1 Software

Für die verschiedenen Versuche und die Umsetzung des Regelkreises müssen Software-Applikationen entwickelt werden. Die Hauptaufgabe besteht in der Berechnung des Regelkreises. Hierfür muss eine HW-Schnittstelle implementiert werden, welche es ermöglicht die Sensorik auszuwerten und die Aktoren anzusteuern. Anschließend müssen die Teilsysteme des Regelkreises berechnet werden. Hierunter fallen die Evaluierung der Sensordaten, die verschiedenen Filter und der letztendliche Regler. Hierbei müssen Echtzeitanforderungen eingehalten werden, welche empirisch im Entwicklungsprozess bestimmt werden. Abbildung 1 zeigt den Signalfluss des Regelkreises. Für die Durchführung der Ver-

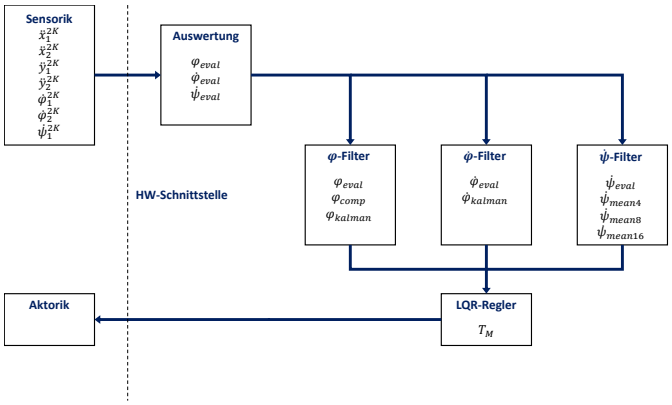

Abbildung 1: Signalfluss Regelkreis, Quelle: eigene Darstellung

suche müssen die dargestellten Signale an eine MATLAB-Applikation übertragen werden, welche auf dem Host-PC ausgeführt wird. Hierunter fallen die folgenden Daten-Pakete.

- Sensordaten, Rohwerte der Sensoren in 2K-Darstellung
- φ-Daten, Werte der Sensorevaluierung, des Komplementär- und Kalman-Filters
- $\dot{\varphi}$-Daten, Werte der Sensorevaluierung und des Kalman-Filters
- $\dot{\psi}$-Daten, Werte der Sensorevaluierung und der Mittelwert-Filter

Andererseits muss die Anwendung in der Lage sein, Steuerbefehle der MATLAB-Applikation zu empfangen und umzusetzen. Hierunter fallen die folgenden Anweisungen.

- Start- bzw. Stopp-Befehl
- Auswahl des φ-Filter
- Auswahl des $\dot{\varphi}$-Filter
- Auswahl des $\dot{\psi}$-Filter
- Setzen des φ-Offset für die Berechnung des Reglers
- Setzen des $\dot{\varphi}$-Offset für die Berechnung des Reglers
- Setzen des $\dot{\psi}$-Offset für die Berechnung des Reglers
- Setzen eines fixen Motormoment

Folglich muss ein Kommunikationsprotokoll implementiert werden um die Verbindung zwischen dem BBB und dem Host-PC herzustellen. Hierbei fällt die Entscheidung auf eine TCP/IP-Verbindung wobei die BBB-Anwendung als Server agiert. Die Gründe für das gewählte Protokoll sind einerseits die komfortable Implementierung auf beiden Seiten, andererseits die Sicherung der fehlerfreien Datenübertragung.

1.1 Allgemeiner Softwareentwurf

Die oben genannten Aufgaben werden auf zwei logische Komponenten verteilt, welche als separate Tasks ausgeführt werden. Hierbei übernimmt die Regelungskomponente die Berechnung des Regelkreises und die Bereitstellungen der Daten. Die Kommunikationskomponente betreibt den TCP/IP-Server um Daten an MATLAB zu senden bzw. zu empfangen. Zusätzlich muss ein dritter Task realisiert werden, welcher für die Zeitgebung verantwortlich ist.

