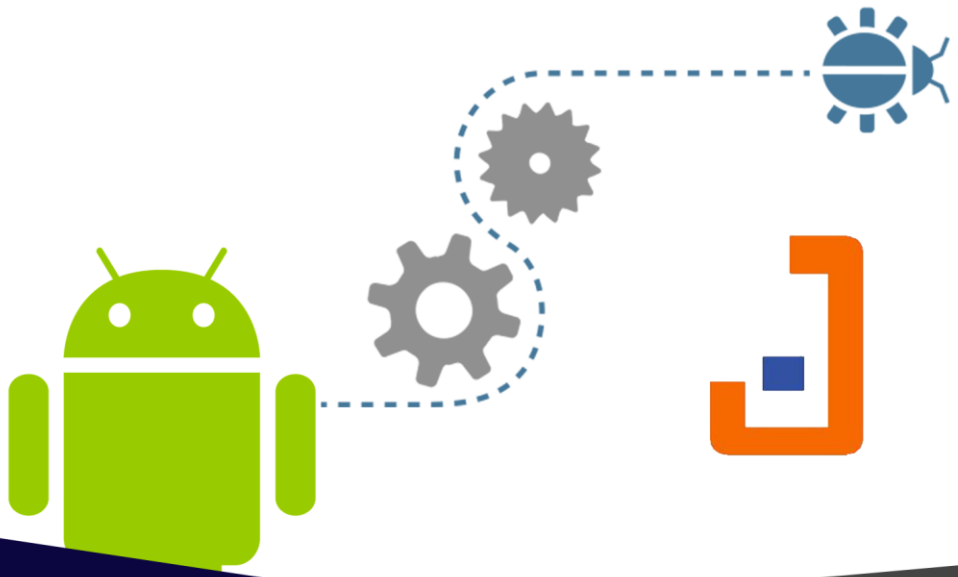

Jugend forscht 2018

Der Einkaufs-Roboter

Ein Projekt von Vincent Schmandt und Fabian Specht

Kategorie Informatik
Projektnummer 149784

Inhaltsverzeichnis

Abbildungsverzeichnis

Einführung

Problemstellung und Umstände

Der demographische Wandel unserer Gesellschaft ist gekennzeichnet von sinkenden Geburtenzahlen, erhöhter Lebenserwartung und Auflösung von traditionellen Familienstrukturen. Ältere und körperlich eingeschränkte Menschen sind vermehrt auf sich alleine gestellt und auf Unterstützung angewiesen.

Dabei ist das Transportieren schwerer Gegenstände, wie zum Beispiel Einkäufe, oftmals ein Problem von großer Bedeutung. Aufgrund dieser Umstände können vor allem alleinlebende, körperlich eingeschränkte Menschen sich nicht mehr selbst versorgen und sind auf Unterstützung, zum Beispiel durch Haushaltshilfen, angewiesen, welche viel Geld kosten, das selten aufgebracht werden kann.

Auch moderne Online-Dienstleistungen in Form von Lieferservices entsprechen häufig nicht den Bedürfnissen älterer Menschen, da diese sich in den wenigsten Fällen ausreichend mit Computern auskennen oder keinen Internetzugang besitzen.

Unsere Lösung für diese Problematik

Um das Problem, welches das Tragen von schweren Einkäufen oder Ähnlichem darstellt, zu lösen, entschieden wir uns, einen Roboter zu konstruieren, welcher diese Aufgabe übernehmen kann. Dieser sollte, um für die meist technikunerfahrene ältere Generation leicht handhabbar zu sein, autonom und somit ohne weitere manuelle Steuerung seinem Besitzer folgen und sonst nur durch Sprachbefehle gesteuert werden. Außerdem muss der Roboter sich den Gegebenheiten, wie zum Beispiel variablen Geschwindigkeiten unterschiedlicher Menschen, anpassen können. Dazu kommen noch einige weitere Grundvoraussetzungen, die es zu beachten gibt, wie beispielsweise eine geeignete Höhe der Tragefläche, um ein anstrengendes Bücken des Besitzers zu vermeiden.

Art und Weise des Vorgehens

Um alle Erforderlichkeiten, die für einen guten Roboter nötig sind, zu erfüllen, planten wir eine ganze Reihe von verschiedenen Modellen, wobei die verwendete Technik und Software sich, auf Basis des jeweils vorherigen Modells, immer weiter verbessern sollte. Durch dieses System setzten die ersten Prototypen noch auf sehr einfache Technik sowie Mechanik, und es fehlten einige der oben genannten Funktionen.

Dabei testeten wir verschiedene Arten der Fortbewegung und der Positionsbestimmung und nutzten verschiedene Programmiersprachen und Bibliotheken, bis wir uns für eine finale Methodik entschieden.

Mit der Zeit sollten außerdem noch einige weitere, zusätzliche Funktionen hinzukommen, um den Nutzern des Roboters den Alltag zusätzlich zu erleichtern.

Nach dem Bau eines Prototyps erstellten wir ein passendes Programm, immer an die Fähigkeiten des gegenwärtigen Roboters angepasst. Dabei teilten wir das gesamte Projekt in mehrere Abschnitte, die Prototypen, ein, wobei sich jeder davon mit einem speziellen Problem, zum Bespiel dem Lokalisieren des Besitzers, beschäftigte. Dies sorgte sowohl für einen besseren Überblick als auch für die Möglichkeit, sich genauer mit einzelnen Problemen zu beschäftigen. Eine solche Vorgehensweise braucht zwar viel Zeit, da jedes Modell einzeln gebaut, programmiert und getestet werden muss, überzeugt jedoch mit guten Ergebnissen und einer klaren und effizienten Strukturierung.

