Loomo

Objektumfahrung

Komplexpraktikum

David Köhler
Falko Krieg

Lehrstuhl für Mensch-Computer-Interaktion
Betreuer: David Gollasch
Betreuender Professor: Gerhard Weber

Inhaltsverzeichnis

Einleitung

Kontext
Verschiedentliche Motivationen begründen einen sich erhärtenden Bedarf an smarter, assistiver Technologie in Form autonomer Assistenzroboter. Ein gesellschaftlicher Druck geht dabei besonders von der alternden Bevölkerung in Verbindung mit dem angespannten Pflegesektor aus. Hier sind die Hoffnungen groß, das Leben von alten Menschen sowie von Pflegekräften deutlich zu erleichtern, indem Assistenzroboter zukünftig ein breites Spektrum von Unterstützung in unterschiedlichen Alltagssituationen bieten können. Zwar sind die Entwicklungen in der Robotik rasant schnell, aber reale Anwendungsfälle beschränken sich bislang eher auf industrielle Fertigungsroboter. Ein Knackpunkt im Bereich der Assistenzroboter ist der Mangel an Funktionsvielfalt, welche einen Roboter überhaupt erst interessant macht. Ein Lösungsansatz ist hier das Bereitstellen einer gemeinsamen, erweiterbaren Entwicklungsplattform. Der Segway Robotics „Loomo“ bietet hierfür eine Basis bestehend aus Hardware- und Softwareplattform. Die Hardware besteht aus einem Segway/Ninebot Self-Balancing Vehicle in Kombination mit einer Intel-Atom-basierten Recheneinheit und verschiedentlichen Sensoren und Aktuatoren zur Wahrnehmung und Interaktion mit der Umgebung. Die Softwareplattform bildet Android in Verbindung mit einem SDK zur Ansteuerung der Sensorik und Aktorik. Die Implementierung von Funktionen für spezifische Anwendungsfälle erfolgt in Form von Android-Apps mit Zugriff auf das Steuerungs-SDK. Weiterhin erlaubt Loomo auch die Erweiterung der Hardware mittels Erweiterungskupplung bestehend aus belastbarer Metallaufhängung, USB-Anbindung und zusätzlicher Stromversorgung.

Ziel
Um reale Anwendungsszenarien abbilden zu können, müssen zunächst Basisfunktionalitäten geschaffen werden. So sollte Loomo zu einer Art Butler entwickelt werden. Hierzu muss Loomo seine Umgebung kennenlernen und erkunden können sowie geeignete Kommunikationsstrategien, entsprechend der Anforderungen seiner Nutzer, unterstützen. Ziel dieses Komplexpraktikums soll es sein, die Navigation von Loomo in bekannten Umgebungen zu verbessern.

Anforderungen
Die Zielerreichung setzt eine umfassende Planung voraus. Das Szenario soll in einer einzelnen App bereitgestellt werden. Die Funktionalitäten selbst sollen jedoch klar getrennt werden, um diese bei Bedarf in einer anderen App weiterverwenden zu können.
3.1 Teilaufgaben: Navigation verbessern 4 Konzeption einer geeigneten Navigationsstrategie bei bekannter Umgebung. 4 Konzeption einer geeigneten Strategie zum Erkennen und Umfahren von Hindernissen. 4 Entwurf einer grafischen Nutzerschnittstelle zur Steuerung der zu entwickelnden App. 4 Einbezug von bereits bekannten Mechanismen zur Obstacle Avoidance und Navigation auf Basis von Graphen. 4 Implementierung der Anwendung. 4 Dokumentieren Sie Ihre Arbeit präzise. 4 Erfüllen Sie Bonus-Anforderungen (bspw. verschiedene Navigationsstrategien (kürzester Weg, Travelling-Salesman-Problem bei mehreren Wegpunkten, ...)

Loomo

„Loomo (Figur 2 “Loomo”) ist ein Roboter, früher unter dem Namen Segway Roboter bekannt, welcher durch das Erstellen eigener Applikationen sowohl Designern, als auch Ingenieuren die Möglichkeit bietet einen eigenen Personal-Roboter zu erstellen.“ (1) Aufgrund vieler Sensoren, über die Loomo verfügt, ist es den Entwicklern möglich anhand des Software Development Kit (SDK) eine Vielzahl von Funktionen in Loomo einzuprogrammieren. Durch das SDK erhält man Zugriff auf die Base (den Hauptteil des Roboters) und den Head (der Kopf des Roboters), des Weiteren kann man die große Vielzahl an Sensoren sowohl einzeln, als auch gleichzeitig ansteuern. Beispiele für diese Sensoren sind die Intel RealSense Camera oder der integrierte Ultraschallsensor, in Figur „Hardware Spezifications“ erhalten Sie einen Überblick über die enthaltenen Sensoren.

