Hochschule
Fachbereich

Projektarbeit

Steuerungstechnik eines Fischertechnik-Industrieroboters

verfasst von
Bojan S

Embedded C++
Dozent: Prof. Dr. T. B.

Abgabetermin: 01. Juli 2020

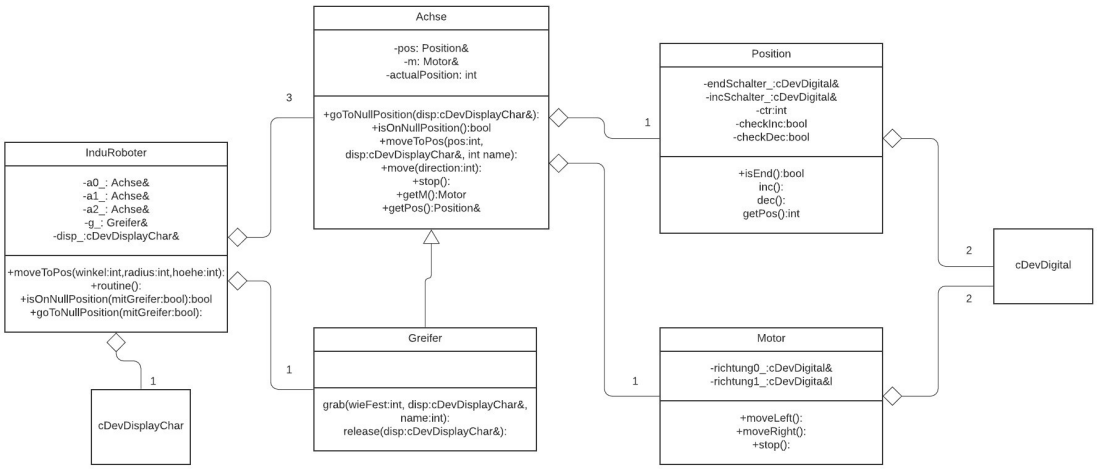
Gliederung

1. Vorwort

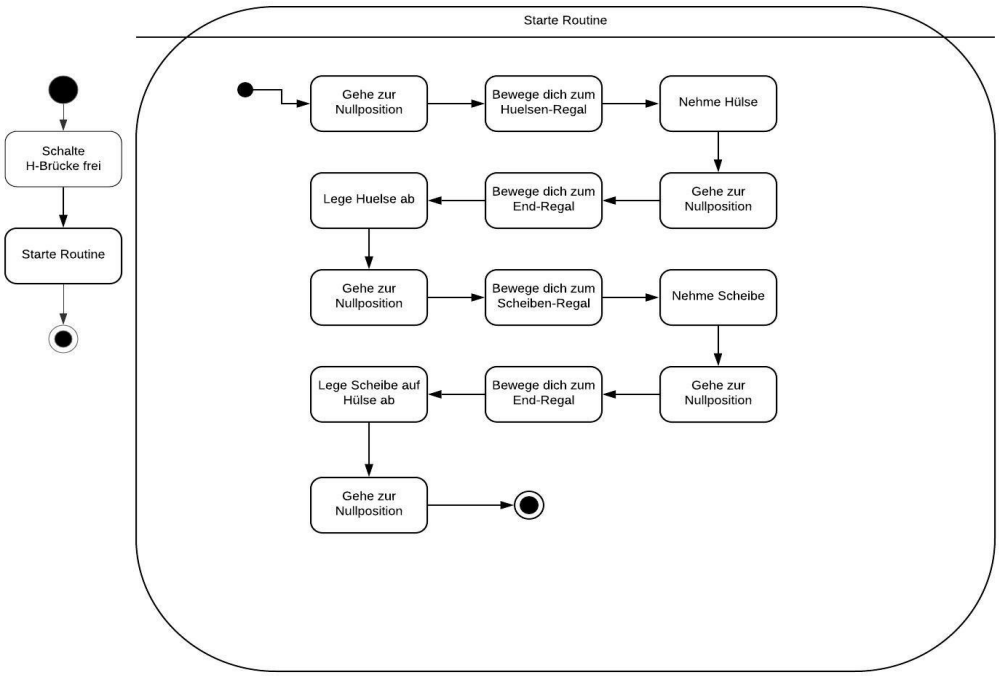
In dem Zustandsdiagramm habe ich mich lediglich auf den Roboter und nicht noch das Anzeigen der Position auf dem LCD Bildschirm beschränkt. Im Zustandsdiagramm sieht man, dass der Industrieroboter sich nach jedem Besuchen eines Regals immer in die Null-Position begibt. Obwohl diese Möglichkeit in der zeitlichen Länge zwar ineffizienter ist, ist sie jedoch in der Gewährleistung funktionaler Sicherheit besser geeignet, als die andere Option direkt zum nächsten Regal zu fahren. Der Grund liegt hierbei, dass man der möglichen Kollision mit potentiellen weiteren Lagerregalen aus dem Weg geht.
Ich habe zusätzlich eine weitere Funktion implementiert, mit der man den Roboter manuell steuern kann. Dies wahr vor allem zu Testzwecken nützlich und findet meines Erachtens auch seinen Platz bei richtigen Industrierobotern, für den Fall, dass unerwartetes Fehlverhalten auftritt und man den Roboter selbst steuern muss. Hierbei werden aber direkt die Motoren über die Motor-Objekte angesprochen. Dies fand ich sinnvoller, als über die Roboter-Klasse zuzugreifen, da sonst weitere Funktionen in der InduRoboter-Klasse nötig wären, die die notwendige Architektur für die eigentliche Aufgabe überflüssigerweise erweitern würde. Das Greifen des Roboter-Greifers habe ich nicht, wie gefordert, über eine Timeout-Funktion implementiert, da das Zählen der notwendigen Inkrementalschritte zum Festhalten der Objekte genauso gut funktioniert und ich so auf bereits implementierte Funktionen zurückgreifen konnte. Im Allgemeinen fehlen zwei geforderte Regeln/Funktionen zum Ablauf: Erstens gibt es keinen Motor-Timeout und keine Pause-Taste, lediglich eine Abbruchtaste.

