Labor Embedded Systems 2

Versuch	Vorbereitung	Durchführung	Fertigstellung	Summe
1	2 h	3 h	2 h	7 h
2	2 h	3 h	8 h	13 h
3	2 h	3 h	2 h	7 h
4	2 h	3 h	4 h	9 h
5	2 h	3 h	4 h	9 h
6	2 h	3 h	10 h	15 h
Summe	12 h	18 h	30 h	56 h

Versuch 6b:

(Prozess-)Automatisierung, Automatisierungspryramide

Raspberry Pi 4 mit RPI SENSE HAT Shield

Embedded Linux mit Qt

Einbindung der Sensoren

Version 0.8

Inbetriebnahme (Wiederholung)

Die Inbetriebnahme erfolgt nach den folgenden Schritten:

- Starten Sie die VM in bwLehrpool.
- Schließen Sie das Netzteil an das Raspberry Pi 4 B Board an. Beim Hochfahren leuchtet das 8x8 LED-Matrix-Display. Die rote LED am Raspberry zeigt an, ob eine Spannungsversorgung anliegt. Die kleinere grüne LED zeigt an, ob der Bootvorgang noch läuft.
- Verbinden Sie danach Raspberry Pi 4 B mit dem Hostrechner (VM Fischerman) mit dem ausgeteilten USB-Netzwerk-Kabel.
- Starten Sie auf dem Hostrechner „Netzwerkverbindungen anzeigen“
- Es müsste jetzt „Ethernet 2“ (ASIX …) angezeigt werden. Rechtsklick auf das Symbol und Eigenschaften anklicken. In der VM müssen Sie nun ein Passwort eingeben. Dieses wird Ihnen am Labortag mitgeteilt.
- Wählen Sie in „Eigenschaften von Ethernet 2“ „Internetprotokoll, Version 4 (TCP/IPv4)“ an und betätigen Sie die Schaltfläche „Eigenschaften“.
- Geben Sie im erscheinenden Fenster als IP-Adresse 10.10.1.20 und als Subnetzmaske 255.255.255.0 ein. Schaltfläche „Okay“ drücken und mit „Schließen“ das Fenster „Eigenschaften von Ethernet 2“ schließen.
- Starten Sie nun den VNC Viewer und geben Sie als VNC Server-Adresse 10.10.1.200 ein.
- Im Authentifizierungsfenster ist als Benutzername „pi“ und als Kennwort „raspberry“ einzugeben. Die Warnung können Sie wegklicken.

Jetzt müsste der Desktop vom Raspberry im VNC Viewer erscheinen. Sie können nun auf den Raspberry zugreifen und können jetzt nach Anwahl der Raspberry Symbols den Menüpunkt „Entwicklung“ und „Qt Creator“ anwählen.

Aufgabenstellung

Es soll eine Qt-Anwendung mit einem Fenster (später Fullscreen) erstellt werden. Die Anwendung soll die Prozessleitebene abdecken (Name PLS: Prozess LeitSystem). Als exemplarische Prozessdaten sollen die Sensordaten des RPI SENSE HAT Shields dienen.

- Temperatur
- Luftdruck
- Luftfeuchtigkeit
- Kompass (Magnetometer): x, y und z

Gyroskop (Winkelgeschwindigkeitssensor) und Accelerator (Beschleunigungssensor) sollen hier nicht verwendet können. Selbstverständlich können Sie dies aber noch durchführen, falls Zeit vorhanden ist.

Diese Prozessdaten sind in entsprechenden Controls (z.B. Labels) darzustellen. Die LED-Matrix kann selbstverständlich ebenso noch zur Visualisierung von Prozessdaten eingesetzt werden.

In diesem Versuch wird **nicht** realisiert:

1. Anbindung an die Betriebsleitebene (Industrie 4.0)
2. Anbindung an die Cloud (IoT)
3. Anbindung an die Steuerungsebene (Industrie 4.0) – z.B. an das MCB1760

Welche Schnittstellen/Technologien würden Sie für 1.-3. verwenden?

Schritt 1: Projekt anlegen und GUI erstellen.

Legen Sie sich ein neues Projekt unter /home/pi/QtProjects/PLS an. Dabei soll als Projekttyp Qt-Widget ausgewählt werden. Zum Zugriff auf die obigen Sensordaten ist eine Bibliothek notwendig, welche schon auf dem Raspberry Pi zur Verfügung steht. Diese Bibliothek muss noch in PLS.pro hinzugefügt werden.

Fügen Sie dazu in PLS.pro unterhalb von FORMS += …

```
LIBS += -lRTIMULib
```

ein.

Wählen Labels zur Ausgabe der Werte. Sie können selbstverständlich die Eigenschaften der Labels so ändern, dass z.B. noch ein Rahmen um das Textfeld erscheint.

Ebenso soll ein PushButton mit dem objectName PBReadProcessImage auf dem Hauptfenster erscheinen. Legen Sie für diesen das Signal clicked() an.