„Loomo basiert auf einem selbst balancierten Zweiradfahrzeug und erlaubt eine schnelle Transformation zwischen Roboter und Scooter. Er kann bei voller Ladung eine Distanz von 30km zurücklegen. “ (1) Weitere Informationen finden Sie im Segway Robotics Developer Portal.

Abb. 1 “Loomo" (2)

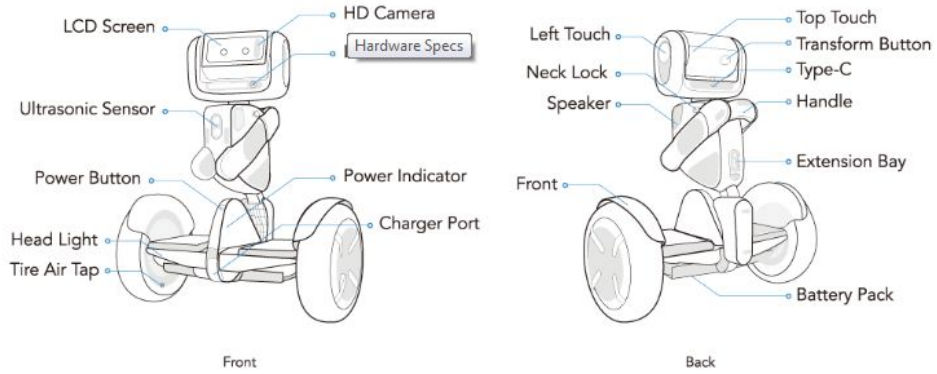

Abb. 2 “Spezifikationen Loomo”(1)

Hardware Specs	Descriptions
Dimension	Height 0.64m, length 0.57m, width 0.28m
Weight	17.5kg, including the battery pack
Battery Capacity	310Wh
Speed Limit	8 kilometers per hour (software limit for the Alpha edition)
Pan Tilt Unit Range	Pan ±150 degrees, Tilt -90 ~ +180, zero is horizontal facing front
LCD Screen	4.3 inch

Platform Specs	Descriptions
Processor	Intel Atom Z8750, 4 cores, 2.4GHz, x86-64 architecture
Operating System	Customized system based on Android 5.1
Memory	4 GB
Storage	64 GB
USB port	USB 3.0 Type-C cable (compatible with USB 2.0)
Extension Bay	USB 2.0 and 24 voltage power (1A limit)

Sensor Specs	Descriptions
RealSense	Real time 30Hz RGB-Depth streaming for both indoor and outdoor uses
HD Camera	High resolution with 104° FOV (Pending fine turning. Image quality is not optimal)
Mic Array	Beamforming and voice localization, powered by 5 microphones
Ultrasonic Sensors	Obstacle distance calculation
Touch Sensors	Located at robot head left, right & back
Wheel Odometry	Approximately 4 degrees precision
Base IMU	6-axis IMU

Abb 3 “Sensoren”(1)

Konzept

Grundkonzept

Wenn Loomo einen vorgegeben Weg fährt stößt er in Gebäuden unweigerlich auf Hindernisse. Unser Konzept setzt an diesen Punkt an. Loomo fährt gerade aus (auf dem vorgegebenen Weg) und erkennt vor sich anhand des Bildes der Tiefenkamera und des Ultraschallsensor ein Objekt. Er fährt langsam an dieses Objekt heran, bis auf eine Entfernung von 60cm. Nachdem er kurz vor diesem Objekt hält, entscheidet er ob er sich nach Rechts oder Links drehen sollte, anhand des Bildes der Tiefenkamera, dass er vor dem Heranfahren ausgewertet hat. Steht das Objekt eher links von ihm dreht er sich nach Rechts, steht es eher rechts von ihm dreht er sich nach Links. Loomo dreht sich so weit, bis er das Objekt mit dem Ultraschallsensor nicht mehr wahrnehmen kann. Anschließend dreht er seinen Kopf um 90 Grad in Richtung des Objektes. Sobald er sich anhand dieser Schritte ausgerichtet hat, fährt er geradeaus und das so lange, bis die Werte seiner Tiefenkamera sich sehr stark voneinander unterscheiden. Wenn ihm die Werte sagen, dass das Objekt näher kommt korrigiert er sich etwas von dem Objekt weg, erkennt er durch die Werte, dass das Objekt sich entfernt so korrigiert er sich in Richtung des Objektes, anhand dieser Korrekturen tastet sich Loomo langsam um das Objekt herum. Diese Korrektur betreibt Loomo so lange bis er merkt, dass er wieder auf derselben Höhe angekommen ist, von der er gestartet hat. Danach richtet er sich wieder gerade aus und fährt weiter.

