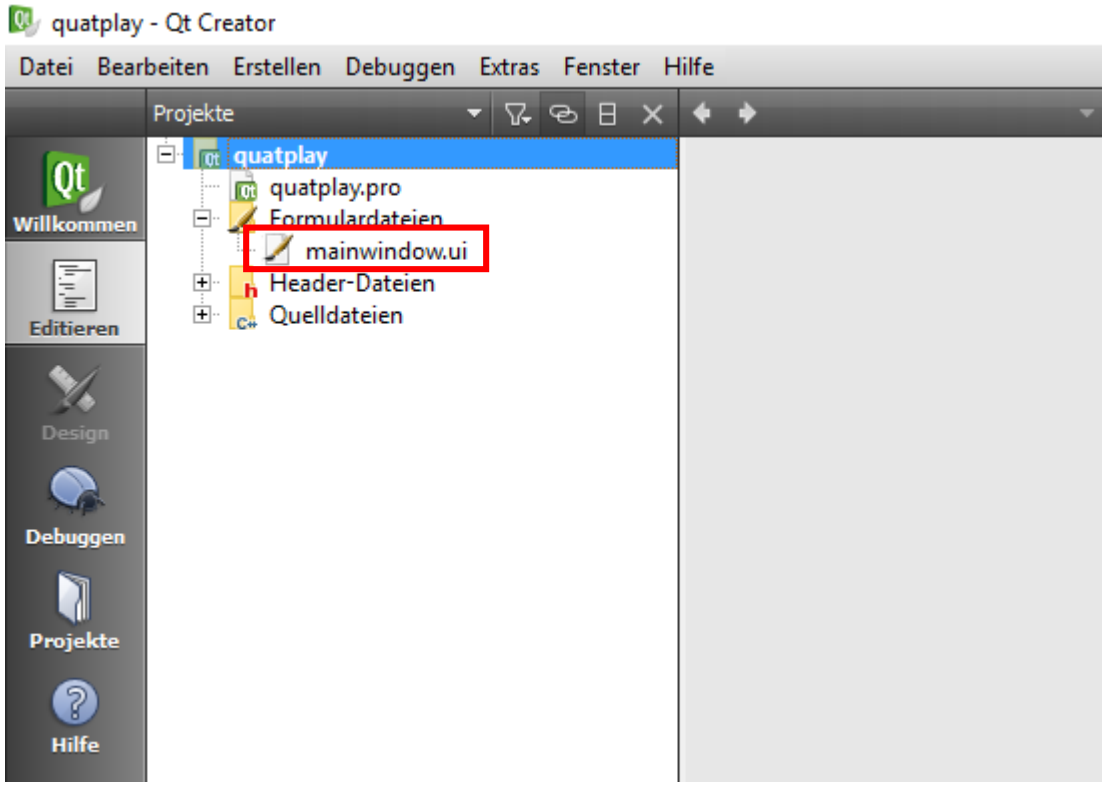


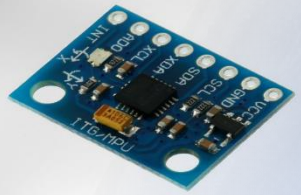
Bearbeitung der Benutzeroberfläche

Bedienelemente hinzufügen und bearbeiten

Doppelklick auf *Formulardateien/mainwindow.ui*

→ Öffnet die Bearbeitungsansicht der Benutzeroberfläche

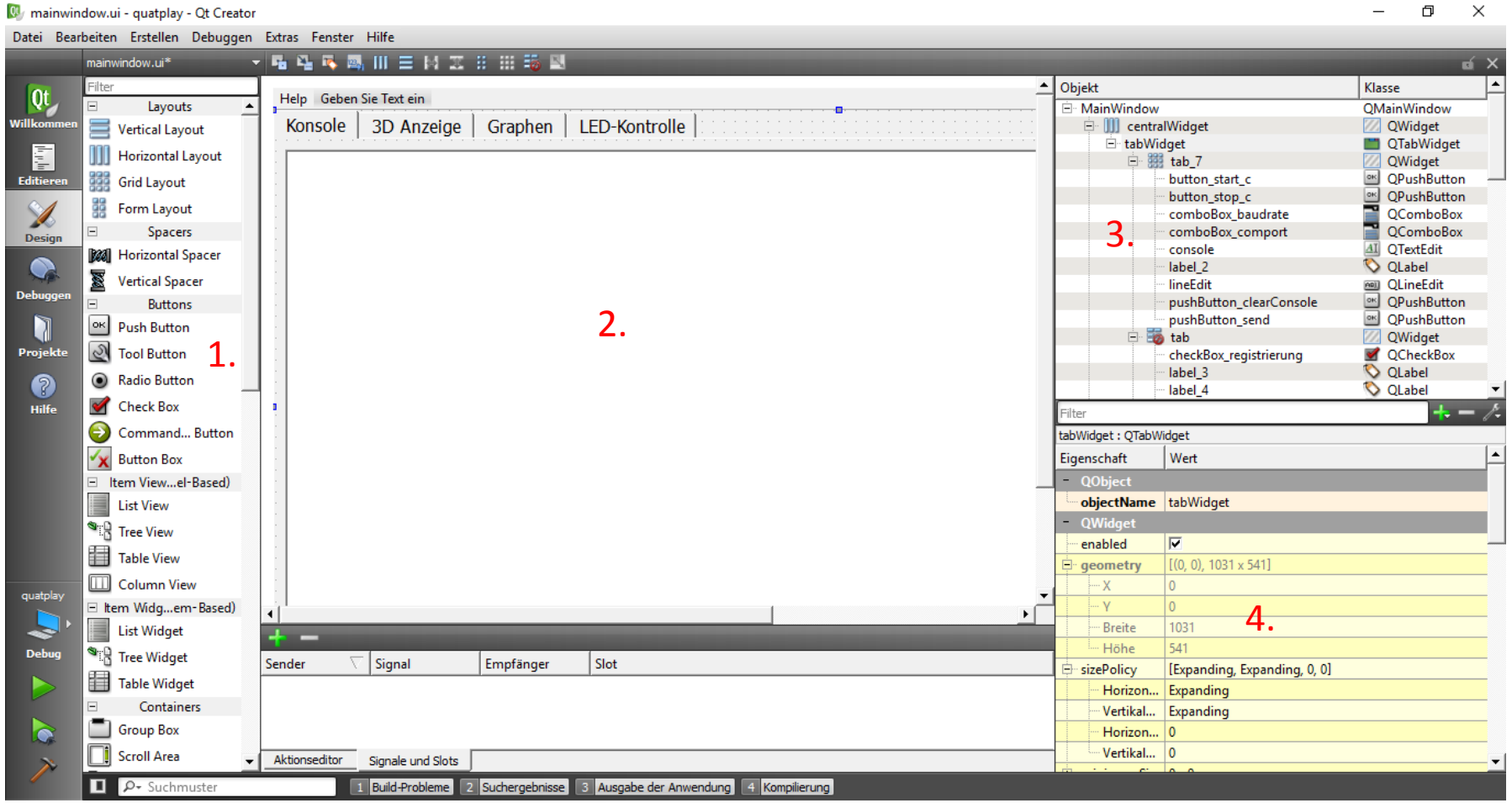


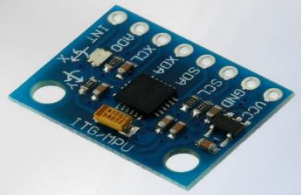


Bearbeitung der Benutzeroberfläche

