

SEMESTERARBEIT

Steuerung eines Roomba Roboters mit Hilfe
von MS Kinect

Autoren: André Reumer, Yves Boillat

Betreuer: Prof. Reto Bonderer

Aufgabenstellung: SA-ESW11.02

Themengebiet: Elektrotechnik, Embedded Software Engineering

23. Dezember 2011



Abstract

Der erste Teil der Arbeit ist eine Untersuchung der Frameworks „Kinect for Microsoft“ und „OpenNI“. Die beiden Frameworks werden einander gegenüberstellt und in verschiedenen Aspekten verglichen. Es zeigte sich, dass OpenNI für den zweiten Teil der Arbeit geeignet ist.

Dieser Teil sieht vor, einen autonomen Roboter zu entwickeln. Als Ausgangslage stehen ein iRobot Roomba, eine Kinect-Kamera, sowie ein Notebook zur Verfügung. Diese Teile wurden mithilfe eines Aufbaus auf dem Roomba verbunden (RooKi genannt) und eine C#-Applikation entwickelt, die verschiedene Funktionen bietet.

Darunter befinden sich eine Gestensteuerung, einen Modus in dem RooKi selbstständig Hindernissen ausweicht und einen zweiten Modus, welcher den RooKi dazu veranlasst eine Person zu verfolgen.

Während der Arbeit hat sich gezeigt, dass Kinect sich gut für 3D-Anwendungen eignet, sofern nicht exakte Tiefenwerte benötigt werden. Jedoch ist zu beachten, dass sie nur mit Kunstlicht richtig arbeitet und Sonnenlicht ihre Funktionen massgeblich beeinträchtigt. Auch die Erkennungsalgorithmen der Kamera (Skelett-Erkennung) arbeiten besser in stationären Anwendungen. Sie lassen nur langsame Bewegungen zu, ohne Fehlverhalten aufzuweisen.

Inhaltsverzeichnis

1. Einleitung	9
2. Zeitplan	10
3. Schnittstelle Roomba	15
3.1. Spezifikation Schnittstelle	15
3.2. Sensoren	15
3.3. Betriebsmodi	16
3.4. Befehle	16
4. Untersuchung Kinect	17
4.1. Aufbau	17
4.2. Ausrichtung	17
4.3. IR-Bild	18
4.4. Einschränkungen	19
4.4.1. Schatten	19
4.4.2. Nullband	20
4.4.3. Lichteinfluss	20
5. Frameworks	22
5.1. Kinect for Microsoft	22
5.1.1. Architektur	22
5.2. OpenNI	23
5.3. Untersuchungen	24
5.3.1. Minimaldistanz	25
5.3.2. Maximaldistanz	26
5.3.3. Distanzmessung	27
5.3.4. Erfassungswinkel	30
5.3.5. Winkelsehschärfe	30
5.3.6. Materialabhängigkeiten	31
5.3.7. Stabilität	32
5.3.8. Gestenlimitationen	32
5.3.9. Motor	33
5.3.10. Audio	34
5.3.11. Fazit	34
5.4. Entscheidung	36
5.5. Aufbau von OpenNI	36
5.5.1. Context	36

5.5.2. Nodes	36
5.5.3. Meta Data Objects	37
5.5.4. Relevante Nodes	37
5.5.5. Features	37
5.5.6. Daten generieren und lesen	38
5.5.7. Gesten erkennen	38
5.5.8. Skeletterkennung	38
6. RooKi	39
6.1. Systembeschreibung	39
6.2. Betriebsverhalten	40
6.3. State Machine	41
6.3.1. Idle State	44
6.3.2. Auto Clean und Gesture Detection State	44
6.3.3. Skeletal Tracking State	47
6.3.4. Manual State	49
6.3.5. Return to Dock State	51
6.4. Recorder / NiViewer	51
6.5. Klassen	52
6.5.1. NuiSensor	52
6.5.2. DriveCtrl	53
6.5.3. Roomba	53
6.5.4. MyTimer	53
6.6. GUI	55
6.7. Systemvalidierung	56
6.7.1. FSM	56
6.7.2. Roomba	57
6.7.3. Nui Sensor	57
6.7.4. Auto Clean State	57
6.7.5. Manual State	58
6.7.6. Skeletal Tracking State	58
7. Projekt Review	60
A. CD Inhalt	61
B. Dokumente	61
C. Messungen	72
C.1. Messwerte Distanzmessungen Kinect for Microsoft	72

C.2. Messwerte Distanzmessungen OpenNI	72
D. Roomba	75
E. Verzeichnisse	76
Abbildungsverzeichnis	77
Tabellenverzeichnis	77
Codeverzeichnis	77

1. Einleitung

Ende des Jahres 2010 brachte Microsoft ein beeindruckendes Produkt auf den Markt. Dabei handelt es sich um Kinect. Dies ist eine 3D-Kamera, welche primär für die Xbox 360 konzipiert wurde. Das Besondere an einer 3D-Kamera ist, dass jedem Pixel kein Farbwert, sondern eine Distanz zugeordnet ist. Diese Art Kamera ist schon länger auf dem Markt erhältlich, ist sehr präzise, aber auch enorm teuer. Kinect verwendet jedoch ein ganz anderes Verfahren, welches 3D-Kameras erschwinglich macht.

Kinect enthält neben der monochromen Kamera zur Bestimmung der Distanz noch eine Farbkamera und vier Mikrofone für Aufnahmen und Sprachbefehle. Des Weiteren ist durch diese Anordnung sogar eine Lokalisierung der Geräuschquelle möglich. Auf dieses Feature wird jedoch in dieser Arbeit eher weniger eingegangen.

In dieser Arbeit soll primär Kinect untersucht und auf die Tauglichkeit für weitere Arbeiten und Anwendungen geprüft werden. Dazu wird die Kamera auf ihre physikalischen Eigenschaften untersucht. Anschliessend werden zwei Frameworks untersucht und miteinander verglichen. Aufgrund dieser Erkenntnisse wird ein Framework ausgesucht. Anschliessend wird ein Roboter entwickelt, welcher sich autonom bewegen, sowie durch Bewegungen (Gesten) gesteuert werden kann. Mit dieser Entwicklung soll ein detaillierter Einblick in die Anwendungsmöglichkeiten der Kinect erbracht werden.

2. Zeitplan

Das Projekt sieht als ersten Schritt vor, eine Untersuchung von Kinect und den beiden Frameworks durchzuführen. Anschliessend soll eine Software spezifiziert und implementiert werden. Diese Arbeitspakete wurden weiter aufgegliedert und in folgendem Projektstrukturplan definiert. Die Aufgliederung der Arbeitspakte ist im Zeitplan am Ende des Kapitels ersichtlich.

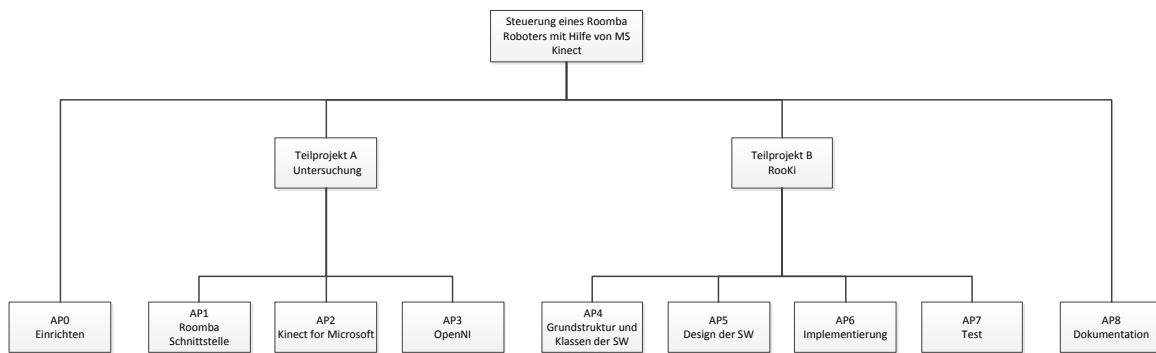


Abbildung 2.1: Projektstrukturplan

Des Weiteren wurden verschiedene Meilensteine definiert, welche auch eingehalten werden konnten. Die Architektur erfuhr während der Entwicklung noch kleine Korrekturen, ist jedoch in seiner angedachten Form implementiert worden. Während den RobOlympics konnte die State Machine mit den *Manual State*-Befehlen, sowie dem *Skeletal Tracking State* gezeigt werden. Die Gesten hatten aber noch Überarbeitung nötig. Der *Auto Clean State* war noch nicht implementiert.