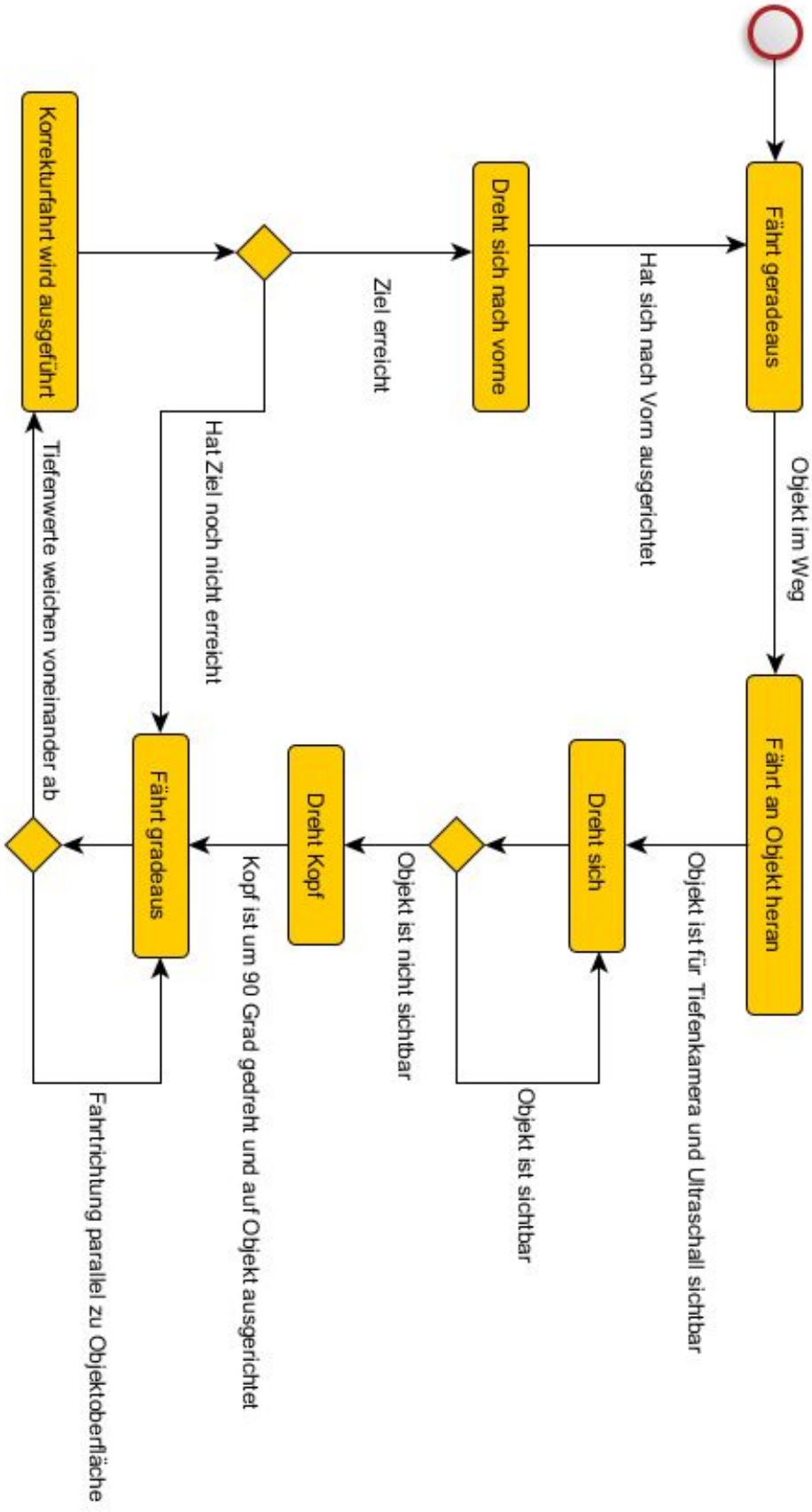

Abb. 4 “Zustandsdiagramm Hauptalgorithmus”

Fälle

Im Folgenden werden die verschiedenen Fälle beschrieben, die in unserem Projekt betrachtet werden. Ebenso wird kurz umrissen, wie Loomo mit diesem Fall umgehen soll. Es gibt noch deutlich mehr mögliche Fälle, wie Loomo auf ein Hindernis treffen kann. Diese zu lösen würde allerdings den Rahmen des Praktikums sprengen.

