PRP2 Praktikumsaufgabe 1: Ansteuerung einer Festo-Anlage

Lernziele

Sie können ...

- ... ein C++-Projekt von einem Repository-Server clonen und in einer IDE weiterentwickeln.
- ... das Festo-Transfersystem über eine OO-Schnittstelle ansteuern.
- ... das weiterentwickelte C++-Projekt auf den Repository-Server hochladen.
- ... Fehler mittels Debugging-Methoden finden und beheben.

Allgemeine Hinweise

Vorbereitung und Durchführung:

- Bitte bereiten Sie die nachfolgenden Aufgaben so weit wie technisch möglich vorm Praktikumstermin vor.
- Um den größten Lerneffekt zu erzielen, wird empfohlen, dass zunächst jeder von Ihnen versucht, die Aufgaben selbständig zu lösen. (Tauschen Sie sich bei Problemen in Ihrer Gruppe oder mit anderen Kommilitonen aus.)
- Die Lösungen sind am Anfang des Praktikums zu präsentieren und ggf. bis zum Ende des Praktikums zu vervollständigen.
- **Die Lösungen sind *nach erfolgreicher* Abnahme auf den Git-Server hochzuladen (push).**

Bitte beachten:

- **Verändern Sie nicht den mitgelieferten Source Code.** (Außer an Stellen, an denen dies explizit erlaubt ist. (siehe README))
- Verwenden Sie ausschließlich die in der Aufgabenstellung angegebenen Bezeichner und Namen (z.B. für Funktionen, Variablen usw.).
- Achten Sie auf Ihren *Coding Style*.
- Achten Sie auf ausreichende und sinnvolle Kommentare in Ihrem Code.

Erfolgreiche Teilnahme / Abnahme:

- Es besteht Anwesenheitspflicht bei allen Praktikumsterminen.

- Zu jedem Programm **muss** ein Design (d.h. z.B. UML Klassendiagramm oder Zustandsmaschine) angefertigt werden. Diese sind zuerst vorzuzeigen! **Ohne Design wird der Code nicht abgenommen!**
- Die erfolgreiche Abnahme *aller* Aufgaben **bis zum Ende des jeweiligen Praktikumstermins.**
- Die Abnahme erfolgt **individuell** und nicht pauschal für jedes Team.
- Der Code **muss** sinnvollen, einheitlichen Codingstyles entsprechen. ("C-Coding Style" in Teams)
- Abnahme im Termin:
  - Vorführung der vollständigen Lösungen
  - Befragung zu der Bearbeitung, den Lösungen sowie verwandten theoretischen Themen
- Abgabe der Lösung erfolgt durch push auf den git-Server. Im push enthalten sein müssen:
  - Design-Dokumente (Idee, Algorithmen, Struktur der Lösung)
  - Source-Code
  - Commit-Message des letzten Commits muss "final-PraktikumX" sein (X=Praktikumsnummer (1-4)).
  - Optional: Testergebnisse/Screenshots etc.

1 Das Festo Transfersystem

Ein Transfersystem der Firma Festo besteht aus einem Laufband mit diversen Sensoren und Aktuatoren, die per C++-Programm ausgewertet und angesteuert werden sollen. Die Sensoren und Aktuatoren des Transfersystems werden technisch über eine USB-Schnittstelle angesprochen. Eine C++-Klasse kapselt diese USB-Schnittstelle und stellt über weitere Klassen und deren Methoden den Zugriff her.

Hinweis: Um die Anlagen im Labor anzusteuern nutzen Sie bitte die Laborrechner. Hier sind alle notwendigen Treiber vorinstalliert. Die Anlage muss nur per USB mit dem Laborrechner verbunden werden und mit Strom versorgt werden. (Achtung: Das Netzteil unter dem Anlagen-Tisch muss auch

angeschaltet sein!) An der schwarzen Platine hinten an den Anlagen muss der Schalter auf USB (untere Position) gestellt sein (siehe Hinweis in Teams).