Umsetzung

Das Entwickeln eines Roboters

Bei der Entwicklung der Prototypen gingen wir wie oben beschrieben vor (siehe Art und Weise des Vorgehens). Das bedeutet, dass wir insgesamt sechs Prototypen entwickelten, um uns einem optimalen Roboter schrittweise anzunähern. Beim Material setzten wir aufgrund der hohen Flexibilität und akzeptabler Stabilität auf die „LEGO-Technic“ Bauteile.

Prototyp 1: Fortbewegung

Unser erstes Konstrukt war nicht viel mehr als ein simples Fahrgestell. Es konnte durch seine zwei gefederten Ketten, welche ohne Getriebe direkt an je einen Motor angebunden waren, nur sehr langsam und nur geradeaus fahren. Daran, diesen zu programmieren, war nicht einmal zu denken, da noch kein Computer verbaut war, welchen man hätte programmieren können.

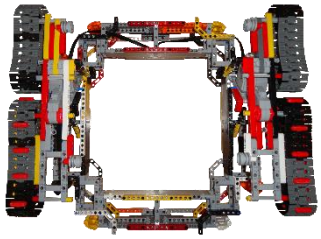

Abb. 1 - Der 1. Prototyp von Unten

Prototyp 2: Fahrwerksverbesserung & Erste Programmierung

Bei unserem zweiten Modell tauschten wir die Ketten gegen Räder aus, bauten einen EV3 (siehe Hardware und Software des EV3) ein und fügten ein pneumatisches Grundsystem (elektronische Pumpe, Drucklufttank, einige Kolben) hinzu, mit dem wir weitere Funktionen, wie eine verschließbare Klappe für den Einkaufskorb, umsetzen wollten. Dieser Prototyp konnte dann mithilfe eines kleinen, in der grafischen LEGO Umgebung erstellten Programmes, simple Bewegungen ausführen. Da allerdings noch keine Sensoren verbaut waren, war er weder in der Lage Objekten auszuweichen noch einer Person zu folgen.

Prototyp 3: Treppen steigen

Bei dem dritten Prototyp unseres Projekts versuchten wir erstmals den Roboter Treppen steigen zu lassen. Wir setzten dabei auf ein System, bei dem sich der Roboter durch vier an den Seiten angebrachte Arme auf Höhe einer Treppenstufe stemmt. Dazu konnte dieser alle Arme nach oben und nach unten schwenken. Zum Fahren waren alle vier Arme nach unten geklappt, sodass die angebrachten Räder auf dem Boden auflagen. Das dafür entwickelte Programm sollte dafür sorgen, dass der Roboter zuerst an eine Stufe heranfährt, anschließend seine vorderen Arme nach oben schwenkt und auf dieser ablegt. Danach sollte sich derselbe Vorgang für die beiden hinteren Arme wiederholen. Dieses Modell scheiterte letztendlich an der mangelnden Stabilität des Baumaterials und der zu geringen Kraft der Motoren, welche nötig gewesen wäre, um den Roboter anzuheben.

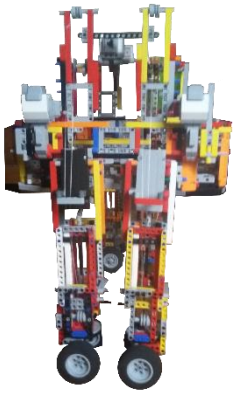

Abb. 2 - Der 3. Prototyp von der Seite

Prototyp 4: Tragflächenhöhe & „Person Tracking“

Nachdem wir das Treppensteigen nun nicht mehr fortführten, beschlossen wir uns bei Modell 4 einem Problem zu widmen, auf das wir zuvor noch nicht geachtet hatten - die Höhe der Tragfläche. Wir entschieden uns für eine Höhe von 80 cm, damit der Besitzer des Roboters sich nicht bücken muss, um seinen Einkauf zu entnehmen. Dazu entwickelten wir einen Roboter, dessen Fahrgestell dem Prinzip des Gestells des zweiten Prototyps glich, jedoch mit der Veränderung, dass diesmal ein großer Aufbau darauf gesetzt war, auf dessen höchsten Punkt sich der Korb für die Einkäufe befand. Außerdem versuchten wir das erste Mal, die Position des Besitzers zu bestimmen. Hierbei setzten wir auf einen Lego Infrarot-Sensor, welcher am Roboter befestigt war und eine dazugehörige Fernbedienung, die, beim Besitzer getragen, durchgehend Signale senden sollte, anhand derer die ungefähre Richtung und Distanz zwischen Sensor und Sender ermittelt werden kann. Da die dazu nötige Programmierung bereits

Abb. 3 - Der 4. Prototyp von Vorne

die Möglichkeiten der Lego-Software überschritten hätte, entschieden wir uns, auf dem Roboter die auf Linux basierende JavaVM „LeJOS“ zu installieren, um nun in Java weiter arbeiten zu können. Mit der Programmierung des Roboters agierte dieser einigermaßen zufriedenstellend, verlor jedoch oft das Signal der Fernbedienung oder ortete sie an einer vollständig falschen Position. Außerdem war, da weder ein Getriebe noch eine Lenkung eingebaut waren, das Fahrverhalten, trotz exakter Software, sehr ungenau.

```
int frontDistance = 0;
int Bearing = 0;
int Distance = -2;

while(true){

    frontDistance = USSFront.getDistance();
    Distance = IRS.getValue().distance;
    Bearing = IRS.getValue().bearing;

    if(Bearing >= 5 && Distance != -2 /*device lost */ && frontDistance > 10){

        motA.setSpeed(SpeedX);
        motC.setSpeed(Speed);
        motD.setSpeed(SpeedS);
        motB.setSpeed(SpeedS2);

        motA.forward();
        motB.forward();
        motC.backward();
        motD.backward();
    }
    else if(Bearing <= -5 && Distance != -2 /*device lost */ && frontDistance > 10){

        motB.setSpeed(SpeedX);
        motD.setSpeed(Speed);
        motC.setSpeed(SpeedS);
        motA.setSpeed(SpeedS2);

        motA.forward();
        motB.forward();
        motC.backward();
        motD.backward();
    }
    else if(Bearing <= 5 && Bearing >= -5 && Distance != -2 /*device lost */ && frontDistance > 10){

        motA.setSpeed(SpeedX);
        motB.setSpeed(SpeedX);
        motC.setSpeed(SpeedX);
        motD.setSpeed(SpeedX);

        motA.forward();
        motB.forward();
        motC.backward();
        motD.backward();
    }
```