Schritt 2: Klasse ProcessImage

Erstellen Sie die Klasse ProcessImage. Eine Instanz dieser Klasse wird im Mainwindow instanziiert (Assoziation). Diese Klasse soll alle Prozessparameter enthalten und soll die Bibliothek RTIMUlib verwenden (`#include "RTIMULib.h"`). Leider gibt es keine wirkliche gute API-Beschreibung für C++. Für Python sieht die Schnittstelle deutlich schlanker und besser dokumentiert aus.

Die Prozessdaten sollen als private Member implementiert werden.

```
class ProcessImage
{
private:
    bool bInitFlag = true;

    float fTemperature = 0.1f;
    float fAirPressure = 0.1f;
    float fHumidity    = 0.1f;
    float fxMagnetometer = 0.1f;
    float fyMagnetometer = 0.1f;
    float fzMagnetometer = 0.1f;

    RTIMUSettings* pRTIMUSettings = nullptr;
    RTIMU* pRTIMU = nullptr;
    RTPressure* pRTPressure = nullptr;
    RTHumidity* pRTHumidity = nullptr;

public:
    ProcessImage();
    bool vReadProcessImage(void);
    //ToDo: Getter-Functions implizit inline
};
```

Die Initialisierung der RTIMULib soll dabei im Konstruktor erfolgen.

```
   pRTIMUSettings = new RTIMUSettings("RTIMULib");
   pRTIMU         = RTIMU::createIMU(pRTIMUSettings);
   pRTPressure    = RTPressure::createPressure(pRTIMUSettings);
   pRTHumidity    = RTHumidity::createHumidity(pRTIMUSettings);

   if ((pRTIMU != nullptr) && (pRTIMU->IMUType() != RTIMU_TYPE_NULL) &&
       (pRTPressure != nullptr) && (pRTHumidity != nullptr))
   {
       pRTIMU->IMUInit();
       pRTIMU->setSlerpPower(0.02f);
       pRTIMU->setAccelEnable(false);
       pRTIMU->setGyroEnable(false);
       pRTIMU->setCompassEnable(true);
       pRTPressure->pressureInit();
       pRTHumidity->humidityInit();
   }
   else
   {
      bInitFlag = false;
   }
```

Die `bool ProcessImage::vReadProcessImage` Funktion liest die Prozessdaten über die Bibliothek ein und speichert die Werte in den Objektvariablen.

Erweitern Sie den Code noch, sodass immer erst überprüft wird ob die Daten valide sind. Selbstverständlich können Sie die Klasse noch erweitern, dass der Accelerator und das Gyroskop ebenso angesprochen werden.

```
bool ProcessImage::vReadProcessImage(void)
{
    if (bInitFlag)
    {
         if (pRTIMU->IMURead())
         {
             RTIMU_DATA RTIMUData = pRTIMU->getIMUData();

             pRTPressure->pressureRead(RTIMUData);
             pRTHumidity->humidityRead(RTIMUData);

             fAirPressure = // todo
             fTemperature = // todo
             fHumidity    = // todo

             RTIMUData.compass.normalize();

             fxMagnetometer = // todo
             fyMagnetometer = // todo
             fzMagnetometer = // todo
         }

    }
    return bInitFlag;
}
```

Schritt 3: GUI mit ProcessImage verbinden

Im Konstruktor des Mainwindows ist eine Instanz von ProcessImage zu instanzieren. Legen Sie sich für PBReadProcessImage das onclicked()-Event an.

In dieser EventHandler-Funktion ist das Einlesen der Prozessdaten zu starten und danach Sie die Werte über die Getter-Funktion abzurufen und die Werte sind in den Controls darzustellen. Verwenden Sie die cast von C++:

```
static_cast<double>(VariableToBeCasted)
```

Die Prozessdaten sind vom Datentyp float. QString biete aber nur ein Klassenmethode number an, um einen double in einen String zu konvertieren. Die setText-Methode eines Labels erwartet einen QString.

Grundsätzliche Diskussion bei der Programmierung mit Qt und C++ (nicht nur auf einem Embedded System):

C++ kann in den letzten Jahren deutliche Fortschritte (Spracherweiterungen) aufweisen. Es ist jetzt die Frage, wie tief im System man die Klassen von Qt noch verwenden soll? Nur noch zur GUI?

Wenn Sie nun den PushButton PBReadProcessImage drücken, dann werden die aktuellen Prozessdaten angezeigt. Die Temperatur können Sie selbst etwas durch „Luftkühlung“ beeinflussen, während die Parameter des Magnetometers durch Drehen des Raspberry PI sich verändern.

Schritt 4: Timer

Legen Sie sich nun in Qt einen QtTimer an (Googlerecherche). Dieser soll jetzt mit einer sinnvollen Updaterate die Prozessdaten automatisch einlesen und in der GUI darstellen. Fügen Sie ein Signal/Slot ein, damit der Timerevent an eine GUI-Funktion geschickt wird.

Schritt 5: Thread (Advanced)

Legen Sie sich nun einen Thread (Googlerechere) an, der das Einlesen der Prozessdaten durchführt. Dieser muss gegenüber der GUI sicher sein.