2. UML-Klassendiagramm

Obwohl es eigentlich nicht üblich ist, habe ich trotz Beziehungspfeile, die Attribute in den jeweiligen Klassen explizit genannt. Dies soll eine bessere Übersicht geben und ein leichteres Verständnis der Architektur bewirken.



3. Zustandsdiagramm für den mechanischen Bewegungsablauf des Industrieroboters



4. Nutzung/Steuerung

Diese Ausrichtung der Regale und des Roboters, wie auf dem Bild zu erkennen, ist die bekannte Umgebung der Software. Der Roboter befindet sich hier in Null-Position.
Ich bin von der geforderten Steuerung abgewichen. Startet man den Microcontroller, muss man Button-A drücken, um die Stromzufuhr zu erlauben bzw. die H-Brücke zu aktivieren.
Button-B startet die geforderte Routine. Die Routine überprüft von selbst, ob der Industrieroboter sich in Null-Position befindet und bringt ihn selbstständig dort hin. Möchte man die Routine abbrechen, muss die schwarze Reset-Taste des Controllers gedrückt werden. Dieser Knopf befördert den Roboter allerdings nicht in Null-Position. Dafür wird der Button-C verwendet oder die Routine neu gestartet, die dies autonom ausführt. Allerdings darf man dann nicht vergessen, die H-Brücke wieder zu aktivieren. Während der Routine wird auf dem Display die Zielposition und die aktuell sich bewegende Koordinate angegeben. Nach jeder beendeten Achsenbewegung wird die gesamte Position in der letzten Zeile aktualisiert.
Für die manuelle Steuerung werden die Buttons C,D und der Drehknopf verwendet.
Button-C setzt den Roboter von selbst in Null-Position. Button-D wählt einen der vier Motoren, welcher nun mit dem Drehknopf gesteuert werden kann. Das Drehen steuert die Motorrichtung. Befindet sich der Drehknopf auf Position 0, werden die Motoren gestoppt.

5. Software-Tests tabellarisch dokumentiert

Test	Erwartet	Ergebnis
Verwenden der gegebenen Koordinaten der Regale	Roboter erreicht Regale. Die Werte stimmen mit den Objektpositionen überein.	Koordinaten stimmen nicht. Wurden neu ermittelt und sind als Makros in der InduRoboter Klasse definiert.
Manueller Modus: Button-D wechselt den Motor (Modulo-Check)	Beim erneuten Drücken wechselt der Motor von 4 auf 1	Modulo Fehler. Display zeigte 0 statt 1.
MoveToPos(x)	Achse soll sich bis Wert x bewegen.	Achse hatte sich bis Wert (x-1) bewegt
MoveToPos(x)	Achse soll Position nur erhöhen wenn der Inkrementalschalter gedrückt wurde.	Achse hatte zu schnell die Position erhöht. Lösung: zusätzlicher bool (checkInc/Dec) in der If-Verzweigung in der Funktion inc()/dec() in der Klasse Position
Display-Anzeige-Test	Das aktualisieren der Positionen klappt ohne Anzeige Fehler	Printf()-Befehl bezog sich nicht immer auf die komplette Zeile wodurch alte

		Werte, wenn sie sehr lang waren (z.B. alle drei Positionen größer 9), nicht immer komplett überschrieben wurden und Anzeigefehler entstanden.
Test der vollständigen Routine (mindestens fünf mal[1])	Roboter führt Routine vollständig und fehlerfrei aus	Ergebnis genau wie erwartet.
Motor-Klasse	Ausführen von MoveLeft() und MoveRight() führt, dazu, dass der Motor sich bewegt.	Hat genau wie erwartet funktioniert.

[1]: Routine wurde häufiger durchlaufen, um zu überprüfen, ob die Bestimmung der Position durch Inkrementalschalter konsistent und präzise bleibt.

