HOCHSCHULE FÜR TECHNIK STUTTGART

Dokumentation im Fach Informationslogistische Prozesse

Sensorik 2

Edric Jerruce Azangue Dongmo, Sibel Karaca, Saskia Brumm

22.06.2017

Entwicklung eines Programms zur Steuerung des Robotino zum Umfahren von Hindernissen

Inhalt

ABBILDUNGSVERZEICHNIS **2**

1. ÜBERSICHT **3**

2. AUFGABENSTELLUNG **3**

3. LÖSUNGSANSATZ **3**

3.1 Alternative mit JTS **3**

3.2 Alternative ohne JTS **4**

4. UMSETZUNG **5**

4.1 Vorgehensweise **5**

4.2 Arbeitseinteilung, Verwendete Werkzeuge und Methoden **7**

5. ZUSAMMENFASSUNG UND AUSBLICK **9**

6. Verwendete Literatur **10**

Abbildungsverzeichnis

Abbildung 1: Test mit der Sim-Demo 5
Abbildung 2Robotino 6
Abbildung 3Winkelsberechnungen 7
Abbildung 4 Network in Github 8

1. Übersicht

Unser Team besteht aus Edric Jerruce Azangue Dongmo, Sibel Karaca und Saskia Brumm. Wir sind Studenten des Studiengangs Informationslogistik. Im Rahmen des 6. Fachsemesters belegen wir das Modul Informationslogistische Prozesse, welches sich unter anderem mit autonomen Transportsystemen beschäftigt.

2. Aufgabenstellung

Das Ziel unserer Studienarbeit besteht darin, ein autonomes Transportsystem, den Robotino, kollisionsfrei zu steuern. Dafür sollen wir die Umgebung anhand von Sensoren abtasten und den Weg, welcher bereits gefahren wurde, nach dem erfassten Objekt fortsetzen.

Dabei sind unsere Erwartungen, Wissen zu erlangen, wie wir den Robotino mithilfe der Sensoren steuern können und was noch benötigt wird, um den Robotino fahren zu lassen. Zudem haben wir uns anhand der Aufgabenstellung vorgestellt, dass die Befehle für die gängigen Richtungen (vorn, zurück, links, rechts) bereits vordefiniert sind, sodass wir uns nur um das Wie kümmern und nur einfache Methoden für das Ausweichen schreiben müssen, um die Aufgabe zu lösen.

3. Lösungsansatz

Es gibt unterschiedliche Wege, die Aufgabenstellung zu lösen. Nachfolgend werden 2 Lösungsansätze erläutert.

3.1 Alternative mit JTS

Bei der Variante, bei welcher JTS verwendet wird, werden Polygone und ein Koordinatensystem benötigt. Es ist dafür nötig zu wissen, an welcher Stelle genau das Hindernis steht, um es in das Koordinatensystem mit einzubringen. Die Umgebung wird mit den zur Verfügung stehenden Sensoren abgetastet und erkannte Objekte können so in dem Raum, für welchen das Koordinatensystem angelegt wurde, lokalisiert werden. Das autonome Transportsystem dreht dann in einem Bestimmten Winkel ab. Der Punkt zum Fortführen des Weges nach der Erkennung kann durch das Koordina-

tensystem und den Polygonen sehr leicht bestimmt werden. Der Nachteil dieser Variante ist, dass alles genau vordefiniert sein muss. Ein Vorteil ist aber dass durch die Definition bekannt ist, wie groß das gerade entdeckte Objekt ist.

3.2 Alternative ohne JTS

Die Alternative ohne JTS verwendet kein Koordinatensystem. Somit ist es auch nicht möglich einzelne Objekte vor zu definieren. Die Hindernisse werden während der Fahrt erkannt, wodurch jedes Hindernis erfasst werden kann. Sofern ein Hindernis erkannt wurde, wird ein Winkel zum Drehen berechnet, anhand dessen eine Kollision vermieden wird. Das umfahren eines Hindernisses geschieht auf der Grundlage der berechneten Winkel. Der eingeschlagene Weg wird wieder eingenommen durch entgegengesetzte Abfolge der Anweisungen (Winkel), die für das Ausweichen verantwortlich waren.

Wir haben uns für die Variante ohne die Verwendung von der Bibliothek JTS entschieden. Grundgedanke unserer Lösung ist das Erkennen aller Objekte und das Navigieren des Robotino mit wenigen Anweisungen. So soll es möglich sein, beim Erkennen eines Gegenstandes, den Winkel, mit welchem sich der Robotino drehen muss, so zu berechnen, dass nur eine Drehung nötig ist, um dem Gegenstand auszuweichen. Nach dieser Berechnung wird dann solange geradeaus gefahren, bis kein Gegenstand mehr in der Reichweite ist und anschließend auf den alten Weg zurückgekehrt. Dies ermöglicht auch das Erkennen von Wänden oder nichtrechteckigen Objekten.