Abb. 4 - Ausschnitt aus der Programmierung des 4. Prototyps: Endlosschleife (Hauptteil)

Prototyp 5: Hinzufügen von Sensoren & Verbesserung des „Person Tracking“

Um die Genauigkeit, welche den 4. Prototypen zum Scheitern gebracht hatte, zu verbessern, entschieden wir uns, das seit dem 2. Prototypen nahezu gleich gebliebene Fahrgerüst vollständig neu zu konstruieren und eine Lenkung einzubauen. Außerdem brachten wir nun zwei Infrarot-Sensoren an, welche beide die Distanz zur Fernbedienung bestimmen sollten. Anhand der beiden Distanzen und der Distanz der Sensoren zum Mittelpunkt des Roboters lassen sich mithilfe der Vektorrechnung zwei Schnittpunkte berechnen, wobei der im positiven Bereich liegende Schnittpunkt der Position der Person in einem Koordinatensystem entspricht. Dazu verwendeten wir ein Programm auf Basis der folgenden Terme:

$$c = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2}$$

$$x = \frac{a^2 + c^2 - b^2}{2c} \qquad y = \sqrt{a^2 - x^2}$$

$$Q_{1,x} = A_x + x \cdot \frac{B_x - A_x}{c} - y \cdot \frac{B_y - A_y}{c} \qquad \text{und} \qquad Q_{1,y} = A_y + x \cdot \frac{B_y - A_y}{c} + y \cdot \frac{B_x - A_x}{c}$$

wobei:

a = Radius von Kreis A

b = Radius von Kreis B

c = Abstand der Kreiszentren

x = X-Abstand der beiden Schnittpunkte zum Ursprung

y = Y-Abstand der beiden Schnittpunkte von der X-Achse

Dieses sah dann aus wie folgt:

```
//return0 wird zurück gegeben wenn
//sich die Kreise nicht schneiden
double[] return0 = {};
double AB0 = B[0] - A[0];
double AB1 = B[1] - A[1];
double c = Math.sqrt(AB0 * AB0 + AB1 * AB1);

if(c == 0) {
    return return0;
}

double x = (a*a + c*c - b*b) / (2*c);
double y = a*a - x*x;

if(y < 0) {
    return return0;
} else if (y > 0) {
    y = Math.sqrt(y);
}

double ex0 = AB0 / c;
double ex1 = AB1 / c;
double ey0 = -ex1;
double ey1 = ex0;
double Q1x = A[0] + x * ex0;
double Q1y = A[1] + x * ex1;

if(y == 0) {
    //return1 wird zurück gegeben wenn
    //sich die Kreise an endlos vielen Punkten schneiden
    //bzw. die Distanz zwischen den Kreisezentren 0 beträgt
    double[] return1 = {Q1x, Q1y};
    return return1;
}

double Q2x = Q1x - y * ey0;
double Q2y = Q1y - y * ey1;
Q1x += y * ey0;
Q1y += y * ey1;

//return2 wird zurück gegeben wenn
//sich die Kreise an zwei Punkten schneiden
double[] return2 = {Q1x, Q1y, Q2x, Q2y};
return return2;
```

Abb. 5 - Programm zum Berechnen der Schnittpunkte zweier Kreise

Jedoch stellte sich nach einigen Testläufen mit der neuen Software heraus, dass auch diese aufgrund der zu hohen Ungenauigkeit noch nicht geeignet war, um final verwendet zu werden.

Prototyp 6: Erweiterte Sensorik & Anpassung der Lenkung

Bei unserem sechsten und zum Zeitpunkt des Wettbewerbs aktuellen Prototyp veränderten wir nahezu alles. Die Infrarot-Sensoren wurden von einem Microsoft Kinect Sensor der zweiten Generation abgelöst (siehe Einbindung des Microsoft Kinect Sensors v2). Weiterhin wurde der Aufbau des Roboters massiv verändert, da nun ein Laptop auf ihm befestigt werden musste, um die Daten des Sensors in Echtzeit auswerten zu können. Des Weiteren setzen wir nun auf ein vollständig neues Lenksystem, bei dem die hinteren Räder sich kontinuierlich in einer der Distanz zum Besitzer angepassten Geschwindigkeit drehen, während die vorderen Motoren der Lenkung dienen.

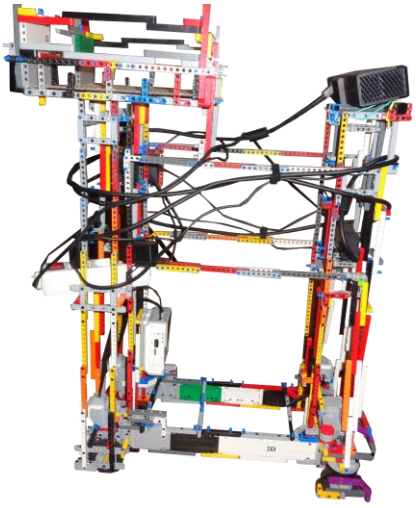

Abb. 6 - Der 6. Prototyp von der Seite

Der verwendete Sensor ist in der Lage, durch Infrarotstrahlen die vor ihm liegende Umgebung zu scannen und verfügt außerdem über eine Kamera. Mit dem Sensor ist es möglich, eine Person zu verfolgen, indem die Bilddaten nach Personen durchsucht werden (Genaueres: siehe ebenfalls Einbindung des Microsoft Kinect Sensors v2). Er ist über ein USB-Kabel mit dem angebrachten Laptop verbunden. Durch eine dauerhafte Datenübertragung via WLAN steht der Laptop mit den anderen Geräten in Kontakt.