SOLL-DATUM	IST-DATUM	BESCHREIBUNG
17.10.2011	17.10.2011	Erfassung der Leistungsfähigkeit von Kinect abgeschlossen und es liegt ein begründeter Entscheid für Kinect for Microsoft oder OpenNI vor.
24.10.2011	23.10.2011	Grundstruktur und Klassen der Software sind definiert.
31.10.2011	29.10.2011	Die Software-Architektur (Design) steht fest und soll in der Besprechung freigegeben werden. Nach der Freigabe kann mit der Implementation begonnen werden.
19.11.2011	19.11.2011	Die entwickelte Software soll an den RobOlympics demonstriert werden.
23.12.2011	23.12.2011	Abgabe Bericht, Ende der Arbeit.

Tabelle 2.1: Meilensteine

Zeitplan

SA-ESW11.02
André Reumer (AR), Yves Boillat (YB)

Task	VA	September 2011												Oktober 2011											
		Soll	Ist	Meilenstein										Sa, 1	So, 2	Mo, 3	Di, 4	Mi, 5	Do, 6	Fr, 7	Sa, 8	So, 9	Mo, 10	Di, 11	Mi, 12
AP0: Einrichten	AR, YB																								
Pflichtenheft	AR, YB																								
AP1: Untersuchung der Roomba Schnittstelle Roomba-Hacks analysiert Level-Shifter neu gebaut Untersuchung Schnittstelle Klasse schreiben	AR																								
AP2: Untersuchung der MS Kinect SDK Dokumentation / Sample-Codes studiert Testprogramm schreiben Messungen Fazit Kinect SDK / OpenNI schreiben	YB																								
AP3: Untersuchung der OpenNI Plattform Inbetriebnahme Kinect & OpenNI (+NITE) Dokumentation studiert Messungen	AR																								
AP4: Grundstruktur und Klassen der SW Features der SW definiert, Flussdiagramm State Chart definiert	AR, YB																								
AP5: Design der SW Boost Library evaluieren State Chart Klassendiagramm	AR, YB AR, YB AR, YB AR, YB																								
AP6: Implementierung Roomba-Klasse FSM GUI Timer-Klasse Rooki-Aufbau Manual-State NuSensor-Klasse Pegelwandler repariert SkeletalTracking-Funktion AutoClean-Funktion Recorder	AR, YB AR AR YB AR AR YB AR AR YB AR																								
Demonstration an Robolympics	AR, YB																								
AP7: Test Rooki-Notebook AutoClean-Funktion SkeletalTracking-Funktion Recorder	AR, YB YB AR, YB AR AR																								
AP8: Dokumentation LaTeX Inbetriebnahme Dokumentation Help File Präsentation	AR, YB																								
Abgabe	AR, YB																								

Zeitplan

SA-ESW11.02

André Reumer (AR), Yves Boillat (YB)

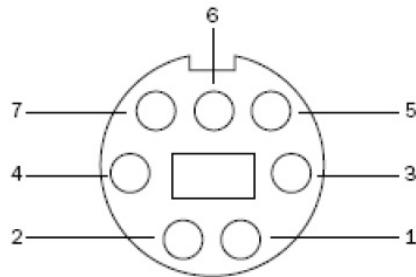
3. Schnittstelle Roomba

3.1. Spezifikation Schnittstelle

Roomba verfügt über eine serielle Schnittstelle, mit der er vollständig kontrolliert werden kann. Zusätzlich können alle Sensoren ausgelesen werden.

- Anschluss: Female Mini-DIN-7
- Spannungspiegel: TTL
- Signal: 1 Startbit, 8 Databits, no parity, 1 Stopbit, 115200 baud (default)
- Verbindung: PC - USBtoRS-232 Konverter - Levelshifter - Roomba

Als Alternative zur Kombination von Levelshifter und USB-Konverter bietet iRobot auch ein Kabel an, welches beide Funktionen bereits vereint und ebenfalls einen virtuellen COM-Port zur Verfügung stellt. Ein weiterer Anschluss ermöglicht direkte Verbindung mit dem Akku. Für die Arbeit wurde ein Levelshifter mit dem MAX3232CWE selber gebaut und mit einem RS-232 zu USB Konverter mit dem Notebook verbunden.



Pin	Name	Description
1	Vpwr	Roomba battery + (unregulated)
2	Vpwr	Roomba battery + (unregulated)
3	RXD	0 – 5V Serial input to Roomba
4	TXD	0 – 5V Serial output from Roomba
5	BRC	Baud Rate Change
6	GND	Roomba battery ground
7	GND	Roomba battery ground

Abbildung 3.1: Pinbelegung Roomba

3.2. Sensoren

Roomba verfügt über ein Set von Sensoren, die entweder einzeln oder zusammen abgerufen werden können. Zu den eher wichtigen zählen: Bumper, Ladezustand des Akkus, Geschwindigkeit

und Kurvenradius. Dazu kommen noch viele eher nebensächlichere wie Temperatur des Akku, Taster die gedrückt werden oder ob Roomba gerade seinen Akku lädt. Eine vollständige Liste der verfügbaren Sensordaten kann dem Abschnitt D entnommen werden.

3.3. Betriebsmodi

Roomba lässt sich in vier Betriebsmodi betreiben. Zu Beginn befindet er sich im *Off Modus* und hört den Bus mit seiner default Baudrate ab. Sobald ein Startkommando erfolgt ist, kann durch einen Befehl einer dieser drei Modi betreten werden.

Passiv Mode: Nach einem Start- oder Putzmodus Kommando (z.B. clean, spot, seek dock) wird dieser Modus betreten. Es ist hier nur möglich, Roombas Sensoren auszulesen oder ihn zu unterbrechen.

Safe Mode: Der User erhält volle Kontrolle über Roomba mit der Ausnahme von folgenden, sicherheitsrelevanten Umständen: Detektion eines abrupten Abfallen des Geländes (z.B. Treppe), ein Rad berührt den Boden nicht mehr oder Ladekabel ist eingesteckt. Sollte einer dieser Fälle eintreten, stellt Roomba alle Aktionen ein und kehrt in den *Passiv Mode* zurück. Roomba wird in diesem Projekt nur in diesem Modus betrieben, damit die Sicherheitsfunktionen so genutzt werden können.

Full Mode: Der User erhält volle Kontrolle über Roomba. Die sicherheitsrelevanten Sensoren können ausgelesen werden, Roomba reagiert aber nicht automatisch.

3.4. Befehle

Eine Liste aller verwendeten Befehle für den Umgang mit Roomba.

- **Start:** Startet das User Interface.
- **Safe:** Versetzt Roomba in den Safe Mode.
- **Power:** Schaltet Roomba aus.
- **Dock:** Sendet Roomba zu seiner Docking Station.
- **Drive:** Direkter Fahrbefehl an Roomba mit Geschwindigkeit und Drehradius.
- **Motors:** Bürsten und Vacuum steuern.
- **Sensors:** Liest die gewählten Sensordaten aus.
- **Song:** Ermöglicht die Definition bzw. Speicherung eines Songs auf dem Roomba.
- **Play:** Spielt einen Song ab.