Fall 1: Kein Objekt befindet sich in Fahrtrichtung
Der Idealfall. Hier wird die Hindernisumfahrung nicht benötigt.

Fall 2: In Fahrtrichtung ist ein Objekt, das parallel zu Loomo steht
Das Standardproblem für unser Projekt. Hier muss man zwischen drei genaueren Fällen unterscheiden.

Fall 2.1: Objekt steht frontal vor Loomo
Loomo muss sich für eine Umfahrrichtung entscheiden und dann das Objekt auf dieser Seite umfahren.

Fall 2.2: Objekt steht rechts vor Loomo
Loomo umfährt das Objekt linksherum.

Fall 2.3: Objekt steht links vor Loomo
Loomo umfährt das Objekt rechtsherum.

Fall 3: In Fahrtrichtung ist ein Objekt, das schräg zu Loomo steht
Dies ist die gleiche Problemstellung, wie Fall 2, allerdings werden hier besondere Ansprüche an die Sensoren und -verarbeitung gestellt, weshalb sie für die Implementierung gesondert behandelt wird.

Umsetzung

Verwendete SDKs

Folgende SDKs wurden in unserer App verwendet:

Von **Android**:

android.graphics.Bitmap;
android.support.annotation.Nullable;
android.support.v7.app.AppCompatActivity;
android.os.Bundle;
android.util.Log;
android.view.SurfaceView;
android.view.View;
android.widget.Button;
android.widget.CompoundButton;
android.widget.Switch;

Von **Loomo:**

com.segway.robot.sdk.locomotion.head.Head;
com.segway.robot.sdk.base.bind.ServiceBinder;
com.segway.robot.sdk.locomotion.sbv.Base;
com.segway.robot.sdk.vision.Vision;
com.segway.robot.sdk.perception.sensor.*;
com.segway.robot.sdk.vision.stream.StreamType;

Von **Java**:

java.util.Timer;
java.util.TimerTask;

Klassen

Die verwendet Klassen und ihre Beschreibungen können der beiliegenden Javadoc entnommen werden.

Algorithmen und Methoden

Im nachfolgenden werden die erstellten Algorithmen beschrieben, mit denen die Umsetzung geschrieben wurde.

fahreGeradeaus, Stop

Triviale Fahralgorithmen in denen mit einer bestimmten Geschwindigkeit geradeaus gefahren oder angehalten wird. Loomo SDK besitzt solche Algorithmen nicht, also mussten sie von uns erstellt werden.

fahreLinks, fahreRechts

Diese Algorithmen werden aufgerufen, wenn der Ultraschallsensor ein Objekt erkennt. Sie bilden gemeinsam mit dem Richtungsbeurteilungsalgorithmus (**AvoidanceInDirection**) den

Start des Umfahralgorithmus. Die Fahrtrichtung des Loomos ist dann Senkrecht zur Objektoberfläche. Ziel des Algorithmus ist es, die Fahrtrichtung parallel zur Oberfläche zu setzen. Dazu dreht sich Loomo auf der Stelle, bis der Ultraschallsensor ausgibt, dass er die Objektoberfläche nicht mehr erkennt. Unsere Tests haben gezeigt, dass dieser Ansatz in nahezu 100% der Fälle funktioniert. Kleinere Ungenauigkeiten im Ergebnis (Fahrtrichtung nicht perfekt parallel zur Objektoberfläche) werden später im Umfahralgorithmus gelöst.

umfahrLinks, umfahrRechts
Diese Algorithmen werden im Umfahralgorithmus aufgerufen, wenn die Tiefenkamera erkennt, dass Loomos Fahrtrichtung nicht mehr parallel zur Objektoberfläche ist. Loomo fährt eine leichte Links- bzw. Rechtskurve, um wieder Parallel zu werden.