4. Umsetzung

Hier in diesem Kapitel geht es um die ganzen Arbeitsschritte, Methoden, Werkzeuge und Prozesse, die wir für unsere Aufgabe brauchen, um es umzusetzen.

4.1 Vorgehensweise

Wir haben uns als erstes mit dem Robotino vertraut machen müssen. Dafür haben wir nach Beispielprogrammen auf der Internetseite von Festo recherchiert. Es stehen 3 Programme zur Verfügung, von welchen wir das „Follow Wall“-Projekt in Eclipse geöffnet und analysiert haben. Anschließend haben wir ein Projekt in Netbeans angelegt und die benötigte Bibliothek API2 zum Kommunizieren mit dem Robotino kopiert und dort hinzugefügt.

Um den Robotino in der virtuellen Welt zu steuern, haben wir das Programm Sim Demo heruntergeladen, mit dessen Hilfe wir das Beispielprogramm und unseren eigenen Code getestet haben.

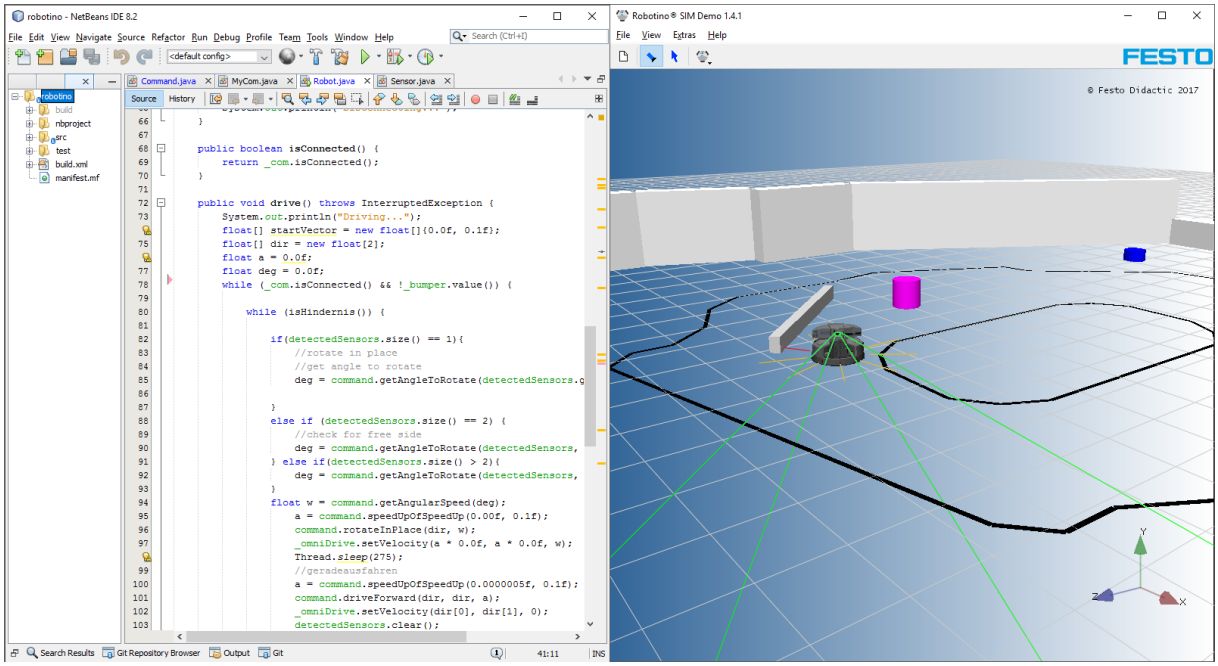

Abbildung 1 : Test mit der Sim-Demo

Wir haben zunächst die für uns wichtigen Codebausteine herausgefiltert und uns dann überlegt, welche Klassen benötigt werden. Dann haben wir für die Verbindung

zum Robotino und den grundlegenden Parametern, welche wir für die Steuerung benötigen, die Klasse Robot.java angelegt. Darin haben wir dann die wichtigsten Codeteile zusammengefügt, die Voraussetzung für das Fahren sind. Außerdem haben wir die Klasse MyCom.java in eine externe Datei ausgelagert.

Anschließend haben wir uns dann um die Umgebungserfassung gekümmert, wozu die Auswertung der Distanzsensoren und der Stoßleiste nötig ist. Da die Stoßleiste nur ein zu überprüfender Parameter ist, haben wir diesen als feste Variable angelegt. Für die Distanzsensoren haben wir dann überlegt, wie wir diese dauerhaft gleichzeitig auswerten können. Dafür haben wir die Klasse Sensor.java angelegt und von Thread abgeleitet, um eine parallele Auswertung zu ermöglichen. Da der Robotino 9 Distanzsensoren besitzt, haben wir eine ArrayList und für jeden Sensor darin ein Objekt angelegt. Im nächsten Schritt haben wir das Verhalten der Sensoren getestet, um eine korrekte Auswertung dieser zu ermöglichen. Danach haben wir die Sensorauswertung mithilfe einer Fallunterscheidung in unser Programm integriert.