4. Untersuchung Kinect

4.1. Aufbau

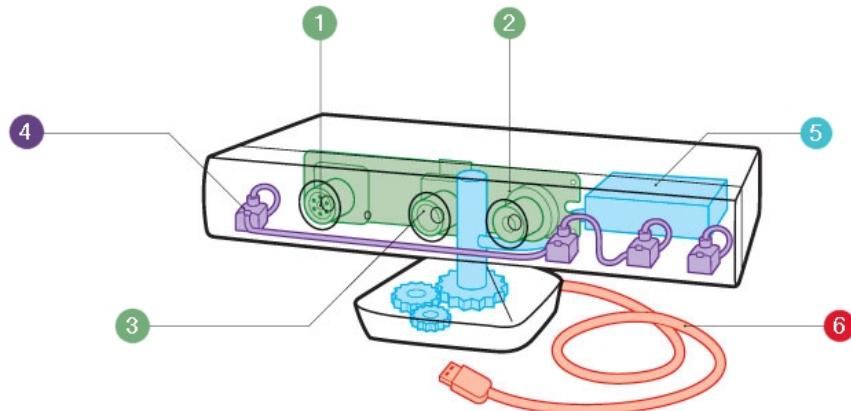


Abbildung 4.1: Kinect Aufbau

1. IR-Emitter
2. IR-Kamera
3. RGB-Kamera
4. Mikrofon-Array
5. Tilt-Motor (Zur Einstellung des vertikalen Neigungswinkels)
6. USB-Kabel mit 12VDC (Adapter 12VDC 1.08A)

4.2. Ausrichtung

Vertikaler Neigungswinkel

Um den vertikalen Neigungswinkel der Kamera zu verändern, ist ein Motor verbaut, den man softwaretechnisch ansteuern kann. Der Neigungsbereich reicht von -27° bis $+27^\circ$, wobei 0° der Horizontallage entspricht.

Horizontaler Drehwinkel

Die horizontale Position lässt sich nur mechanisch von Hand verändern.

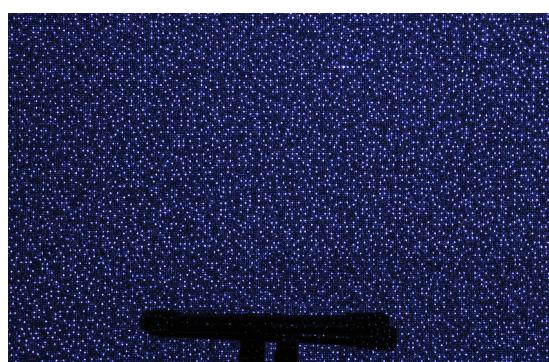
4.3. IR-Bild

Im Gegensatz zu den bis anhin erhältlichen Systemen verwendet Kinect eine andere Technik zur Tiefenbestimmung. Herkömmliche Systeme messen die Zeit, die ein IR-Lichtstrahl von der Kamera bis zum Objekt und wieder zurück benötigt (time-of-flight). Dies verlangt eine sehr genaue Zeitmessung, was das System sehr teuer macht. Kinect verwendet hingegen Light Coding.

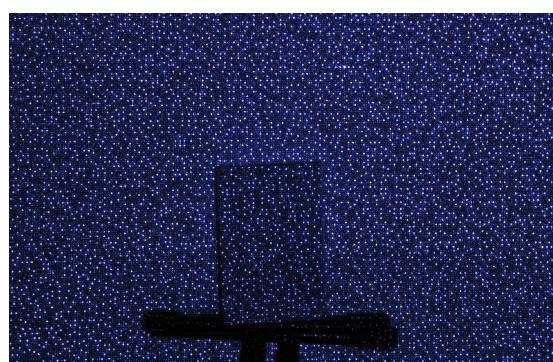


Abbildung 4.2: Kinect IR-Bild

Bei der Kinect ist die Lichtquelle konstant eingeschaltet und sendet ein nicht moduliertes IR-Pattern aus. Das Pattern (Abb. 4.2) welches die Kamera projiziert besteht aus einer drei Mal drei Matrix. Die IR-Kamera erfasst das Pattern. Anschliessend wird das IR-Bild mit einem hardcoded IR-Bild korreliert und somit die Tiefenwerte ermittelt.



(a) Ohne Objekt



(b) Mit Objekt

Abbildung 4.3: IR-Bild Ausschnitt

4.4. Einschränkungen

4.4.1. Schatten

Je nach Distanzunterschied zweier Objekte zueinander ergibt sich im Tiefenbild einen Schatten. Dies ist gut in Abb. 4.4 zu sehen. Die Ursache für diesen Effekt findet man im örtlichen Versatz des IR-Emitters und der IR-Kamera. Durch den Versatz der IR-Kamera zum IR-Emitter kann es vorkommen, dass die IR-Kamera Bereiche sieht (Abb. 4.5), welche für den IR-Emitter durch ein Objekt (Obstacle 1) verdeckt wird.

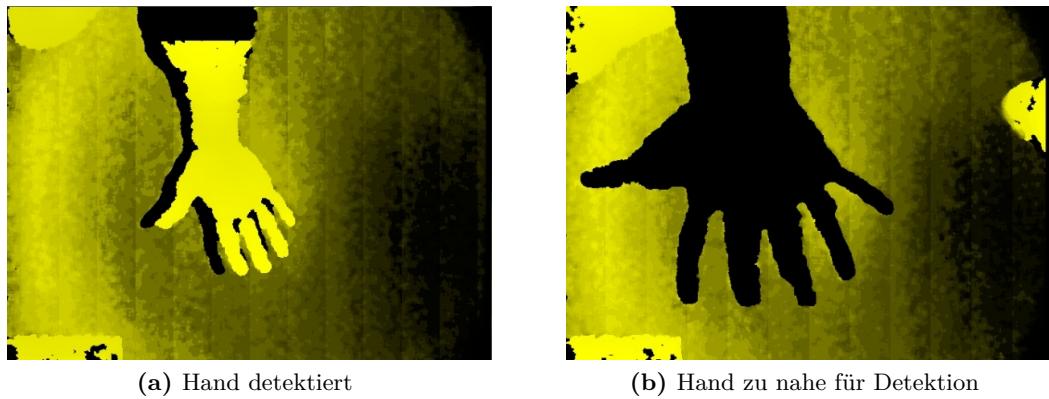


Abbildung 4.4: Schattenbildung im Tiefenbild

Seen by the camera but no pattern visible = no depth information

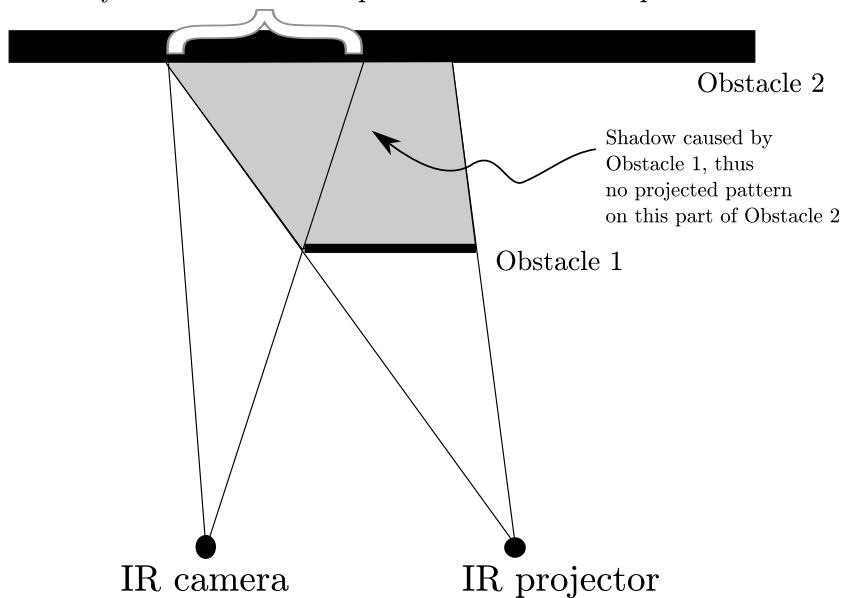


Abbildung 4.5: Schattenbildung

4.4.2. Nullband

Jedes Tiefenbild mit der Auflösung 640x480 hat am rechten (nicht gespiegelt) beziehungsweise linken (gespiegelt) Bildrand ein Nullband. Darunter versteht man einen 8 Pixel breiten Streifen am Bildrand, der keine Tiefenwerte liefert (Abb. 4.6). Dies könnte von der Korrelationsmethode herrühren.¹



Abbildung 4.6: Nullband

4.4.3. Lichteinfluss

Licht, welches im Spektrum keinen Infrarotanteil besitzt, beeinflusst die Tiefenmessung in keiner Weise. Die Messwerte bleiben unverändert wenn es dunkel ist oder Kinect direkt mit einer Taschenlampe beleuchtet wird.