Zusätzlich zur Hardware gibt es einen Simulator, den Sie zum Testen zuhause nutzen können. Die Aufgaben können also an der Hardware aber auch in der Simulation vorbereitet werden. (Die Simulation hat die gleichen Eigenschaften, die im Folgenden für die Festo-Anlagen beschrieben sind.) Der Simulator ist in dem git-Repostitory des Praktikums enthalten (./simulationcore und ./simulationadapterbmt). Konkrete Beschreibung des Simulators entnehmen Sie der ./festo/README.md.

2 Ansprechen des Festo-Transfersystems

Das Festo-Transfersystem kann über eine Klasse angesprochen werden, welche die Kommunikation über das USB-Interface kapselt. Auf die interne binäre Darstellung wird mit Hilfe von weiteren Klassen, die an die Klasse gebunden sind **(Stichwort: Objekthierarchie)**, zugegriffen.

2.1 Klasse für den Zugriff auf das Festo-Transfersystem

Ein Objekt der Klasse `FestoTransferSystem` stellt Objekte und Methoden für die Kommunikation mit den Sensoren und Aktuatoren des Festo-Transfersystems bereit. Folgende Methoden gelten für das Gesamtsystem:

- `updateSensors()`: Liest den aktuellen Zustand der Sensorwerte über das USB-Interface aus und stellt die Werte intern zu weiteren Auswertung den entsprechenden Sensorobjekten zur Verfügung.
- `updateActuators()`: Die Methode überträgt die von den entsprechenden Aktuatorobjekten eingeplanten Ansteuerungen über das USB-Interface zu den Aktuatoren.

Die Methoden sollte nur aufgerufen werden, wenn dieses erforderlich ist, da der Datenaustausch ca. 100-200 ms in Anspruch nimmt.

2.2 Sensorik

Ein Objekt der Klasse `FestoTransferSystem` stellt über weitere Objekte (Attribute der Klasse) den eigentlichen Zugriff auf Sensoren bzw. Aktuatoren bereit. In der folgenden Tabelle sind die Objekte und die Methoden zum Zugriff auf die Sensoren aufgelistet, sowie die Bedeutung (Semantik) der Ergebnisse.

Zu beachten: die Rückgabewerte entsprechen dem Anlagenzustand beim letzten Aufruf der Methode `updateSensors()`.

Tabelle1: Ergebnisse von `getState()`

Objekt	True	False
`lightbarrierBegin`	Lichtschranke 1 geschlossen	Lichtschranke 1 unterbrochen
`lightbarrierHightSensor`	Lichtschranke 2 geschlossen	Lichtschranke 2 unterbrochen
`lightbarrierFeedSeparator`	Lichtschranke 3 geschlossen	Lichtschranke 3 unterbrochen
`lightbarrierBufferFull`	Lichtschranke 4 geschlossen	Lichtschranke 4 unterbrochen
`lightbarrierEnd`	Lichtschranke 5 geschlossen	Lichtschranke 5 unterbrochen
`pushbuttonStart`	Taste Start gedrückt	Taste Start nicht gedrückt
`pushbuttonReset`	Taste Reset gedrückt	Taste Reset nicht gedrückt
`pushbuttonStop`	Taste Stop nicht gedrückt	Tast Stop gedrückt
`switchEmergency`	Schalter Emergency nicht aktiv	Schalter Emergency aktiv

2.3 Aktuatorik

Ein Objekt der Klasse `FestoTransferSystem` stellt über weitere Objekte (Attribute der Klasse) den eigentlichen Zugriff auf Aktuatoren bereit. Die Aktionen/Ansteuerungen werden dabei zunächst intern vermerkt und gemeinsam bei Aufruf der Methode `updateActuators()` an das Festo-Transfersystem übertragen. In der folgenden Tabelle sind die Objekte und die Methoden zum Zugriff auf die Aktuatoren aufgelistet, sowie die Bedeutung (Semantik) der Aufrufe.