Kommunikation zwischen den Komponenten

Der Datenaustausch zwischen den Tasks wird mit Hilfe von Nachrichten implementiert, welche sich aus Steuerinformationen über Typ bzw. Inhalt und einem Datenpaket zusammensetzen. Die Grund für die Entscheidung Nachrichten für die Datenübertragung zu verwendet ist einerseits, dass nur kleine Datenpakete übermittelt werden müssen, andererseits können die Nachrichten als TCP/IP-Paket wiederverwendet werden. Somit besteht die Aufgabe der Kommunikationseinheit in der Weiterleitung von Nachrichten mit Regeldaten an MATLAB bzw. der Weiterleitung von Nachrichten mit Befehlen an den Proxy. Diese werden von einem zentralen Proxy erzeugt und zugestellt. Da die Empfänger der verschiedenen Nachrichtentypen vor Laufzeit bekannt sind ist keine dynamische Registrierung der Komponenten erforderlich.

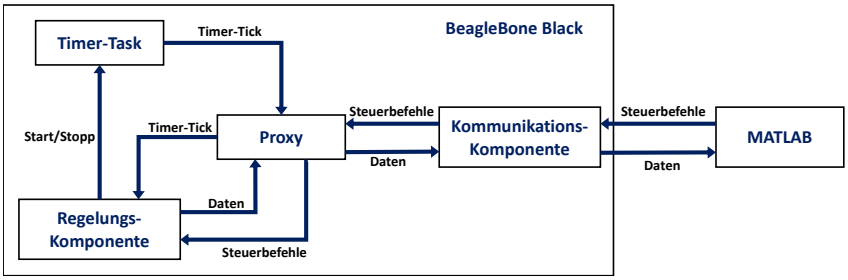

Abbildung 2: SW-Kommunikation, Quelle: eigene Darstellung

Nach Abbildung 2 werden Nachrichten für die Übertragung von Daten aus dem Regelkreis und für die Zustellung von Befehlen verwendet. Folglich setzt sich der Header aus dem Nachrichtentyp und dem Befehl bzw. Datentyp zusammen. Die Größe des Datenfeldes ist fix und wird nach dem größten Signalpaket ausgerichtet. Dadurch vereinfacht sich die Übertragung der Nachrichten über den TCP/IP-Stream.

```
1  class CMessage
2  {
3  public:
4    struct CHeader
5    {
6    EMessageType mType;
7    ECommand mCommand;
8    EDataType mDataTyp;
9    }
10   ...
11 private:
12   CHeader mHeader;
13   static constexpr Int32 sSize = 20;
14   UInt8 mData[sSize];
15 };
```

Um Nachrichten zu empfangen verfügen die Komponenten über Eingangspuffer, welche als Queues implementiert werden. Das Grundgerüst der Komponenten wird in einer abstrakten Basisklasse festgelegt, welche rein virtuelle Methoden zur Initialisierung und Ausführung deklariert. Die *dispatch*-Methode ist für die Verarbeitung von Nachrichten zuständig.

```
1  class AComponentBase
2  {
3  public:
4    virtual void init() = 0;
5    virtual void run() = 0;
6  protected:
7    TQueue<Config::QueueSize> mQueue;
8  }
```

Die Erzeugung und Zustellung von Nachrichten übernimmt ein Proxy, welcher Methoden für die verschiedenen Ereignisse zur Verfügung stellt. Dadurch wird die Kommunikationsstruktur gekapselt, wodurch eine übersichtliche Applikation entsteht. Der Proxy kennt die Eingangsqueues der Komponenten und reicht die Nachrichten nach Typ an die Empfänger weiter.

```
class CProxy
{
public:
  bool clientDisconnect(bool waitForever);
  bool transmitSensorData(const CSensorData& data, bool waitForever);
  bool transmitPhi(const CPhi& data, bool waitForever);
  bool transmitPhi__d(const CPhi__d& data, bool waitForever);
  bool transmitPsi__d(const CPsi__d& data, bool waitForever);
  bool timerTick(bool waitForever);
  bool routeMATLABMessage(bool waitForever);
  ...
private:
  TQueue& mControlQueue;
  TQueue& mCommQueue;
}
```

Aufbau der Regelungskomponente

Die Aufgaben der Regelungskomponente bestehen einerseits in der Interaktion mit der HW und der Berechnung des Regelkreises. Diese Teilsysteme werden als Klassen implementiert, welche Methoden bereitstellen um die Sensorwerte auszulesen, die Motoren zu steuern und die Filterwerte bzw. den Regler zu berechnen. Zusätzlich gehört der Timer-Task zu der Regelungskomponente und wird als eigenständiger Thread ausgeführt.