Werden zwei Kinect aufeinander gerichtet, so entsteht im Tiefenbild ein blinder Fleck um die gegenüberliegende Kinect, in welchem kein Tiefenwert bestimmt werden kann. Das restliche Tiefenbild rauscht etwas stärker, vor allem Konturen werden schlechter erkannt.

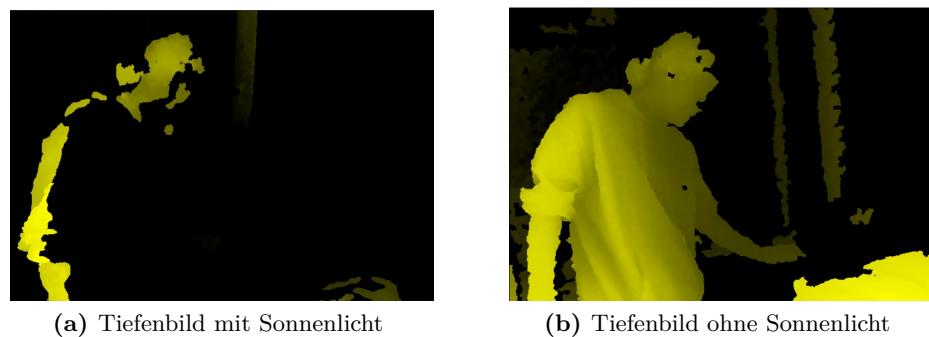
Probleme entstehen, wenn zu messende Gebiete mit Sonnenlicht bestrahlt werden. In diesem Fall lässt sich nichts erkennen. Im Freien bei Sonnenschein können grundsätzlich keine Distanzen bestimmt werden, da der Infrarotanteil der Sonne das IR-Pattern der Kinect komplett überdeckt und dieses nicht moduliert ist.

Der Einfluss der Sonne wurde an einem Fenster betrachtet, welches direkten Lichteinfall hatte (Abb. 4.7). Wird die Kamera nun auf Höhe des Fensters bei offenem Rollladen auf einen Mensch gerichtet, ist nur sehr wenig zu erkennen. Lediglich die der Sonne abgewandte Seite kann detektiert werden. Im Vergleich ist bei heruntergelassenen Rollläden wieder alles erkennbar. (Abb. 4.8)

¹Quelle: http://www.ros.org/wiki/kinect_calibration/technical



Abbildung 4.7: Messung des Lichteinfluss vor dem Fenster

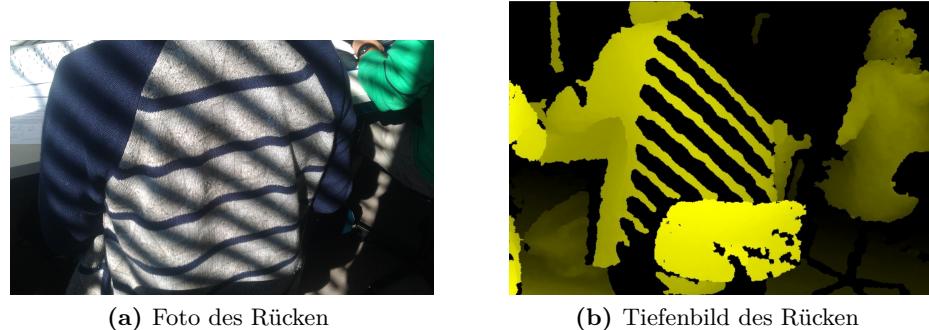


(a) Tiefenbild mit Sonnenlicht

(b) Tiefenbild ohne Sonnenlicht

Abbildung 4.8: Vergleich mit und ohne Sonnenlicht

Dieser Effekt lässt sich gut anhand von Abb. 4.9 erkennen. Der Rücken einer Person wird mit Sonnenlicht beschienen. Auf dem Rücken bilden sich Streifen mit Sonnenlicht, welche im Tiefenbild den Wert null haben.



(a) Foto des Rücken

(b) Tiefenbild des Rücken

Abbildung 4.9: Demonstration des Sonnenlichteinflusses

5. Frameworks

5.1. Kinect for Microsoft

Kinect for Microsoft ist momentan nur als Beta Version erhältlich.

Für Kinect for Microsoft wird zwingend eine Windows 7 Version vorausgesetzt.

Zusätzlich benötigte Software:

- Microsoft .NET Framework 4
- Visual Studio 2010 (Express)
- Kinect for Microsoft

Zusätzlich benötigte Software für Skelett Applikationen:

- DirectX Software Development Kit (June 2010 oder neuer)
- DirectX End-User Runtime

Zusätzlich benötigte Software für Audio Applikationen:

- Microsoft Speech Platform - Server Runtime, version 10.2 (x86 edition)
- Microsoft Speech Platform - Software Development Kit, version 10.2 (x86 edition)
- Kinect for Windows Runtime Language Pack, version 0.9

5.1.1. Architektur

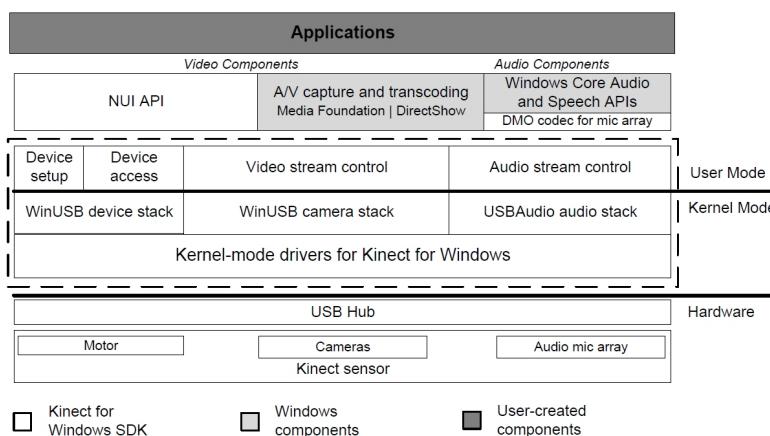


Abbildung 5.1: Kinect for Microsoft Architektur

5.2. OpenNI

OpenNI ist ein cross-platform Framework, welches erlaubt, Anwendungen zu schreiben, die NI (Natural Interaction) beinhalten. NI ist ein Konzept, welches auf den menschlichen Sinnen beruht. Zumeist Sehen und Hören. OpenNI ist Open Source und verfolgt das Ziel, die Entwicklung solcher Anwendungen zu standardisieren, erleichtern und zu beschleunigen.

OpenNI ist nicht nur mit Kinect kompatibel. Es können auch andere Geräte verwendet werden. OpenNI übergibt seine Daten an Middleware Components (z.B. NITE), welche dann die gewünschten Informationen aus den Rohdaten errechnen, wie zum Beispiel eine Geste.