Tabelle2: Parameter für `setState()`

Objekt	True	False
`ledStart`	LED ein	LED aus
`ledReset`	LED ein	LED aus
`ledQ1`	LED ein	LED aus
`ledQ2`	LED ein	LED aus
`feedSeparator`	Weiche geöffnet	Weiche geschlossen
`lampRed`	Lampe an	Lampe aus
`lampYellow`	Lampe an	Lampe aus

Objekt	True	False
`lampGreen`	Lampe an	Lampe aus

Der Motor des Förderbands wird über Werte angesteuert, die eine bestimmte Bewegung codieren. Die Klasse `BeltDrive`, als Objekt `drive` über die Klasse `FestoTransferSystem` verfügbar, stellt zur Steuerung des Motors die Methode `setSpeed()` bereit.

Tabelle3: Motorsteuerung

Wert des Parameters	Bedeutung
0	Band steht
1	Langsam nach links
2	Schnell nach links
3	Langsam nach rechts
4	Schnell nach rechts
5-255	Band steht

3. Aufgaben

Die folgenden Teilaufgaben sind vor dem Praktikum zu bearbeiten. Die Lösungen sollen den Betreuern im Praktikum vorgestellt werden.

Achtung: Vermeiden Sie grundsätzlich, die mitgelieferten Dateien zu verändern! Also: nutzen, nicht modifizieren! (siehe README)

Ausnahme: Wenn Sie zu Hause arbeiten wollen (und dementsprechend die Festo-Hardware nicht zur Verfügung steht), können Sie in den Simulationsmode umschalten, in dem Sie in der Datei `.../festo/CMakeLists.txt` in der Zeile

```
# add_definitions(-DHOME -DSIMULATION) #Schaltet die Simulation an
```

das erste #-Zeichen entfernen.

3.1 Entwicklungsumgebung einrichten

Clonen Sie das Repository vom Praktikum in Clion oder per Konsole in ein lokales Verzeichnis (z.B. C:\temp – bitte kein Netzlaufwerk oder Cloud). Der Link zum Repository wird jedem Team individuell zugeteilt.

3.2 Ansteuerung der Lampen

Schreiben Sie ein Programm, das unter Verwendung eines Objektes der Klasse `FestoTransfersystem` die Lampen der Anlage ansteuert. Es soll fünfmal die Folge:

1. Rot
2. Rot + Gelb
3. Grün
4. Gelb

aufleuchten. Die Umschaltung zur nächsten Phase erfolgt jeweils nach 1 Sekunde.

Nach Ausführung der Folge leuchtet nur die gelbe Lampe.

Um das Programm für eine bestimmte Anzahl Milisekunden warten zu lassen kann man die Funktion `sleepForMs(unsigned int ms)` aus dem Header "`TimeHelper.h`" verwenden. (Siehe `main.c`)

3.3 Anzeige der aktuellen Sensorwerte

Ergänzen Sie Ihr Programm aus der vorigen Teilaufgabe um folgende Funktionalität: Bei jedem Umschalten der Farbe werden die aktuellen Zustände der Lichtschranken auf der Konsole ausgegeben.

Hinweis: Falls Sie diese Funktionalität in eine Funktion auslagern wollen, die Referenz auf das Objekt für den Zugriff auf das Festo-Transfersystem muss per Call-by-Reference übergeben werden. Im gesamten Programm darf nur ein Objekt der Klasse `FestoTransfersystem` existieren.

3.4 Interaktive Ansteuerung

Ergänzen Sie Ihr Programm aus der vorigen Teilaufgabe um folgende Funktionalitäten, die eine interaktive Steuerung der Aktuatoren ermöglicht. Die Funktionalität soll erst nach dem Durchlauf der Farbfolge zur Verfügung stehen.

- Wird die Start-Taste gedrückt und gehalten, so bewegt sich das Laufband schnell nach rechts. Die LED der Start-Taste leuchtet, wenn die Start-Taste gedrückt wird.

  - Optional: Das Laufband hält automatisch an, wenn die letzte Lichtschranke unterbrochen wird.

- Wird die Stop-Taste gedrückt und gehalten, so öffnet sich die Weiche. Die LED der Reset-Taste leuchtet, während die Stop-Taste gedrückt wird.
- Wird die Lichtschranke am Anfang nicht unterbrochen, so leuchtet die LED Q1.
- Wird die Lichtschranke an der Höhenmessung unterbrochen, so leuchtet die LED Q2.
- Wird der Emergency Schalter betätigt, so endet Ihr Programm.