Hardware und Software des EV3

Die Hardware

Der EV3, ein kleiner Computer hergestellt von LEGO, stellte sich für unser Projekt als durchaus passend heraus. Mit seinem Set verfügt der Besitzer über eine vielfältige Auswahl von sowohl Motoren, welche wir zu der Fortbewegung des Roboters nutzten, als auch Sensoren, welche nur bei Prototyp 1-5 eine Rolle spielten. Durch diese Komponenten war eine gute Grundlage für dieses Projekt geschaffen.

Die Software

Das Ansprechen der Motoren und Sensoren sowie ein sicherer Datenaustausch zwischen dem EV3 und weiteren Geräten wurden möglich, indem wir die jeweiligen Programme in der Programmiersprache Java schrieben. Zwar war schon eine vorgefertigte, graphische Oberfläche zu der Programmierung speziell für EV3s vorhanden, jedoch ist diese nicht auf größere Projekte mit mehreren verbundenen Geräten ausgelegt, weshalb wir damit nicht arbeiten konnten. Um geschriebenen Code auch verarbeiten zu können, wurde ein neues Betriebssystem namens „LeJOS“ auf dem EV3 installiert. Dabei handelt es sich um eine „Java Virtual Machine“, welche unter Linux agiert und extra für den EV3 vorgesehen ist.

Die implementierte Bibliothek „LeJOS EV3“ fügt die nötigen Funktionen und Klassen in Java hinzu, welche beispielsweise das Ansteuern der Motoren ermöglichen.

Das Programm des EV3s besteht darin, die Position des Besitzers auf der X-Achse sowie der Z-Achse zu empfangen und diesem zu folgen. Außerdem muss er auf Befehle, welche er vom Smartphone erhält, reagieren. Die Daten des Sensors erhält er via WLAN, während er die Daten des Handys über Bluetooth empfängt.

Die Applikation

Ein Grundkonzept unseres Projekts ist die einfache Handhabung des Roboters auch für Menschen, welche sich nicht ausreichend mit Computern auskennen. Daher entwickelten wir eine Applikation für Smartphones, welche Sprachbefehle erkennt und an den Roboter weitergibt. Dieser sollte dem Befehl entsprechend reagieren. Dazu nutzten wir die im Android System integrierte Sprachverarbeitung und ließen diese nach einigen „Hotwords“ suchen. Hier beispielsweise nach dem Wort „Stopp“:

```
//Stop
if (results.get(i).toLowerCase().contains("Stop".toLowerCase())) {
    found = true;
    try {
        MainActivity.onSend("stop");
    } catch (IOException e) {
        e.printStackTrace();
    }
    Toast.makeText(BackgroundRecognizerService.this, "Roboter wird gestoppt...", Toast.LENGTH_LONG).show();
}
```

Abb. 7 - Ausschnitt aus der Sprachverarbeitung der Applikation

Verbindung Smartphone und EV3

Um die erkannten Begriffe an den EV3 weiterzugeben, richteten wir eine Verbindung via Bluetooth zwischen dem EV3 und dem Handy ein, welche jedes Mal, wenn die Applikation gestartet wird, neu initialisiert wird. Bisher zur Steuerung des Roboters nutzbare Begriffe sind:

- „Stopp“
  - ⇨ Der Roboter wird gestoppt
- „Weiter“
  - ⇨ Der Roboter fährt weiter / los

Die Befehle zum Steuern des Roboters sind auch durch Buttons auf der Startseite der Applikation ausführbar.

Weitere Funktionen der Applikation

Nachdem nun ohnehin eine Applikation auf dem Smartphone des Benutzers installiert werden musste, beschlossen wir diese mit weiteren Funktionen auszustatten.