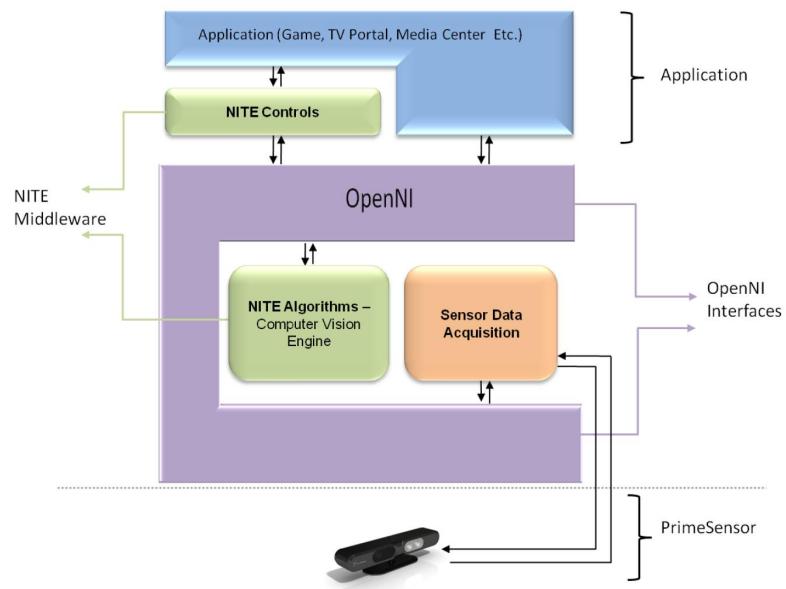


Abbildung 5.2: Struktur OpenNI

5.3. Untersuchungen

Kinect for MS:

Für die Untersuchung von Kinect for Microsoft wurde ein Testprogramm entwickelt (TestGUI). Alle Messungen wurden mit dem TestGUI durchgeführt.

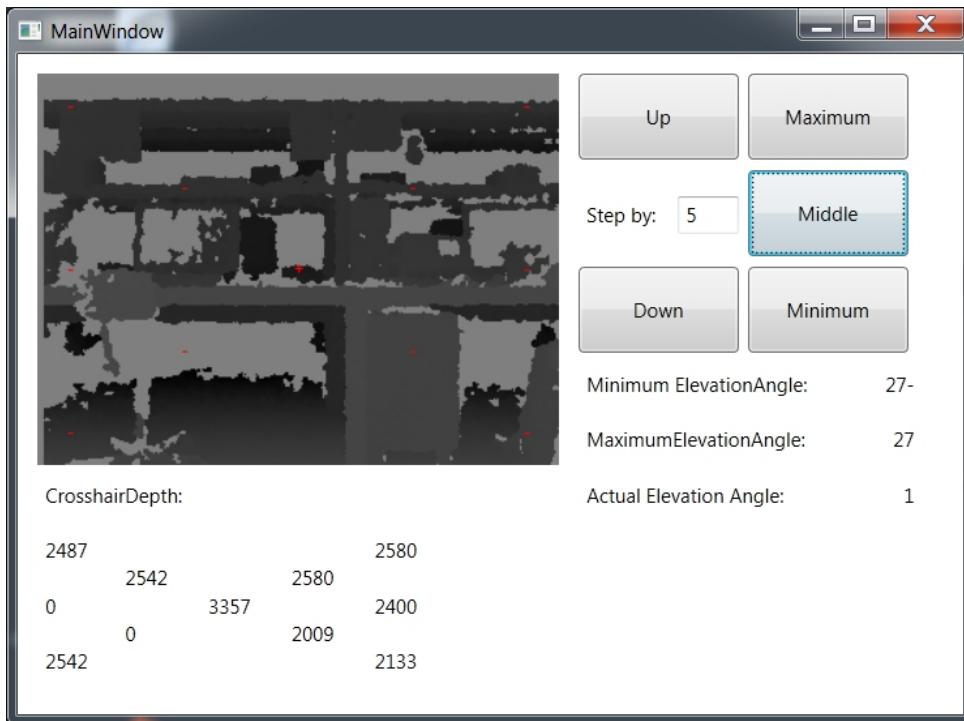


Abbildung 5.3: TestGUI

Die roten Markierungen im Tiefenbild bezeichnen die Positionen, an welchen die Tiefenwerte unten aufgelistet werden. Die Tiefenwerte sind im gleichen Muster wie die Markierungen angeordnet.

Mit den Buttons „Up“ und „Down“ kann die Kamera um den Wert im Feld „Step by:“ nach oben beziehungsweise nach unten verstellt werden.

BUTTON	ELEVATION ANGLE
Up	Schritt nach oben
Down	Schritt nach unten
Maximum	27° (nach oben)
Middle	0° (Horizontal)
Minimum	-27° (nach unten)

Tabelle 5.1: Elevation Angle der Buttons

OpenNI:

Für die Untersuchungen des Frameworks wurde das Sample Programm *SimpleViewer* angepasst. Mit einem Mausklick kann ein Ort im Bild festgelegt werden, dessen Tiefenwert kontinuierlich in der Konsole ausgegeben wird. Zusätzlich wurde das Pattern von Abschnitt 5.3.3 implementiert und die Werte in der Konsole ausgegeben.

5.3.1. Minimaldistanz

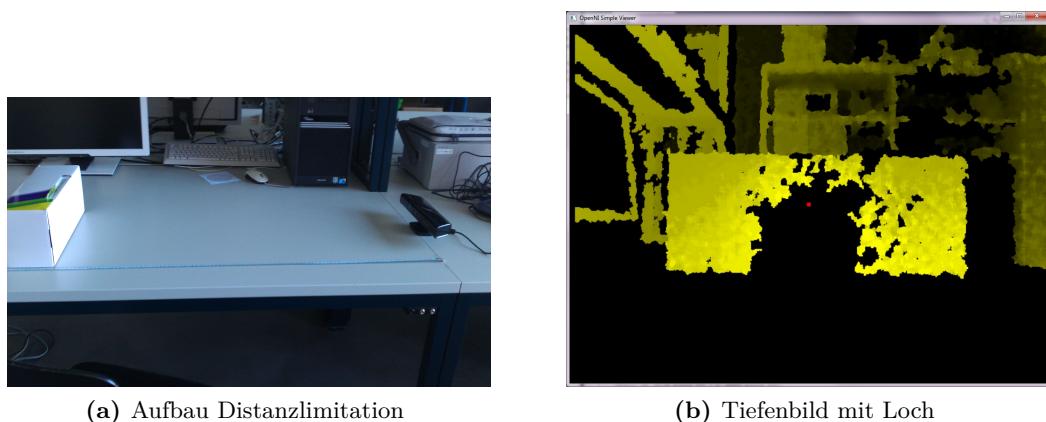


Abbildung 5.4: Messung der Minimaldistanz

Gut zu erkennen ist, dass in der Mitte ein Loch (Abb. 5.4b) entsteht, welches schwarz ist. Dies bedeutet, dass dort der Wert 0 gemessen wird, also nicht mehr messbar. Dieses vergrößert sich, wenn man das Objekt näher an die Kamera schiebt. Dies liegt vermutlich daran, dass die Kamera bei grösserem Betrachtungswinkel mehr bzw. wieder genügend IR-Punkte sieht.

Kinect for MS:

Die Minimaldistanz wurde an verschiedenen, ebenen Oberflächen gemessen.

MATERIAL	MINIMALDISTANZ
Spiegel	Nicht detektierbar, Verfälschung durch gespiegelte Objekte
Glas	Nicht detektierbar
Metall	801mm (Blinder Fleck in der Mitte)
Holz	801mm
Kunststoff (glatt)	801mm
Textilien	801mm
Karton/Papier	801mm
Flache Hand	801mm

Tabelle 5.2: Minimaldistanzen Kinect for Microsoft

Die Minimaldistanzen unterscheiden sich prinzipiell nicht. Dies röhrt daher, dass die Minimaldistanz bei Kinect for Microsoft softwaretechnisch auf 800mm begrenzt wurde.