Für die verschiedenen Versuche ist eine zusätzliche Kontrolllogik erforderlich, welche in Form eines Zustandsautomaten implementiert wird. Die FSM verarbeitet die empfangenen Nachrichten und passt den aktuellen Zustand und somit das Verhalten entsprechend an. Auf der obersten Schicht verfügt das Statechart über den Zustand *STANDBY* und Zustände für die verschiedenen Versuche. Hierdurch wird das An- bzw. Abschalten der Komponente ermöglicht. Dadurch kann dieselbe Applikation flexibel eingesetzt und während des Entwicklungsprozesses erweitert werden. Die letztendlichen Aktionen werden in der Klasse *CControlAciton* gekapselt um eine klare Trennung von Kontroll- und Signalfluss zu gewährleisten.

Bei Betreten der Versuch-Zustände wird der Softtimer gestartet, welcher zyklisch Timer-Events an die Regelungskomponente sendet. Bei Erhalten einer solchen Nachricht führt die FSM die, dem aktuellen Zustand entsprechende, Logik aus. Die ersten drei Versuche sind aus Sicht des Kontrollflusses identisch, aus Gründen der Einheitlichkeit werden dennoch drei explizite Zustände implementiert. In diesen Zuständen werden bei Eintreffen eines Timer-Events die aktuellen Sensorwerte abgefragt und and die MATLAB-Applikation gesendet. Für die Versuche 4 und 5 müssen die Sensorauswertung und die Filter implementiert werden. Deren Ausgangswerte werden wiederum zyklisch berechnet und mit Hilfe des Proxys an MATLAB übertragen. Versuch 6 besteht in der Umsetzung des Reglers. Der entsprechende Zustand muss einerseits den vollständigen Regelkreis berechnen, als auch die Filter- und Reglerwerte an MATLAB übertragen. Zusätzlich muss der Zustand die Befehle zur Auswahl der Filter und zum Setzen der Offsets annehmen und umsetzen.

Aufbau der Kommunikationskomponente

Die Hauptaufgabe der Kommunikationseinheit besteht in dem Betrieb des TCP/IP-Server und der Weiterleitung von Nachrichten. Der Server wird in Form einer Klasse implementiert, welche Methode zum Empfangen und Versendet von Nachrichten bereitstellt.

```
class CServer
{
  public:
  void init();
  void waitForConnection();
  bool transmitMessage(CMessage& msg, bool waitForever);
  bool receiveMessage(CMessage& msg, bool waitForever);
  ...
  private:
    Int32 mSocketFD;
    Int32 mConnectionFD;
}
```

Hierfür wird ein zweiter Thread gestartet, welcher auf Nachrichten der MATLAB-Anwendung wartet und diese an den Proxy weiterleitet. Die Kontrolllogik der Komponente wird ebenfalls in Form einer Zustandsmaschine implementiert. Hierbei wird lediglich zwischen den Zuständen *STANDBY*

und *RUNNING* unterschien. Im Ausgangszustand wartet die Komponente auf die Verbindungsanfrage eines Client, welcher die MATLAB-Anwendung ist. Der Haupt-Thread der Komponente wartet auf eingehende Kommunikationsnachrichten und verarbeitet diese entsprechend. Der zweite Thread wartet zuerst auf die Verbindungsanfrage des Client, diese signalisiert er mittels einer Nachricht. Daraufhin wechselt die Komponente in den *RUNNING*-Zustand, in welchem empfangene Nachrichten mit Regelungsdaten an MATLAB weitergeleitet werden. Der zweite Thread wartet nun auf eingehende TCP/IP-Daten von MATLAB. Sowohl beim senden als auch empfangen von Daten kann ein Verbindungsabbruch des Clients erkennt werden. In diesem Fall wechselt die Komponente in den *STANDBY*-Zustand und der zweite Thread wartet auf eine neue Verbindungsanfrage.