Abbildung 2 : Robotino

Im nächsten Schritt haben wir uns die entsprechenden Berechnungen für die Drehwinkelbestimmung überlegt. Dafür recherchierten wir die Dreiecksberechnungen. Um die Berechnungen für die Drehwinkel und das Fahrkommando für geradeaus an einer Stelle zu sammeln, haben wir die Klasse Command.java angelegt. Die Berechnung der Drehwinkel muss je nach erfassten Sensoren angepasst werden. Dafür ist es nötig das Programm immer wieder zu testen und zu schauen, ob der Robotino das umsetzt, was ihm per Code mitgeteilt wird.

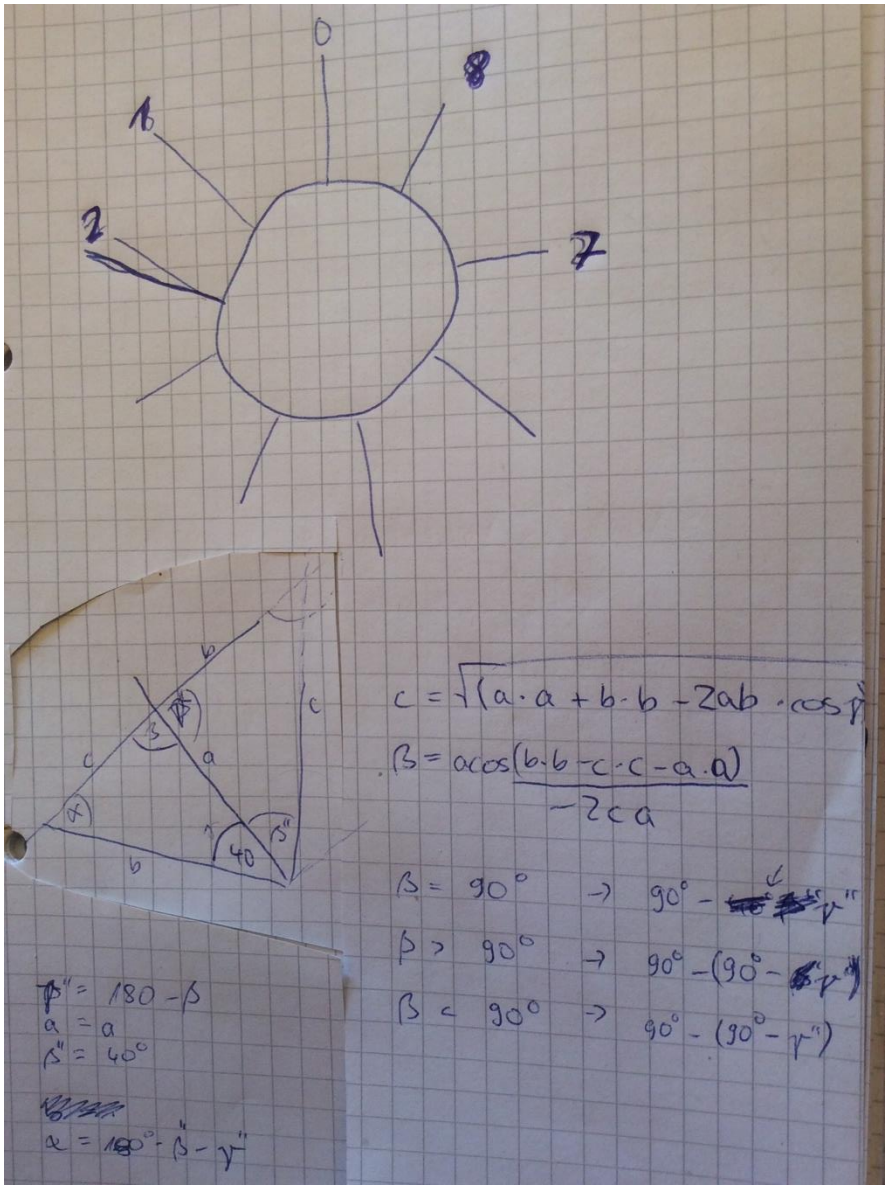

Abbildung 3 : Winkel Berechnungen

Nach erfolgreich erzeugten Drehungen haben wir den Roboter dann geradeaus fahren lassen, sodass der Robotino sich vom Hindernis entfernt.