Spiegel und Glas sind nicht detektierbar oder verfälschen das Bild sogar durch gespiegelte Objekte. Bei Metall entsteht in der Mitte ein blinder Fleck (Abb. 5.4b).

OpenNI:

Für die minimale Distanz wurden verschiedene, glatte Oberflächen geprüft. Die Messwerte wurden allesamt dem Output von Kinect bezogen. Die aufbaubedingten (Abb. 5.4a) Messfehler sind viel grösser als die Messfehler der Kamera. Hierzu wurde der modifizierte *SimpleViewer* verwendet, welcher das Tiefenbild ausgibt und die Tiefenwerte in der Konsole anzeigt.

MATERIAL	MINIMALDISTANZ
Spiegel	Nicht detektierbar, Verfälschung durch gespiegelte Objekte
Glas	Nicht detektierbar
Metall	500mm
Holz	490mm
Kunststoff (glatt)	460mm
Textilien	500mm
Karton/Papier	490mm
Flache Hand	490mm

Tabelle 5.3: Minimaldistanzen OpenNI

Unter den verschiedenen Materialen gibt es nur feine Unterschiede. Man kann also gesamthaft von einer Minimaldistanz von 500mm ausgehen.

5.3.2. Maximaldistanz

Kinect for MS:

Die Maximaldistanz wurde durch stetiges Vergrössern des Abstandes der Kamera zu einer Wand ermittelt. Wie in der Spezifikation von Kinect for Microsoft endet der Messbereich bei 4m. Der höchste Wert, der noch ermittelt werden konnte beträgt 3.98m bei einer effektiven Distanz von 3.80m.

Bei 3.8m kann man eine Streuung der Messwerte von 265mm beobachten. Die genauere Betrachtung ist im Abschnitt 5.3.3 zu finden.

OpenNI:

Für eine erste Distanzbetrachtung wurde eine 8.2m entfernte Wand ausgemessen. Dafür wurde die Kamera im Labor etwas höher positioniert und auf die Wand gerichtet (Abb. 5.5a). Aus dem resultierenden Tiefenbild (Abb. 5.5b) ist ersichtlich, dass nahe Objekte gut erkannt werden, der Switch an der Wand jedoch nicht mehr von der Wand unterschieden werden kann. Die Messwerte an der Wand haben eine Streuung von rund 1.5m. Genauere Werte sind dem Abschnitt 5.3.3 zu entnehmen.

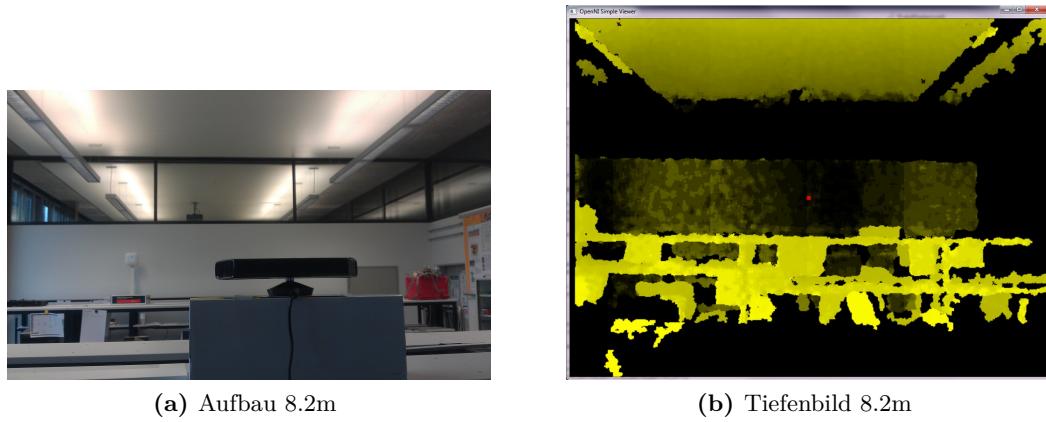


Abbildung 5.5: Messung über weitere Distanzen

5.3.3. Distanzmessung

Die Kamera wurde auf verschiedene Distanzen gegen eine grosse, weisse Wand gerichtet. Anschliessend wurde die Distanz bei verschiedenen Punkten im Bild (Abb. 5.6) gemessen. Die Resultate sind dem Abschnitt C.1 (Kinect for Microsoft) beziehungsweise Abschnitt C.2 (OpenNI) zu entnehmen.

Aufgrund dieser Daten wurden Mittelwert und Standardabweichung berechnet.

DISTANZ [m]	0.8	1	2	3	3.8
MITTELWERT [mm]	814	1012	2024	3063	3874
STANDARDABWEICHUNG [mm]	6.14	5.74	15.3	49.9	94.1

Tabelle 5.4: Mittelwerte und Standardabweichungen Kinect for Microsoft

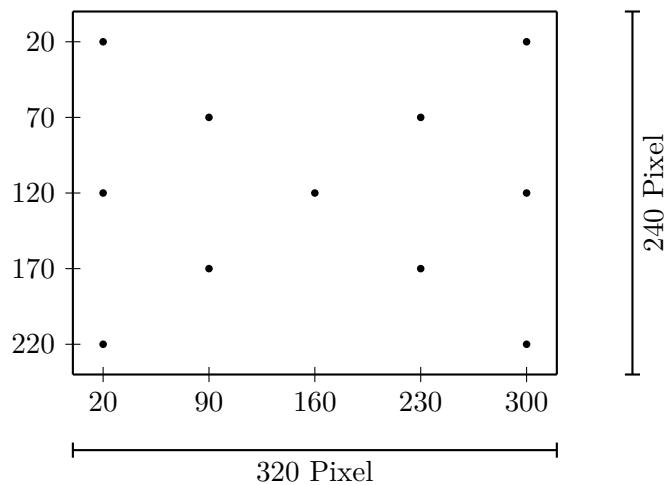


Abbildung 5.6: Anordnung der Messpunkte

DISTANZ [m]	0.52	1	2	3	4	5	6
MITTELWERT [mm]	527	1006	1986	2997	3975	5004	6084
STANDARDABWEICHUNG [mm]	12	22	35	72	128	202	233

Tabelle 5.5: Mittelwerte und Standardabweichungen OpenNI

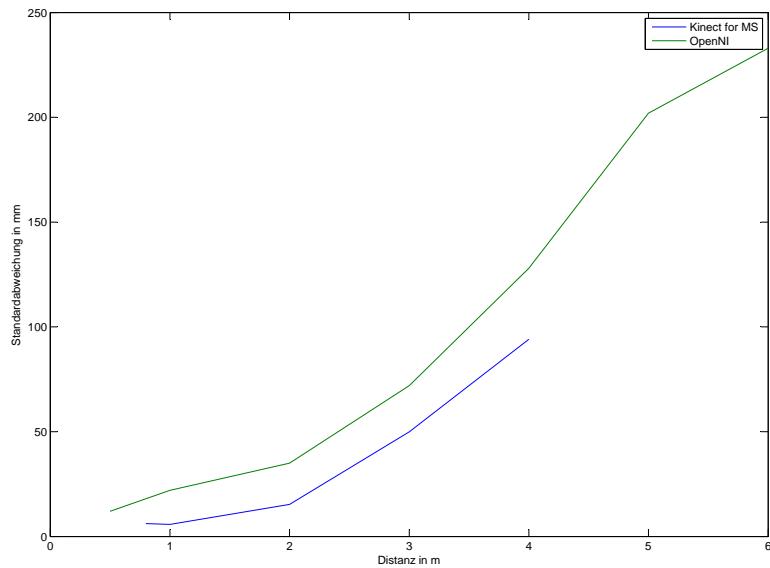


Abbildung 5.7: Standardabweichung der Tiefenwerte

Im späteren Verlauf wurde die Tiefenmessung anhand des Tools NiViewer und des Recorders von OpenNI (siehe Abschnitt 6.4) wiederholt. Dabei wurden wieder die Tiefenwerte aufgenommen, wenn die Kamera gegen eine Wand gerichtet wird. Dieses Mal wurden aber sämtliche Werte in MATLAB importiert. Dabei wurden Bilder der Abweichung vom Mittelwert erstellt (siehe Abschnitt C.2). Des Weiteren wurde mit MATLAB die Standardabweichungen von 0.5m bis 8m berechnet.