IImageState: updateImage
In dieser Methode wird der Umfahralgorithmus aufgerufen und durchgeführt. Sie ist das Kernstück unserer Umsetzung. Sie spiegelt auch das Ablaufdiagramm aus dem Kapitel Konzept wieder. Sie wird in jedem Frame der Anwendung aufgerufen. Zu beginn des Programms werden werden die ersten 50 Frames gewartet, um die Tiefenkamera und den Ultraschallsensor zu kalibrieren. Dies ist für die Tiefenkamera zuverlässig, für den Ultraschallsensor in ¾ der Fälle.
Nach der Kalibrierungsphase fährt Loomo nach vorne. Dies tut er solange, bis sein Weg von einem Hindernis unterbrochen wird. Hat er mit dem ultraschallsensor ein Hindernis entdeckt, fährt er auf ca 60 cm heran, beurteilt mit **AvoidanceInDirection** die Umfahrrichtung und dreht sich mit **fahreLinks** oder **fahreRechts** in die entsprechende Richtung. Anschließend wird der Kopf in einem 90° Winkel zu Loomo auf das Objekt ausgerichtet. Nun überprüft er mit **umfahrSchauen** seine Position in relation zum Objekt und macht mit **umfahrRechts** und **umfahrlinks** gegebenenfalls korrekturen an seiner Position. sind keine Korrekturen notwendig fährt er geradeaus. So tastet er sich um das Objekt herum. Hat das Objekt eine harte Abbiegung (beispielsweise ein Tetris-Z) erkennt er es über den Ultraschallsensor und richtet sich mit **fahreLinks** und **fahreRechts** neu aus. Loomo merkt sich wie weit er sich gedreht hat und hat dafür die Zählvariable **currentPosition**. Wenn **currentPostion** ca. 0 (plus/minus Schwellwert) erreicht hat, ist die Umfahrung abgeschlossen. **currentPostion** wird mit **umfahrLinks, umfahrRechts, fahreLinks, fahreRechts** abhängig von der Umfahrrichtung erhöht bzw verringert.

Wenn das Umfahren beendet ist, wird der Umfahralgoritmus beendet, Loomo richtet sich nach vorne aus, und auf dem Bildschirm wird angezeigt, dass das Objekt umfahren wurde. Zu Präsentationszwecken hält Loomo komplett an, er könnte allerdings auch einfach seinen Weg fortsetzen und wieder von vorne beginnen. Darauf ist der Algorithmus ausgerichtet.

straightHead
Diese Methode richtet den Kopf nach vorne aus und dreht den Bildschirm gleichzeitig nach hinten, damit man Loomo hinterherlaufen und die Kamerabilder und Debuginformationen sehen kann. Sie wird am Ende jedes Umfahrzyklus und beim Start des Programms aufgerufen

AvoidanceInDirection
Der erste Bilderkennungsalgorithmus der Anwendung. Hier beurteilt Loomo ob er an einem durch den Ultraschallsensor erkannten Objekt Links- oder Rechtsherum vorbeifahren soll. Dazu betrachtet er den vor sich liegenden Raum mit der Tiefenkamera. Deren Bild wird dann von diesem Algorithmus analysiert. Es wird (zur Beschleunigung des Algorithmus) das Bild im 5-Pixel-Raster abgetastet. Des Weiteren wird das Bild oben zugeschnitten. Objekte, die über Loomos kopf wären und unter denen Loomo einfach durchfahren kann werden damit nicht betrachtet. Die einzelnen Pixel werden nun auf ihre Farbe untersucht. Der Grünwert gibt Informationen über die Distanz aus. Ist er Null(0), ist nichts im Weg. Somit werden verglichen auf der Linken und Rechten Seite verglichen, wo mehr schwarze Pixel sind. Wenn auf beiden Seiten ein Schwellwert von schwarzen Pixeln unterschritten wird, heißt das, Loomo steht vor einer Wand, die so breit ist, dass das Kamerabild Links und Rechts keine Vorbeifahrrichtung erkennen kann.
Je nachdem, auf welcher Seite mehr schwarze Pixel sind, gibt der Algorithmus Links oder Rechts (durch -1 und 1 beschrieben) als Umfahrrichtung zurück. Wenn eine Wand erkannt wird, wird ein Sonderwert (400) zurückgegeben, der in unserem Algorithmus allerdings wie Links behandelt wird. Ein gesonderter Umgang mit diesem Wert kann in darauf aufbauenden Projekten umgesetzt werden.

umfahrSchauen
Der zweite Bilderkennungsalgorithmus. Auch hier wird das Bild der Tiefenkamera analysiert. Dieser Algorithmus wird verwendet, wenn Loomos Fahrtrichtung parallel zur Objektoberfläche ist. Der Kopf ist dann um 90° nach Links bzw rechts gedreht und zeigt auf das Objekt. (Abbildung) Das Bild wird wieder im 5-Pixel-Raster untersucht, allerdings werden diesmal nur zwei Bereiche abgetastet. Ein linker und ein rechter Teil des Bildes. Die beiden Bereiche können durch globale Variablen wie Streifenbreite, Streifenabstand, Zentrumsanfang usw. angepasst werden. Ist der Kopf nach Rechts gedreht, ist der Linke Teil des Bildes vorne, der rechte Teil hinten. Der durchschnitt ihrer Grünwerte wird miteinander verglichen. Ein kleinerer Grünwert bedeutet Nähe, ein größerer Grünwert bedeutet Distanz. Ebenso wird sichergestellt, dass beide Streifen gleichmäßige Werte enthalten. Enthält einer der Streifen zu sprunghafte Werte wird angenommen, dass das Objekt nicht mehr gesehen wird.
Wenn beide Werte ungefähr gleich sind, bedeutet dies, dass Loomo parallel zur Objektoberfläche steht und weiter geradeaus fahren soll.
Ist der vordere Wert größer als der hintere, bedeutet dies, dass das Objekt vorne nach rechts abfällt und Loomo nach rechts korrigieren muss.
Ist der hintere Wert größer als der vordere bedeutet dies, dass das Objekt nach Links anwächst und Loomo Korrigiert sich nach Links.
Für die Korrektur werden die Befehle **umfahrRechts** und **umfahrLinks** verwendet.
Für den Fall, dass der Kopf nach Links gedreht ist, wird das Ganze spiegelverkehrt ausgeführt.