4.2 Arbeitseinteilung, Verwendete Werkzeuge und Methoden

Anfangs haben wir zusammen unsere Vorgehensweise geplant und durchgesprochen. Dabei haben sich dann die Aufgaben für jedes Teammitglied herauskristallisiert. Je nach Kenntnisstand der Programmierung und Einfallsreichtum die Problem-

stellung zu lösen, hat jeder die entsprechenden Aufgaben bekommen. Während der Codeerstellung haben wir mit NetBeans und Github gearbeitet. Wir haben dadurch eine gemeinsame Grundlage für die Entwicklung gehabt, sodass nachvollzogen werden kann, wer welche Aufgabenbereiche übernimmt. Grob eingeteilt gibt es den Bereich der Winkelberechnung und den der Steuerung, der von zwei Teammitgliedern bearbeitet wird. Bei der Bearbeitung hat jedes Teammitglied eine eigene Branch bei Github bekommen, welche in gewissen Abständen in die Hauptbranch zusammengeführt werden.

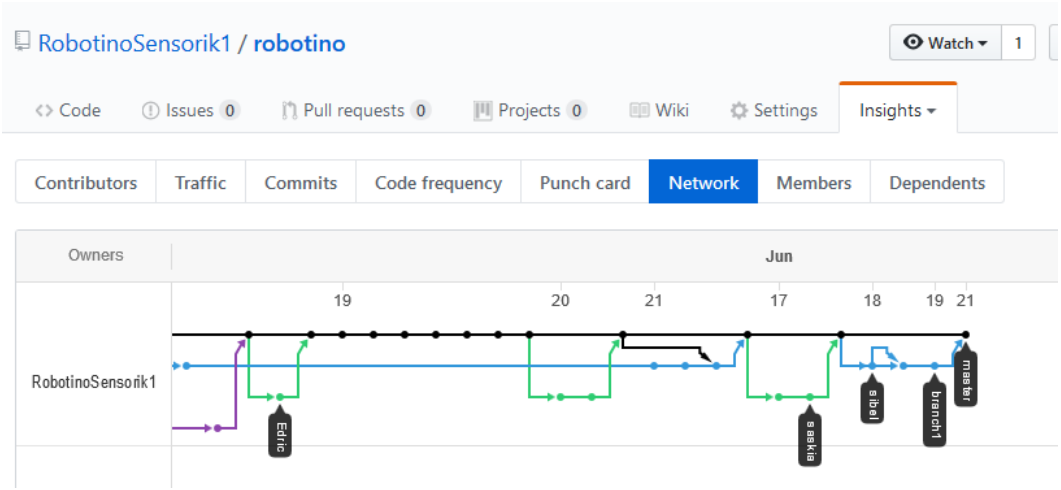

Abbildung 4 : Network in Github

5. Zusammenfassung und Ausblick

Bezüglich der Aufgabenstellung, einen Robotino Hindernisse ausweichen zu lassen und den angefangenen Weg fortzusetzen, haben wir den Großteil umsetzen können. Das autonome Transportsystem erkennt Hindernisse und weicht denen aus. Das Fortsetzen des Weges funktioniert leider nicht wie geplant. Ebenso ist die Steuerung ungenau. Diese beiden Punkte hätten vermieden werden können, wenn wir dafür doch mit der Bibliothek JTS gearbeitet hätten. Problematisch war das Einarbeiten in Github, was sehr viel Zeit in Anspruch genommen hat. Da wir damit anfangs nicht zurechtkamen und nicht wussten, wie wir ein gemeinsames Verzeichnis erstellen, um an dem Code gemeinsam zu arbeiten, mussten wir ein paar Mal von neuem anfangen, bis alle Teammitglieder einen Zugriff auf das Verzeichnis hatten.

Wir haben Erfahrung in der Teamarbeit sammeln können, sowie die Erkenntnis gewonnen, dass es nicht ganz einfach ist ein autonomes Transportsystem zu steuern, da man sehr viele Dinge berücksichtigen muss. Außerdem hatten wir die Möglichkeit neue Werkzeuge einzusetzen (Github) und dadurch gelernt Quellcode im Team zu entwerfen. Die Erwartungen, welche wir zu Beginn hatten, wurden teilweise erfüllt, wie zum Beispiel Wissen zu erlangen, wie man den Robotino steuern kann.

Allgemein sind wir zufrieden mit unserem Ergebnis, welches allerdings noch ausbaufähig ist. So könnte man beispielsweise die Auswertung der Sensoren noch genauer machen, oder auch andere Sensoren zur besseren Orientierung hinzuziehen.

6. Verwendete Literatur

http://www.festo-didactic.com/de-de/service/robotino/hardware/sensoren/distanzsensoren/?fbid=ZGUuZGUuNTQ0LjEzLjM0LjE0NTI

http://wiki.openrobotino.org/index.php?title=Building_the_examples_with_Eclipse

http://www.mathematik-wissen.de/kongruenzsaetze.htm