DISTANZ [m]	0.54	1	2	3	4	5	6	7	8
MITTELWERT [mm]	540	1005	1981	2974	3959	4760	5716	6786	7884
STANDARDABWEICHUNG [mm]	3.88	3.7	14.7	28.5	60.4	96.4	150	211	296

Tabelle 5.6: Mittelwert und Standardabweichung OpenNI über ganze Map

Nach 8m werden die Bilder undeutlich und mit vielen Pixeln, denen kein Tiefenwert mehr zugeordnet wird (Wert 0). Aus den Tiefenbildern im Anhang und dem Vergleich der beiden Messungen ist zu erkennen, dass die Tiefenwerte, die sich am Bildrand befinden, mit mehr Fehler behaftet sind als jene in der Mitte. Die Standardabweichung ist tiefer, wenn alle Punkte einbezogen werden. In der Betrachtung davor wurde lediglich ein Punkt von der Mitte und viele am Rand in die Rechnung einbezogen. Die absolute Distanz in der Mitte des Bildes ist aber auch auf weite Distanz sehr genau.

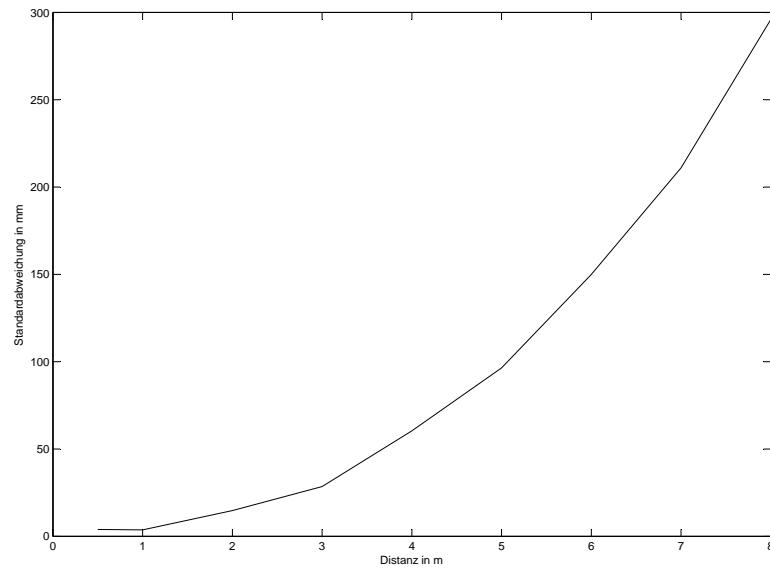


Abbildung 5.8: Standardabweichung der Tiefenwerte

5.3.4. Erfassungswinkel

Horizontal: Hierzu wurde die Kamera so lange an eine $b = 1.00m$ breite Platte heran geschoben, bis sich die Kanten des Bildes mit denen der Platte überdeckten. Diese Distanz nennen wir d .

$$\text{Erfassungswinkel} = 2 * \tan^{-1}\left(\frac{b}{2 * d}\right)$$

Vertikal: Hierzu wurde die Kamera in der Distanz d von der Platte entfernt und der vertikale Erfassungsbereich b bestimmt.

$$\text{Erfassungswinkel} = 2 * \tan^{-1}\left(\frac{b}{2 * d}\right)$$

Kinect for MS:

Horizontal: $d = 916mm$

$$2 * \tan^{-1}\left(\frac{1000}{2 * 916}\right) = 57.3^\circ$$

Vertikal: $d = 919mm, b = 730mm$

$$2 * \tan^{-1}\left(\frac{730}{2 * 919}\right) = 43.3^\circ$$

OpenNI:

Horizontal: $d = 910mm$

$$2 * \tan^{-1}\left(\frac{1000}{2 * 910}\right) = 57.6^\circ$$

Vertikal: $d = 910mm, b = 750mm$

$$2 * \tan^{-1}\left(\frac{750}{2 * 910}\right) = 44.8^\circ$$

5.3.5. Winkelsehschärfe

Die Untersuchungen haben ergeben, dass sich die Winkelsehschärfe nicht so einfach wie beispielsweise bei dem menschlichen Auge bestimmen lässt, da nicht nur die Breite des Gegenstandes ein Faktor ist, sondern auch die Fläche. So wird ein Objekt mit bestimmter Breite besser erkannt, wenn es fünfmal so lang ist, als wenn es quadratisch ist. Somit wurden zwei verschiedene Messungen durchgeführt, um einen Anhaltspunkt zu geben.

Der erste Gegenstand hat eine glatte Oberfläche und quadratische Form mit Seitenränder von $b = 50mm$. Dieser Gegenstand wird bis zu einem Abstand d zuverlässig erkannt.

$$\text{Winkelwahrnehmung} = 2 * \tan^{-1}\left(\frac{b}{2 * d}\right)$$

Der zweite Gegenstand hat die Länge $346mm$ und eine Breite von $b = 50mm$. Dieser wird bis zu einer Distanz d erkannt.

$$\text{Winkelwahrnehmung} = 2 * \tan^{-1}\left(\frac{b}{2 * d}\right)$$

Kinect for MS:

Gegenstand 1 (50mm x 50mm): $d = 2100mm$

$$2 * \tan^{-1}\left(\frac{50}{2 * 2100}\right) = 1.36^\circ$$

Gegenstand 2 (346mm x 50mm): $d = 3950mm$

$$2 * \tan^{-1}\left(\frac{50}{2 * 3950}\right) = 0.725^\circ$$

OpenNI:

Gegenstand 1 (50mm x 50mm): $d = 2500mm$

$$2 * \tan^{-1}\left(\frac{50}{2 * 2500}\right) = 1.15^\circ$$

Gegenstand 2 (346mm x 50mm): $d = 4800mm$

$$2 * \tan^{-1}\left(\frac{50}{2 * 4800}\right) = 0.597^\circ$$

Somit lässt sich sagen, dass auch dünne Hindernisse erkannt und umfahren werden können.

5.3.6. Materialabhängigkeiten

Kinect for MS und OpenNI:

Wie im Kapitel Minimaldistanz bereits erwähnt, gibt es keine grossen Unterschiede zwischen einzelnen Materialien. Stark spiegelnde Oberflächen können nicht präzise bis gar nicht erfasst werden. Des Weiteren können leicht spiegelnde Oberflächen, welche orthogonal zur Kamera stehen, in einem kleinen Bereich um den Mittelpunkt nicht erkannt werden (Abb. 5.4b). Erhöht

sich die Distanz, tritt dies jedoch nicht mehr auf. Glas ist auch für den Tiefensor durchsichtig. Es wird die Distanz zum Objekt hinter dem Glas angegeben.

5.3.7. Stabilität

OpenNI:

Teilweise verhält sich das System mit dem Sample Programm *SimpleViewer* in C++ instabil (Version: Win32-1.3.2.1). Die Software erhält manchmal nach ca. einer Minute Laufzeit keine Daten mehr von der Kamera. Dieses Problem ist nicht reproduzierbar, da es an manchen Tagen gar nicht auftritt und an anderen regelmässig. Dies ist kein Problem der Kamera, da dies mit einer anderen Kinect ebenfalls passiert. Dieses Problem wurde aber nicht weiter verfolgt, da bei der eigenen Implementation keine Probleme vorlagen.