Die Bearbeitungsansicht

1. Elementbibliothek
2. UI – Vorschau
3. UI – Elemente
4. Element - Details





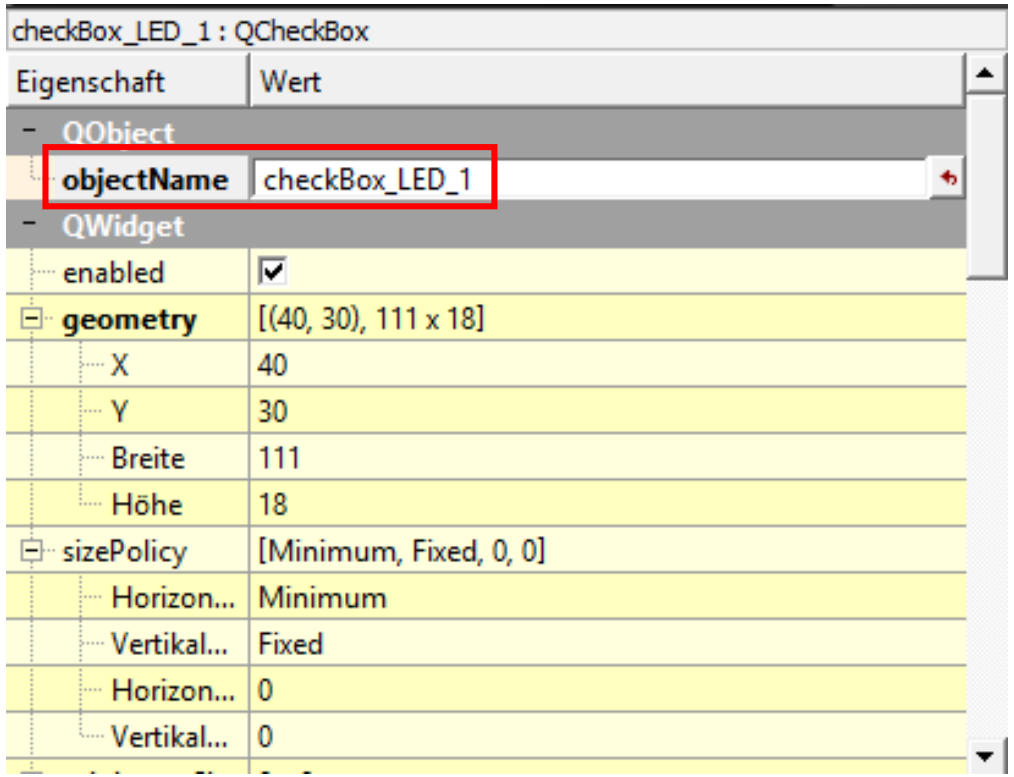
Bearbeitung der Benutzeroberfläche

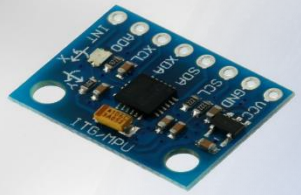
Element hinzufügen:

- Einfach über Drag und Drop aus Elementbibliothek einfügen
- **Wichtig:** Nach Einfügen gleich den Namen unter *Element – Details / objectName* aussagekräftig anpassen:
- Alle Elemente der Benutzeroberfläche können in *mainwindow.cpp* mit ui->objectName aufgerufen werden
- Die Funktionen von Methoden können über ui->objectName-> … aufgerufen werden

  Hier z.B.: `ui->checkBox_LED_1->isChecked()`

- Mit <strg> + <leer> nach dem ‚->‘ öffnet sich eine Liste möglicher Ergänzungen. So lassen sich gewünschte Funktionen schnell finden.