ultraSonic, ultraSonicNear
Algorithmen, um zu überprüfen ob ein Objekt in einem bestimmten Abstand vor Loomo ist. **ultraSonic** wird verwendet, wenn er nicht im Umfahrzustand ist, **ultraSonicNear** wird verwendet, wenn er im Umfahrzustand ist.

Bedienung

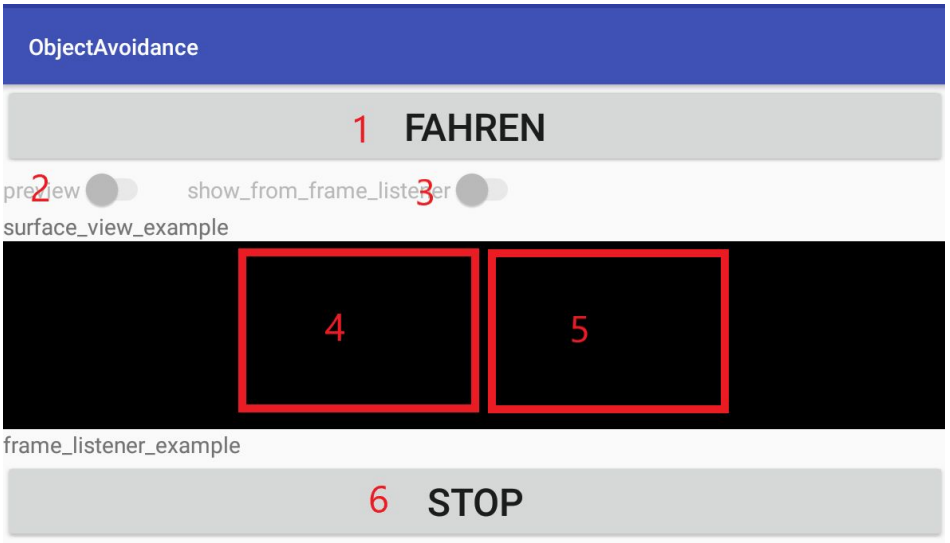

Abb 5 Screenshot der App

In Abbildung 5 sehen Sie den Aufbau der App.
Zum Starten der Applikation betätigen Sie den linken Togglebutton(2). Dadurch wird die Anzeige (Normale Kamera 4, Tiefenkamera 5) der Intel Realsens gestartet. Durch das Betätigen des zweiten Togglebuttons (3) beginnt der Fahralgorithmus. Mit dem oberen Button (1) richten Sie den Kopf aus und mit dem unteren Button (6) wird die Applikation beendet. Die Buttons zeigen den Zählerstand für die Rotation der Base an. Der obere Button (1) zeigt den derzeitigen Stand an und der untere Button (6) zeigt den Startwert an.

Probleme und Lösungen

Im Nachfolgenden finden Sie eine Aufzählung von Problemen, auf die wir während unserer Arbeit mit Loomo gestoßen sind. Sofern wir sie lösen konnten, haben wir eine Lösung dazu angegeben.

Gelöste Probleme

Zu viele Bilder der Tiefenkamera

Die Tiefenkamera liefert so viele Bilder, dass wir sie nicht schnell genug mit unseren Bilderkennungsalgorithmen abarbeiten können, bevor das nächste Bild kommt. Dazu haben wir zwei Lösungen eingearbeitet. Zum einen werden Bilder gespeichert bis sie bearbeitet sind und erst wenn das Bild fertig analysiert worden ist, wird das nächste Bild geladen. Alle weiteren Bilder die in diesem Zeitraum angekommen werden werden nicht geladen und verworfen. Weiterhin werden die Bilder nicht Pixel für Pixel, sondern Rasterartig abgetastet. dadurch wird die Abtastgeschwindigkeit verfünffacht.