6. Vorgehensweise

Meine Vorgehensweise lässt sich grob in verschiedene Stufen/Kategorien unterteilen. Da ich bisher noch keine Vorkenntnisse mit Microcontrollern besitze und ich C++ erst in diesem Semester in zwei Modulen gelernt habe, musste ich mich erst mit der neuen Umgebung vertraut machen.(**1**) Ich begann damit einzelne Pins aus der Dokumentation im Programm zuzuweisen und lies zur Test-Kontrolle eine LED aufleuchten. Ich belegte alle notwendigen Pins im Config-Header und testete diese einzeln.(**2**) Im nächsten Schritt habe ich die Problemstellung abstrakt betrachtet und mit dem Software-Design begonnen. Hier folgte ich dem geforderten objektorientierten Ansatz und erstellte das UML-Klassendiagramm.(**3**)

Nach dem Entwurf der Software-Architektur begann ich mit der Implementierung des Motors, da man diesen ähnlich wie beim Test-Driven-Development sofort testen kann, weil die Klasse Motor direkt mit den Pins (in Form von cDevDigital-Objekten) für die Motoren arbeitet. Nachdem sich mein Verständnis bezüglich des Verhaltens der Embedded Systems Library (EmbSysLib) nun gefestigt hat, begann ich mit der Implementierung von der anderen Richtung.(**4**) Ich habe inkrementell vom Abstrakten/Komplexen zum Konkreten hin programmiert.

Also von der Klasse InduRoboter hin zur Klasse Achse und dann zur Klasse Position.

Durch diese Vorgehensweise konnte ich den Anforderungskatalog noch einmal überarbeiten und genau schauen, welche Funktionen der Roboter braucht, um seine Aufgabe zu erfüllen.

Während der Entwicklung der Software sind auch Probleme aufgetreten. Das Größte war vermutlich, dass ich in der getPos() Funktion der Klasse Achse das „&“ in der Festlegung des Rückgabewertes im Funktionskopf ausgelassen habe. Das führte dazu, dass ich das Attribut „counter“ nicht in der Funktion inc() verändern konnte und so die Funktion MoveToPos() zunächst nicht implementieren konnte. Da C++ für mich eine neue Programmiersprache ist, war mir nicht bewusst, dass es sich hierbei um einen Fehler handelt. Aus diesem Grund hat mich dieser Fehler ca. einen Tag gekostet.

Zusammenfassend:

(1): Hardware kennenlernen, Umgebung überblicken
(2): Software-Design (UML)
(3): Erste Implementierung (testgetrieben)
(4): Inkrementelles Entwickeln von abstrakt zu konkret

7. Fazit

Um meine gesammelte Erfahrung hiermit zusammen zu fassen. Ich habe sehr viel neues dazu gelernt, vor allem da ich als Informatikstudent bisher wenig mit Hardware konfrontiert wurde. Es war interessant zu sehen, wie man mechanische Objekte mit Hilfe von Software steuern kann. Tatsächlich wäre diese Richtung eine Option für meinen zukünftigen Werdegang.
Die Steuerungs-Software, die ich hier erstellt habe, erfüllt seine Aufgabe. Die Objekte, der ersten beiden Regale, werden auf das Endregal befördert und ineinander gesteckt. Allerdings bin ich an einigen Stellen von den Anforderungen abgewichen, wie zum Beispiel die Greiffunktion des Greifarmes implementiert werden sollte. Meine Greiffunktion funktioniert lediglich an Objekten, die dieselben Maße aufweisen, wie die hier Gegebenen. Außerdem habe ich kein Beobachter Design Pattern verwendet, da ich es als nicht notwendig empfand. Es wäre aber wahrscheinlich in einem Projekt von größerem Maße besser gewesen, dies anzuwenden, um zukünftige Wartungsarbeiten oder das Austauschen von Modulen zu erleichtern. Außerdem wäre es eventuell möglich gewesen, dass der Roboter nicht nach jedem Zwischenschritt wieder in die Null-Position gehen muss. Man hätte eventuell auch von der aktuellen Position aus weiter zählen können. Allerdings befürchtete ich hardwarebedingtes inkonsistentes Verhalten des Roboters. Außerdem wird durch meine Lösung funktionale Sicherheit gewährleistet für den Fall, dass mehr Regale hinzugefügt werden. Denn wenn der Roboter immer in seine Ausgangslage versetzt wird, kann er keine anderen Regale treffen, wie als wenn er direkt zum nächsten Regal wandert. Möglicherweise wäre aber ein Kompromiss gewesen, Radius und Höhe in die Nullposition zu versetzen, aber den Winkel nicht. So könnte man eventuell eine höhere Effizienz gewährleisten.
Trotz dieser genannten Punkte bin ich mit meinem Ergebnis und der gewonnen Erfahrung sehr zufrieden und würde mein Ergebnis als gelungen bezeichnen.