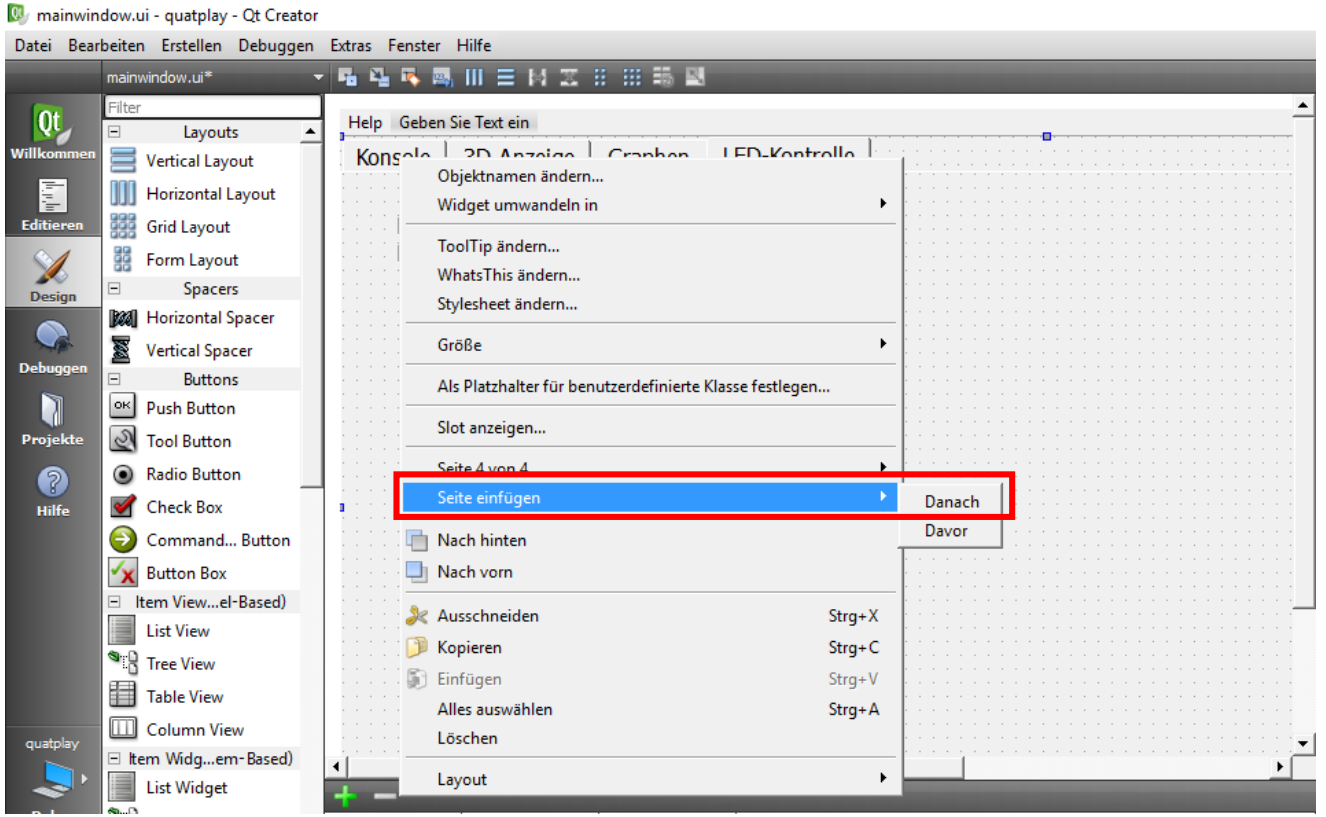


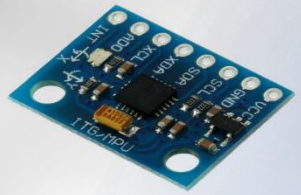
Bearbeitung der Benutzeroberfläche

Spezialfall: Hinzufügen von Tabs:

1. Rechtsklick auf bestehenden Tab
2. Seite Einfügen -> Danach auswählen
3. *currentTabText* unter *Element-Details* eingeben
4. *currentTabName* unter *Element-Details* ändern

Eigenschaft	Wert
tabPosition	North
tabShape	Rounded
currentIndex	4
iconSize	16 x 16
elideMode	ElideNone
usesScrollButtons	☑
documentMode	☐
tabsClosable	☐
movable	☐
currentTabText	Seite
currentTabName	tab_3
currentTabIcon	
currentTabToolTip	
currentTabWhats...	