Mit der Spracherkennung als Grundlage integrierten wir eine Einkaufsliste, welche vollständig durch Sprache gesteuert und bei Bedarf vorgelesen werden kann. Die Sprachsteuerung inkludiert zu diesem Zeitpunkt das Hinzufügen und Entfernen von Produkten, das Leeren der Liste und das vollständige Vorlesen der Liste. Diese Funktionen sind ebenfalls über die graphische Oberfläche der App verfügbar.

```
try {
    server = new ServerSocket(port);
    client = server.accept();
    DataInputStream stream = new DataInputStream(client.getInputStream());
    String message = "";
    WlanConnected = true;
    while (true) {
        message = "";
        message = stream.readUTF();
        if (message != null && !message.equals("Nix")) {
            this.Command = message;
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

Abb. 8 - Programmfragment zum Empfangen von Zeichenketten über einen Socket

Ebenfalls auf Basis der Spracherkennung entwickelten wir eine Notruffunktion, welche, nachdem sie in den Einstellungen in der App konfiguriert wurde, bei einer dabei festgelegten Phrase eine SMS an eine ebenfalls dort definierte Nummer sendet.

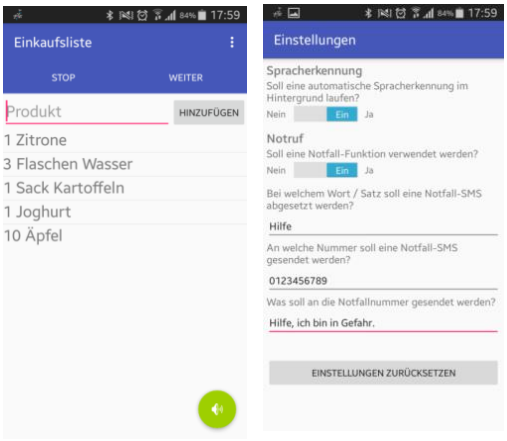

Abb. 9 – Hauptmenü und Einstellungen der Applikation

Einbindung des Microsoft Kinect Sensors v2

Hardware

Die hier verwendete zweite Version des Kinect – Sensors von Microsoft verfügt über einige Sensoren, welche gut zum Erkennen von Objekten in der näheren Umgebung eingesetzt werden können.

Dazu zählen:

- ein Tiefensensor (Time-Of-Flight)
    - ⇨ Dieser Sensor funktioniert auf Grundlage der von ihm gesendeten Infrarotstrahlung: Anhand der Zeit, die die Strahlung benötigt, um zum Sensor zurück zu gelangen, erstellt der Computer eine dreidimensionale Karte.
- Eine Kamera
    - ⇨ Die 1080p Farbkamera sendet in einer Geschwindigkeit von 30Hz Daten an den Computer
- Mehrere Infrarot Emitter
    - ⇨ Diese Komponente ermöglicht es dem Sensor, auch bei schlechter Beleuchtung gute Bilder von der Umgebung aufzunehmen.

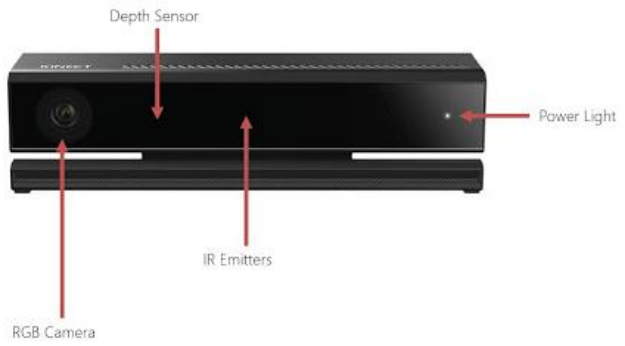

Abb. 10 - Abbildung des Microsoft Kinect Sensors v2 (www.physio-pedia.com)

Software

Damit der Roboter einer Person folgen kann, muss er diese erkennen. Da der Kinect-Sensor ursprünglich dazu entwickelt wurde, Spiele mit Gesten, welche von ihm erkannt werden, zu steuern, verfügt das von Microsoft zur Verfügung gestellte Kinect-SDK („Software-Developement-Kit“) über eine Funktionalität zum Erkennen und Verfolgen von Personen. Weil der Sensor in einem solchen Szenario allerdings fest montiert (z.B. über dem Fernseher) ist, basiert diese Funktion auf dem Erkennen und Auswerten von Bewegungen zwischen zwei Frames (Phasenbildern) und funktioniert daher nicht, sobald der Sensor bewegt wird, so wie in unserem Anwendungszweck.

Aus diesem Grund beschäftigten wir uns einen Großteil der Zeit damit, eine Software zu entwickeln, welche dieselben Ergebnisse zur Position einer Person liefert, allerdings nicht auf Bewegungserkennung beruht. Im Folgenden stellen wir die zwei wichtigsten Versionen vor, wobei die zweite Version die final verwendete Version darstellt.

Der Source-Code aller Programme ist unter https://github.com/vincentscode/jugend-forscht/ zu finden.

Version 1

Die erste Version unserer Software gliedert sich in fünf wichtige Teile.

Im ersten Teil, dem Setup, werden die verschiedenen Datenströme des Sensors aktiviert und dieser initialisiert. Des Weiteren wird eine Verbindung mit der App auf dem Smartphone hergestellt sowie die die Bildverarbeitungsklasse initialisiert.

```
kinect.enableBodyTrackImg(true);
kinect.enableColorChannel(true);
kinect.enableColorImg(true);
kinect.enableCoordinateMapperRGBDepth(true);
kinect.enableDepthImg(true);
kinect.enableDepthMaskImg(true);
kinect.enableInfraredImg(true);
kinect.enablePointCloud(true);
kinect.enablePointCloud(true);
kinect.enablePointCloud(true);
kinect.enableSkeleton(true);
kinect.enableSkeleton3dMap(true);
kinect.activateRawDepth(true);
kinect.activateRawColor(true);
kinect.enableLongExposureInfraredImg(true);

kinect.init();
```

```
public static void connect(String address, int port) {
    try {
        client = new Socket(address, port);
        dataOutputStream = new DataOutputStream(client.getOutputStream());
        dataInputStream = new DataInputStream(client.getInputStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
}



ImageRecognizer.init(matchRatio, minMatches, imageCount);
```

Abb. 11 - Ausschnitt aus der ersten Version der Kinect-Software: Initialisierung

Zu Beginn des zweiten Teils wird der Besitzer des Roboters dazu aufgefordert, sich langsam im Kreis zu drehen. Dabei werden in Intervallen von 1,875 Sekunden insgesamt acht Bilder aufgenommen. Um keine Objekte im Hintergrund, sondern die Person zu verfolgen, wird der Hintergrund dieser Bilder anhand der Tiefendaten entfernt, sodass nur noch die Person auf dem Bild verbleibt. Die verarbeiteten Bilder werden in einem Array gespeichert und in dieser Form an die Klasse zur Bildverarbeitung übergeben. Diese extrahiert aus den Bildern auf Basis des SURF-Algorithmus (engl. „**S**peeded **U**p **R**obust **F**eatures“) sämtliche Erkennungsmerkmale des Bildes, welche sich aus Ecken, Kanten und Farbdifferenzen zusammensetzen. Diese werden zwischen zwischengespeichert, um sie zur Wiedererkennung der Person zu verwenden.