Tiefenkameraungenauigkeit bei Bewegung

Während Loomo fährt, liefert die Tiefenkamera ungenaue Tiefenwerte für Objekte die mehr als 1m von Loomo entfernt sind. Diese Ungenauigkeit kann bis zu 2m betragen. Dadurch war es nicht möglich, mit der Kamera zu entscheiden, an welcher Seite Loomo an einem Objekt vorbeifahren soll. Der einzige Wert der zuverlässig war ist der Schwarzwert, also dass kein Hindernis sichtbar ist. Auf diesem Wert aufbauend betrachtet Loomo also nicht mehr das Objekt, sonder schaut, wo überall kein Objekt ist, um zu entscheiden, auf welcher Seite er das Hindernis umfahren soll.

Tiefenkamera versagt bei starker Nähe

Wenn ein Objekt zu nah an der Tiefenkamera ist, also näher als 30cm, erkennt die Tiefenkamera das Objekt nicht mehr als solches, sonder zeigt es Schwarz an, was bedeutet dass dort nichts ist. Die Lösung liegt auf der Hand: Unser Programm erlaubt es Loomo nicht, näher als 30 cm an ein Objekt heranzufahren.

Kopf dreht sich nicht mit

Wenn Loomo seinen Körper dreht, schaut der Kopf weiterhin geradeaus und dreht sich erst dann, wenn der körper bereits zum stillstand gekommen ist. Dies hat uns vor einige Probleme gestellt, da ein Teil unseres Programms darauf aufbaut, dass Loomo sich dreht, bis er bestimmte Dinge mit dem Kopf sieht bzw nicht mehr sieht. Die Lösung dafür war es, dem Kopf gleichzeitig mit dem Körper einen Drehbefehl zu geben, der etwas stärker als der des Körpers ist. Damit steuert er der Körperdrehung entgegen und sitzt weiterhin perfekt auf 90° bzw 0°.

Positionserkennung

Loomo hat eine eingebaute Positionserkennung, über die er seine eigene Position jederzeit feststellen kann. Diese sollte verwendet werden, um zu erkennen, wann er um das Hindernis herumgefahren ist, und somit seinen weg normal fortsetzen kann. Leider hat diese Positionserkennung eine Ungenauigkeit von +50 cm bis -50cm und die meisten Hindernisse

die Loomo umfährt sind nicht breiter als 1m.
Unsere Lösung ist, dass wir über eine Zählvariable messen, wie oft Loomo in eine vom Weg abweichende Richtung fährt. Sobald er dementsprechend oft in die Gegenrichtung gefahren ist, sollte er also wieder auf dem ursprünglichen Weg sein. Diese Methode hält eine ungenauigkeit von ca 20 cm und ist damit geeignet, um die meisten Objekte zu umfahren.

Ungelöste Probleme

Randerkennung
Loomos Ultraschall hat folgendes Problem: Der Kegel des Ultraschalls ist schmaler als Loomos Schulterbreite (Bild). Das führt dazu, dass Loomo an Objekten hängen bleibt, die er nur mit dem Reifen streift. Eine Lösung dieses Problems wurde während des Projektes über die Tiefenkamera entwickelt, konnte allerdings nicht in das endgültige Ergebnis aufgenommen werden, da die Tiefenkamera für andere Funktionalitäten benötigt wurde.

Schräge Objekte
Wenn Loomo auf ein zu schräges Objekt zufährt, schlägt der Ultraschall nicht aus und somit erkennt Loomo das Objekt nicht. Dies ist ein Hardwareproblem des Ultraschalls, eine optische Lösung über die Tiefenkamera ist im Rahmen des Projekts leider nicht möglich gewesen.

Ultraschall überhitzt
Wenn Loomos Ultraschall, und somit unser Programm zu lange verwendet wird, überhitzt der Ultraschall und gibt falsche Positivwerte zurück. Eine Softwareseitige Lösung für dieses Problem ist uns leider nicht möglich gewesen. Hier ist die einzige Möglichkeit, Loomo auszuschalten und 15-30 min abkühlen zu lassen.

Gittererkennung
Damit die von uns geschriebene Objektumfahrung funktioniert, müssen die umfahrenen Objekte optisch massiv sein(von Vorhängen bis Metallblöcken ist alles möglich). Optisch durchlässige Objekte wie beispielsweise Käfige oder Gitterstangen werden zwar vom Ultraschall häufig erkannt, aber nicht mehr von der Tiefenkamera beim Umfahren.