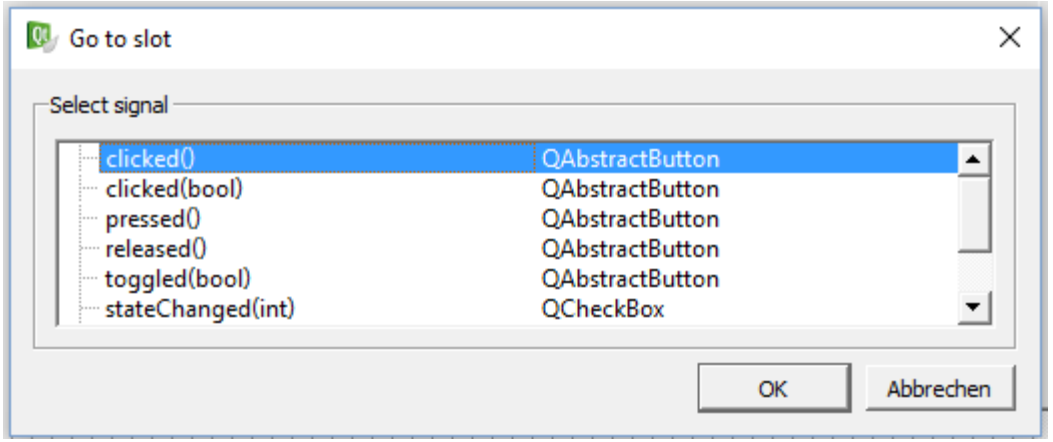


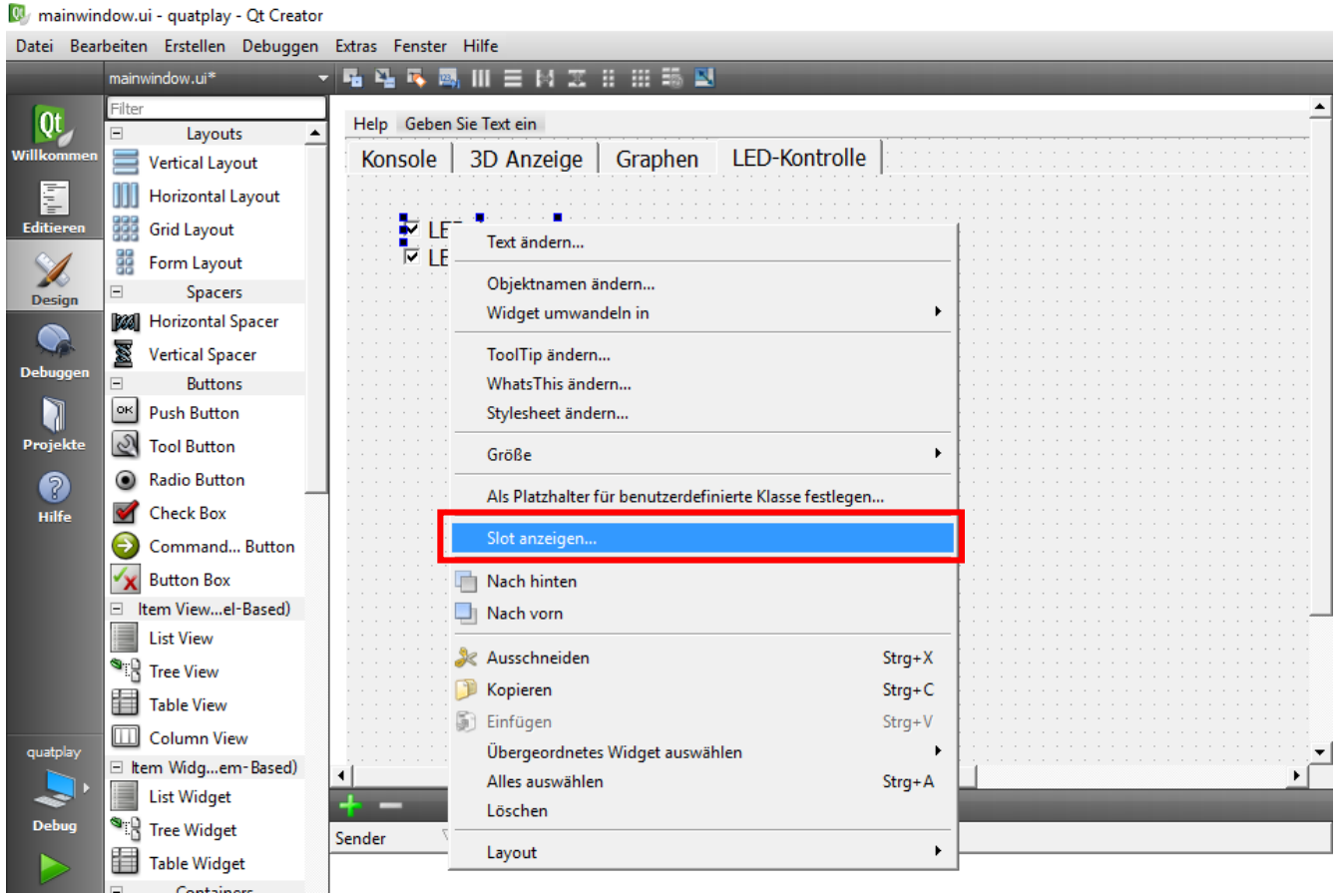
Bearbeitung der Benutzeroberfläche

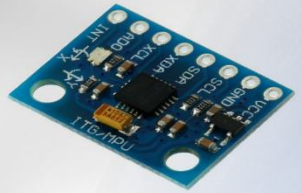
Funktionalität programmieren:

1. Rechtsklick auf Element
2. *Slot anzeigen…* auswählen
3. Gewünschte Aktion auswählen und mit *OK* bestätigen

→ Entsprechende Funktion wird unter *mainwindow.cpp* angelegt







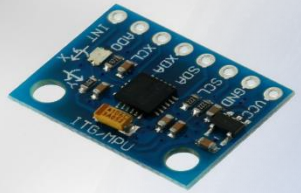
Aufgaben

Notwendige Hardware:

- EVK1100
- Mikro USB Kabel für Strom und zum Flashen
- QCS / QCSF

Notwendige Software:

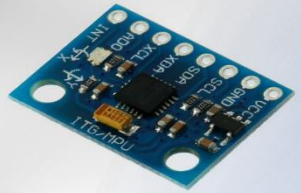
- AVR Studio 32 installiert (mit Tool Chain und Flip Treiber)
- Qt SDK Version 4.8.1 oder 4.7.4.
- Quatplay Qt Framework (Code)



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.

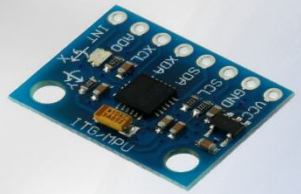


Aufgaben

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.



Aufgaben

Aufgabe 4:

a. Erstellen Sie einen neuen Reiter neben dem Tab Graphen mit der Beschriftung „LED-Kontrolle“. Fügen Sie im neu erstellten Bereich zwei CheckBoxen LED 1 und LED 2 hinzu, die bei Programmstart aktiviert sind.

b. Programmieren Sie die neuen CheckBoxen so, dass man damit die LEDs 1 und 2 des EVK ansteuern kann. `void createFrameMessageAndSend(int type, int data_length, float data [])` in *mainwindow.cpp* sendet einen Frame über *USART* zum *EVK1100.* Um die Telekommandos zu verarbeiten, sind die Funktionen `void my_read_telecommand()` und `void my_read_LED_Command()` in *telecommand.c* zu implementieren. Die Funktion `int extract_type_and_data_from_telecommand_frame()` hilft Ihnen, den Frametyp des Telekommandos auszulesen. Vergleichen Sie dazu die *defines* in *protocol.h* .