```
public static void learn(Mat[] objects) {
    for (int i = 0; i < imageCount; i++) {
        Mat objectImage = objects[i];

        MatOfKeyPoint objectKeyPoints = new MatOfKeyPoint();
        MatOfKeyPoint objectDescriptors = new MatOfKeyPoint();
        featureDetector.detect(objectImage, objectKeyPoints);
        descriptorExtractor.compute(objectImage, objectKeyPoints, objectDescriptors);

        Mat outputImage = new Mat(objectImage.rows(), objectImage.cols(), Highgui.CV_LOAD_IMAGE_COLOR);
        Features2d.drawKeypoints(objectImage, objectKeyPoints, outputImage, newKeypointColor, 0);

        processedObjectKeyPoints[i] = objectKeyPoints;
        processedObjectDescriptors[i] = objectDescriptors;
        processedOutputImages[i] = outputImage;
        processedObjects[i] = objects[i];
    }
}
```

Abb. 12 - Ausschnitt aus der ersten Version der Kinect-Software: Verarbeiten der Bilder (1)

Nun folgt eine Endlosschleife, welche sich in die restlichen drei Teile gliedert.

Zunächst wird in der Schleife das Farbbild vom Sensor abgerufen.

Auf diesem wird nun auf Basis der vorher aus den Bildern gesammelten Identifikationsmerkmale nach dem Besitzer gesucht und sofern dieser gefunden wurde, seine Position auf der X-Achse bestimmt. Die Verarbeitung des Bildes passiert parallel, wobei für jedes der im zweiten Teil erstellten Bilder ein einzelner Thread erstellt wird, welcher dieses mit dem aktuellen Farbbild abgleicht. Das Ergebnis des Threads mit den meisten Übereinstimmungen wird weiter verwendet.

```
// gets preprocessed object-keypoints
Mat objectImage = processedObjects[side];
MatOfKeyPoint objectKeyPoints = processedObjectKeyPoints[side];
MatOfKeyPoint objectDescriptors = processedObjectDescriptors[side];

// gets scene-keypoints
MatOfKeyPoint sceneKeyPoints = new MatOfKeyPoint();
MatOfKeyPoint sceneDescriptors = new MatOfKeyPoint();
featureDetector.detect(input, sceneKeyPoints);
descriptorExtractor.compute(input, sceneKeyPoints, sceneDescriptors);
KeyPoint[] sceneKeyPointsArray = sceneKeyPoints.toArray();

// gets matches of scene and object
Mat matchoutput = new Mat(input.rows() * 2, input.cols() * 2, Highgui.CV_LOAD_IMAGE_COLOR);
List<MatOfDMatch> matches = new LinkedList<MatOfDMatch>();
descriptorMatcher.knnMatch(objectDescriptors, sceneDescriptors, matches, 2);
```

Abb. 13 - Ausschnitt aus der ersten Version der Kinect-Software: Verarbeiten der Bilder (2)

Nach der Bestimmung der Position wird diese über einen „DataOutputStream“ via WLAN an den EV3 gesendet.

Version 2

Die finale Version unser Software gliedert sich nur noch in 3 wichtige Teile: Die Initialisierung, das Ermitteln der Startwerte und die Endlosschleife. Auch stützt sich die zweite Version nicht mehr auf den SURF-Algorithmus, sondern verwendet einen „MOSSE-Tracker-Algorithmus“, welcher auf adaptiven Korrelations-Filtern basiert und (vereinfacht) immer den vorherigen Frame, mit dem aktuellen Frame vergleicht.

Bei der Initialisierung wird der Sensor gestartet, und es werden Ausleseeinheiten für die verschiedenen Datenströme des Kinects, wie zum Beispiel den Farb- oder Tiefendatenstrom, eingerichtet. Des Weiteren verbindet sich der Laptop in diesem Schritt mit dem EV3 um ihm, wie auch in Version 1.0, später die Position des Besitzers mitzuteilen.

Um die Startwerte für die Software zu ermitteln, wird die von Microsoft entwickelte Methode zur Personenerkennung verwendet, da der Roboter zu diesem Zeitpunkt noch steht und daher diese noch funktioniert. Es werden die Positionen sämtlicher vom SDK erkannter Körperteile erfasst und daraus eine maximale Distanz und eine minimale Distanz zur Person ermittelt. Des Weiteren wird auf Basis der Hüft- und Schulterpositionen ein Bereich festgelegt, welcher mittels „MOSSE-Tracker“ verfolgt werden soll. Der Tracker wird dann mit einem Bild, auf dem alles ausgeschnitten wurde, was nicht innerhalb der vorher ermittelten minimalen bis maximalen Distanz liegt, sowie dem ermittelten, zu verfolgenden Bereich gestartet.

Die darauf folgende Endlosschleife läuft ab wie folgt:
Zuerst werden sämtliche Farb- sowie Tiefendaten aktualisiert, dann wird ein Bild erstellt, auf welchem alles außer der Bereiche, welche im Bereich zwischen der minimalen und der maximalen Distanz zwischen Person und Sensor liegen, ausgeschnitten werden, sodass nur noch die Person und alles, was sich mit ihr auf einer Ebene befindet, bleibt. Das Bild wird an den Tracker weitergegeben,

welcher dieses verarbeitet und anhand der Ergebnisse einen neuen Bereich, in dem die Person sich befindet, definiert. Für die Pixel in diesem Bereich werden nun die jeweiligen Tiefendaten abgerufen und auf deren Basis eine neue minimale und maximale Distanz ermittelt. Die neue minimale Distanz sowie die X-Position der Person werden über die in der Initialisierung hergestellte Verbindung an den EV3 übermittelt.