Mindestgröße
Damit die Objekte zuverlässig umfahren werden können, sollten sie mindestens 50cm Durchmesser haben und mindestens so groß sein wie Loomo. Für schmalere Objekte ist die Tiefenkamera zu langsam und kleinere Objekte kann man nicht umfahren, da wir die Tiefenkamera nicht nach unten neigen können.

Menschen
Unser Programm wurde dafür ausgelegt, dass Loomo alleine im Raum ist während er Objekte umfährt. Menschen, oder andere bewegliche Objekte die nah an Loomo vorbeigehen, während er ein Objekt umfährt, verfälschen die Messergebnisse der Sensoren und können somit dazu führen, dass das Umfahren scheitert.

Lichtverhältnisse
Unser Programm funktioniert nur in einem gut ausgeleuchteten Raum. Tests in schlecht beleuchteten Räumen haben gezeigt, dass die Tiefenkamera dort verfälschte Werte liefert. Eine Softwareseitige Lösung des Problems war im Rahmen des Projekts leider nicht möglich.

Verworfene Ansätze
Im Laufe des Projektes haben wir einige Ansätze entworfen, welche wir nach einiger Zeit wieder verworfen haben, da sie nicht für unser System funktioniert haben.
Ein Ansatz war die Steuerung von Loomo allein über die Bildverarbeitung laufen zu lassen. Wir wollten das Tiefenbild analysieren lassen und Loomo dadurch entscheiden lassen, wo er hinfährt. Diesen Ansatz mussten wir verwerfen, da die Verarbeitungsgeschwindigkeit nicht sehr schnell war und auch sehr ungenau.
Als weiteren Ansatz wollten wir Loomo nur durch den Ultraschall steuern, doch auch das haben wir schnell verworfen, da wir nur mit dem Ultraschall zu wenig Informationen über die Objekte vor Loomo erhalten haben.
Zur Navigation von Loomo und zur Ansteuerung der Räder gibt es zwei Modi. Wir wollten den Navigationsmodus, welcher auf Positionskoordinaten basiert für diese Steuerung verwenden, aber dieser Modus hatte zum Teil eine Ungenauigkeit von bis zu einem Meter. Aus diesem Grund haben wir auch diesen Ansatz verworfen.

Fazit

In diesem Projekt ist es uns gelungen, eine Hindernisumfahrung für Loomo zu entwickeln. Sie wird angewendet, wenn er auf einer vorgegebenen Strecke auf ein Hindernis trifft. Wie die Strecke vorgegeben wird, war nicht Teil unseres Projektes. Aus Demonstrationsgründen fährt Loomo bei uns einfach geradeaus.
Es ist ihm möglich, Hindernisse zu erkennen und zu umfahren. Ebenso betrachtet er das Objekt optisch und schätzt eine ideale Umfahrrichtung ab. Damit wird die Umfahrzeit weiterhin verringert. Die Form des Objektes spielt für Loomo keine Rolle, solange es optisch massiv und mindestens so groß wie er ist. Objekte, die in einer Höhe sind, unter der Loomo einfach hindurchfahren kann, werden ignoriert. Loomo erkennt, wenn das Objekt umfahren ist und richtet sich automatisch wieder aus, um seinen Weg fortzusetzen. Diese Ausrichtung ist leider nicht immer zuverlässig, sollte aber nach einem Softwareupdate des Loomo-SDKs fehlerfrei funktionieren.

Ausblick

Es gibt einige Dinge mit denen man unser System verbessern bzw. erweitern könnte. Eines unserer Probleme war die Erkennung von Objekten, die Loomo beim Vorbeifahren nur leicht streift. Eine sinnvolle Erweiterung wäre also eine Erkennung von solchen Objekten und deren Umfahrung mit in unser System zu integrieren. Des Weiteren hatte Loomo häufig Probleme Wände zu erkennen und hat versucht diese nach unserem Algorithmus zu umfahren. Somit wäre eine weitere sinnvolle Erweiterung eine Wanderkennung, welche Loomo veranlasst diese als Objekt zu ignorieren. Ein weiteres Problem unseres Systems sind sehr komplexe Objekte. Durch eine Auswertung der Punktwolken, welche die Intel Realsense erzeugt würde Loomo Objekte wesentlich besser erkennen und umfahren können. Um diese Analyse mit in das System zu inkludieren müsste man allerdings den gesamten Fahralgorithmus neu schreiben.

Quellen:

1. Segway Robotics Developer Portal
https://developer.segwayrobotics.com/developer/documents/segway-robot-overview.html
(13.11.2018 12:15)

2. Robotics on the agenda for CES 2018
https://www.ctvnews.ca/sci-tech/robotics-on-the-agenda-for-ces-2018-1.3660628
(13.11.2018 12:16)