```
while (true) {
    bool updated = updateData();
    if (updated) {
        cv::Mat m = getColorInDepthRange(minZ, maxZ);
        bool TRACKING_OK = tracker->update(m, bbox);
        if (TRACKING_OK) {
            rectangle(m, bbox, cv::Scalar(255, 0, 255), 2, 1);
            cv::Point p = cv::Point(bbox.x + bbox.width / 2, bbox.y + bbox.height / 2);
            int nMinZ = 8000;
            int nMaxZ = 0;
            coordinateMapper->MapColorFrameToDepthSpace(depthWidth * depthHeight, (UINT16*)depthBuffer, colorWidth * colorHeight, depthCoordinates);
            for (int i = 0; i < bbox.width; i ++) {
                for (int j = 0; j < bbox.height; j++) {
                    cv::Point x = cv::Point(bbox.x + i, bbox.y + j);
                    int pIndex = x.y*colorWidth + x.x;
                    int z;
                    if (pIndex > 0 && pIndex < (colorWidth * colorHeight)) {
                        DepthSpacePoint p = depthCoordinates[pIndex];
                        if (p.X != -std::numeric_limits<float>::infinity() && p.Y != -std::numeric_limits<float>::infinity()) {
                            int depthX = static_cast<int>(p.X + 0.5f); int depthY = static_cast<int>(p.Y + 0.5f);
                            if ((depthX >= 0 && depthX < depthWidth) && (depthY >= 0 && depthY < depthHeight)) {
                                z = depthBuffer[depthX + (depthY * depthWidth)];
                            }
                        }
                        if (z > minZ - 10 && z < maxZ + 10) {
                            if (z < nMinZ) {
                                nMinZ = z;
                            }
                            if (z < nMaxZ) {
                                nMaxZ = z;
                            }
                        }
                    }
                }
            }
            minZ = nMinZ;
            maxZ = nMaxZ;
            std::string str = std::to_string(p.x - (colorWidth / 2)) + "#" + std::to_string(minZ);
            send(wlanSocket, str.c_str(), (int)strlen(str.c_str()), 0);
        }
        imshow("M", m);
    }
}
```

Abb. 14 - Ausschnitt aus der zweiten Version der Kinect-Software: Endlosschleife

Ergebnisse

Im Folgenden werden wir sowohl die Vorteile unseres Projekts, als auch die Nachteile aufstellen.

Ein wichtiger Vorteil unseres Projekts liegt in den Kosten. Im Punkt zur Optimierung der Hardware des Roboters gehen wir genauer auf die Kosten in einer Massenproduktion ein. Nachdem wir eine Gesamtsumme des Roboters von 250 € schätzen, stellt dies eine niedrige Summe im Vergleich zu Haushaltshilfen dar. Während eine Haushaltshilfe in einem Jahr durchschnittlich 1600 € kostet bei drei Stunden wöchentlicher Arbeit, kostet der Roboter einmal 250 € und kann über eine lange Zeit hinweg zur Verfügung stehen.

Da eine Haushaltshilfe jedoch auch andere Arbeiten außer Einkäufen verrichtet, lohnte sich der Kauf unseres Roboters vor allem für Personen, welche Schwierigkeiten beim Tragen schwerer Gegenstände und somit beim Einkaufen haben.

Weiterhin bringt der Roboter beispielsweise den Vorteil einer Sprachsteuerung mit sich. Die Sprachsteuerung ermöglicht dadurch, dass sie durchgehend (sofern sie nicht in den Einstellungen deaktiviert wurde) im Hintergrund läuft, dem Besitzer einen simplen und zuverlässigen Gebrauch. Somit muss der Besitzer nicht einmal eine Fernbedienung benutzen.

Der Nachteil der Sprachsteuerung liegt allerdings darin, dass ein Smartphone mit durchgehender Internetverbindung gebraucht wird, um die Sprachbefehle auswerten zu können. Dies verbraucht jedoch sehr wenig Datenvolumen.

Durch die weiteren Zusatzfunktionen der App, wie die integrierte Einkaufsliste und die Notruffunktion, bringt der Roboter weitere Vorteile gegenüber einer Haushaltshilfe mit sich.

Genutzte Bibliotheken

Software Version 1:

Bei der ersten großen Version zur Steuerung des Roboters verwendeten wir die folgenden Bibliotheken:

- OpenCV (https://opencv.org)

  → Eine Bibliothek zur Bildverarbeitung

- KinectPV2 (https://github.com/ThomasLengeling/KinectPV2)

  → Ein Wrapper für das Kinect-SDK zum ansprechen des Sensors

- Processing 2 (https://processing.org)

  → Eine Bibliothek, genutzt von KinectPV2

- JOGL (Java OpenGL) (https://jogamp.org/jogl/www/)

  → Eine Bibliothek zur Visualisierung von Bildern, genutzt von Processing 2

- Gluegen (https://jogamp.org/gluegen/www/)

  → Eine Bibliothek, genutzt von JOGl

- LeJOS EV3 (http://www.lejos.org/ev3.php)

  → Eine Bibliothek zur Steuerung des EV3s

Software Version 2:

Bei der zweiten, finalen Version zur Steuerung des Roboters verwendeten wir die folgenden Bibliotheken:

- OpenCV (https://opencv.org)

  → Eine Bibliothek zur Bildverarbeitung

- Microsoft Kinect-SDK (https://developer.microsoft.com/de-de/windows/kinect/develop)

  → Eine von Microsoft entwickelte Bibliothek zum ansprechen des Kinect-Sensors

- Winsock2 (https://msdn.microsoft.com/de-de/library/windows/desktop/ms740673.aspx)

  → Eine Standardbibliothek zur Netzwerkkommunikation

- LeJOS EV3 (http://www.lejos.org/ev3.php)

  → Eine Bibliothek zur Steuerung des EV3s

Diskussion

Probleme bei der Umsetzung

Da wir, um unser Projekt umsetzen zu können, nicht weiter mit der einfachen, LEGO-eigenen Software arbeiten konnten, mussten wir, da wir es nie in der Schule gelernt hatten, Java auf eigene Faust lernen. Dies war aufgrund mangelnden Angebots nur im Internet via Videotutorials möglich. Da man dabei keinen direkten Ansprechpartner, wie beispielsweise einen Lehrer, hat, mussten wir uns alle Fragen außerhalb der Tutorials durch „Googlen“ selbst beantworten.

Des Weiteren führt, da LEGO nicht für derartig große Projekte gedacht ist, auch die Benutzung von LEGO-Technik zu einigen Problemen. Ein sehr großes Problem, vor allem für das Treppensteigen, war, dass die kompatiblen Motoren recht schwach sind. Für den derzeitigen Prototyp ist dies zwar kein Problem, jedoch bringen die Motoren auch mit sich, dass sich bei langem Vor- und Zurückfahren der Null-Punkt aufgrund der aus Reibung mit dem Boden resultierenden Ungenauigkeit verschiebt. Dies ist vor allem für die Lenk-Motoren ein Problem.

Weiterentwicklungsmöglichkeiten

Optimierung der Hardware

In Bezug auf die Weiterentwicklung des Roboters haben wir viele Ideen. Die meisten davon konnten wir bisher allerdings noch nicht umsetzen, da diese einiges mehr an Geld zur Entwicklung benötigen würden, als wir derzeit zur Verfügung haben. Im Folgenden listen wir Einige auf.

Als wichtigste Verbesserung des Roboters sehen wir den Austausch des jetzigen Materials gegen ein serienmäßig hergestelltes Hartplastik-Chassis. Dies wäre bei einer seriellen Herstellung jedoch ohnehin Normalfall. Durch diese Veränderung wäre durch eine drastisch erhöhte Stabilität auch eine noch höhere Traglast denkbar.

Eine weitere Optimierung wäre durch ein Austauschen der Motoren gegen leistungsstärkere möglich. Durch einige Umbauten ähnlich den Armen bei Prototyp 3 könnte der Roboter so auch Treppen steigen.

Dadurch, dass im Rahmen unseres Projektes ein Prototyp entsteht, werden einige Teile verbaut, an deren Stelle in einer Produktion ressourcensparendere und günstigere Komponenten verwendet werden könnten. Man könnte statt eines Laptops und eines EV3s beispielsweise ein extra dafür angefertigtes SoC („**S**ystem **O**n a **C**hip“) mit einem relativ schwachen Prozessor (vergleichbar Intel Pentium G4560 (~65€) mit wenig Arbeitsspeicher (ab 2 GB) und ohne Grafikeinheit verwenden. Auch anstelle des Kinect-Sensors wäre natürlich eine den Notwendigkeiten angepasste Version denkbar. Insgesamt käme man so in einer Massenproduktion, selbst bei normalen Kundenpreisen, auf einen Gesamtpreis von etwa 250€. Des Weiteren könnte man leistungsstärkere Akkus verbauen, was zu einer längeren Benutzbarkeit führen würde.

Optimierung der Software

Einen der bei Ergebnisse genannten Nachteile stellt die dauerhaft benötigte Internetverbindung dar. Weil diese lediglich für die Spracherkennung benötigt wird, (da diese die Google „Cloud-Speech-API“ verwendet) könnte man das Problem mit einer auf dem lokalen Laptop laufenden Spracherkennung beheben.

Außerdem könnte man, um dem Nutzer den Weg zum Einkaufen noch einfacher zu machen, dem Programm eine Funktion hinzufügen, welche zum einen bereits gefahrene Wege speichert, sodass

der Roboter diese dann voraus fahren kann und zum anderen auch Wege zu nahegelegenen Supermärkten beispielsweise von Google-Maps beziehen und so voraus fahren kann.

Zuletzt könnte man den Verbindungsprozess aller derzeit involvierten Geräte weiter automatisieren und beschleunigen, da dies aktuell noch eine bis mehrere Minuten in Anspruch nehmen kann.

Danksagung

An dieser Stelle möchten wir allen Personen danken, welche uns auf unserem Weg zum finalen Projekt geholfen haben.

Insbesondere danken wir Lukas Justen für seine Hilfe bei der Programmierung einer Bluetooth-Verbindung zwischen EV3 und Smartphone.

Des Weiteren bedanken wir uns bei Herrn Ossmann für seine Unterstützung im Laufe der letzten Jahre, sowie für die Betreuung dieses Projekts.

Quellen- und Literaturverzeichnis

OpenCV 2.4.3 Java-Dokumentation:
https://docs.opencv.org/java/2.4.3/

The Morpheus Tutorials, Android Tutorials Deutsch:
https://www.youtube.com/playlist?list=PLNmsVeXQZj7qShNeVpdDAQedIq2n8BvqC

Stackoverflow: Android Speech Recognition as a service on Android 4.1 & 4.2:
http://stackoverflow.com/questions/14940657/android-speech-recognition-as-a-service-on-android-4-1-4-2/14950616#14950616

Android Developer Webseite:
https://developer.android.com/guide/topics/connectivity/bluetooth.html

IT in der Hosentasche: Bluetooth zwischen Raspberry Pi und Android:
http://it-in-der-hosentasche.blogspot.de/2014/03/bluetooth-zwischen-raspberry-pi-und.html

LeJOS EV3 Dokumentation:
http://www.lejos.org/ev3/docs/

Lengling, Thomas: KinectPV2 Bibliothek:
https://github.com/ThomasLengeling/KinectPV2

Colingorrie: Ausreißer in Arrays finden:
http://colingorrie.github.io/outlier-detection.html#iqr-method