Des Weiteren ist zumindest bei der zurzeit aktuellen 64 Bit Version (Win64-1.3.4.3) Vorsicht geboten. Es traten Exceptions wegen Zugriffsverletzungen auf *protected memory* auf. Diese konnten nicht erklärt werden. Alle Softwareteile wurden systematisch entfernt, bis nur noch der *Depth Generator* übrig war. Der Fehler blieb jedoch bestehend. Dies ist anscheinend ein bekanntes Problem. Bei Verwendung der 32 Bit Version mit dem genau gleichen Code traten diese Probleme nicht auf.

5.3.8. Gestenlimitationen

Kinect for MS:

Kinect for Microsoft erkennt das Skelett einer Person sobald diese ins Bild tritt. Die Erkennung geschieht automatisch ohne spezielle Anweisungen der Person. Eine Gestenerkennung ist nicht im Framework implementiert und muss selbst programmiert werden. Um Gesten einfacher erkennen zu können, muss das Skelett erkannt werden. Das heisst, die Kamera muss die ganze Person erfassen. Die Erkennung einer Hand gibt es bei Kinect for Microsoft ebenfalls nicht.

OpenNI:

Bei OpenNI gibt es zwei verschiedene Generatoren zur Erkennung von Personen. Einer, welcher ein ganzes Skelett erkennt, wobei eine *Detection Pose* (Abb. 6.8) eingenommen werden muss, und verfolgt und einer für die Hand. Für Gesten muss sich lediglich die Hand im Bild befinden und nicht der ganze Körper. Für die Erkennung von Gesten ist weiter noch ein *Gesture Generator* nötig. Damit kann durch eine von zwei sogenannten *Fokus Gesten* eine Hand erkannt und registriert werden. Diese wird nun kontinuierlich verfolgt. Ist NITE referenziert, stehen weitere Gesten zur Verfügung. Es können aber auch eigene definiert werden.

Fokus Gesten

- Click (Hand vor und zurück)
- Wave (min. 2 Hin- und Her-Bewegungen)

NITE-Gesten

- Wave
- Push (Hand vor)
- Swipe up, down, left, right (Hand nach oben, unten, links, rechts)
- Circle left, right (Kreisbewegung links, rechts)
- Steady (Hand bewegt sich nicht)
- Slider 1D, 2D

Besonders auf nähere Distanzen (~2m) funktionieren die Gesten zuverlässig. Auf dem RooKi montiert werden die Gesten weiter entfernt getätigigt, was die Zuverlässigkeit stark verringert. Es ist möglich einige Parameter der Gesten einzustellen. Jedoch sind diese nicht dokumentiert, was deren Deutung um einiges erschwert. Nach einigen Versuchen konnte eine Verbesserung erzielt werden. Eine genaue Einstellung muss jedoch anwendungsspezifisch erfolgen.

5.3.9. Motor

Kinect verfügt über einen Tilt-Motor, mit dem sich der vertikale Neigungswinkel der Kamera einstellen lässt.

Kinect for MS:

Mit Kinect for Microsoft kann der Tilt-Motor über ein Property verändert und ausgelesen werden.

OpenNI:

Der Tilt-Motor wird von OpenNI nicht unterstützt. Es sind aber Treiber im Internet erhältlich, die über diese Funktionalität verfügen.

5.3.10. Audio

Kinect for MS:

Audio kann mit Kinect for Microsoft nicht nur aufgenommen werden. Es ist auch möglich, die Position der Audioquelle zu bestimmen, sowie mithilfe von Microsofts Speech Platform Spracherkennung durchzuführen. Die Spracherkennung ist bis zu diesem Zeitpunkt nur in Englisch möglich.

OpenNI:

OpenNI verfügt über einen *Audio Generator*. Dieser kann entsprechend den persönlichen Konfigurationen Audio Samples aufnehmen und speichern. Folgende Parameter können festgesetzt werden: Sample Rate, Bits pro Sample und Kanäle. Es ist jedoch nicht möglich, die Audiodaten direkt zu verarbeiten beziehungsweise zu interpretieren. Hierzu wären zusätzliche Bibliotheken notwendig.

5.3.11. Fazit

Kinect for MS:

Kinect for Microsoft gibt es nur für C++ und C#, Visual Studio 2010 (Express) und Windows 7. Dafür beinhaltet es viele Funktionen und ist sehr gut dokumentiert.

Vorteile

- Skeletterkennung ohne *Detection Pose*
- Tilt-Motor implementiert
- Spracherkennung
- Ortsbestimmung einer Audioquelle mittels Mikrofon-Array
- Events für neue Tiefen- oder Bild-Frames
- Sehr ausführliche Dokumentation

Nachteile

- Softwaremässige Begrenzung der Tiefenwerte
- Kein Hand Tracking oder Gestenerkennung implementiert
- Keine Aufnahme- oder Playback-Funktion
- Kein Zugriff auf IR-Bild (raw)
- Plattform-, OS-, IDE-Abhängig

- Starke Beschränkungen durch die Lizenz
- Unterstützt nur die Kinect-Kamera

OpenNI:

Das Framework ist komfortabel in der Programmierung. Alle Teilbereiche sind sauber in Module aufgeteilt, welche bei Bedarf einfach zu initialisieren sind. Zudem lässt sich damit in C, C++, C# und Java programmieren.

Vorteile

- Keine softwaremässige Begrenzung der Tiefenwerte
- Hand Tracking und Gestenerkennung bereits implementiert
- Recorder- / Playback-Funktion
- Zugriff auf IR-Bild (raw)
- Plattformunabhängig
- Frei zur kommerziellen Nutzung
- Unterstützt Sensoren unterschiedlicher Hersteller

Nachteile

- Tilt-Motor nicht implementiert
- Spracherkennung nicht implementiert
- Benötigt eine Kalibrierungspose (*Detection Pose*) für das Skelett
- Keine Events für neue Tiefen- oder Bild-Frames (Polling nötig)
- Dokumentation der Wrapper ist in C++ mangelhaft, in C# nicht verfügbar

Während des Projekts hat sich herausgestellt, dass es Probleme mit Gesten und Skeletten gibt, wenn sich die Kamera bewegt. Es treten in diesem Fall sehr viele Fehldetections von Gesten (hauptsächlich Click oder Push) und fälschlich erkannten Usern auf. Des Weiteren verliert OpenNI das Skelett, wenn sich der User zu schnell bewegt oder zu weit entfernt ist.

5.4. Entscheidung

Nach der Untersuchung der beiden Frameworks wurden sie miteinander verglichen, um eines der beiden für RooKi auszuwählen. Dabei fiel der Entscheid auf OpenNI. Kinect for Microsoft sticht zwar in Punkten Skelettverarbeitung deutlich heraus, OpenNI hat dafür wesentlichere Vorteile in anderen Bereichen. Zu diesen zählen folgende:

- Minimal- und Maximaldistanz nicht softwarebedingt beschränkt.
- Hand Tracking ohne ganzes Skelett möglich.
- Gesten sind bereits implementiert.
- Unabhängigkeit vom .NET Framework. Dies ist für dieses Projekt nicht sehr wichtig. Für zukünftige Projekte mit eingebetteten Systemen ist es aber von Vorteil, schon einige Kenntnisse von OpenNI zu haben.
- Recorder für Debugging und Analysezwecke.

Die Nachteile von OpenNI sind nicht sehr wesentlich für dieses Projekt. Einzig die fehlende Ansteuerung des Motors für den Neigungswinkel wäre sehr nützlich gewesen, aber nicht direkt nötig.

5.5. Aufbau von OpenNI

In diesem Kapitel sollen die wichtigsten Eigenschaften von OpenNI erläutert werden.

5.5.1. Context

Der Context ist ein zusammenfassendes Objekt, welchem ein Sensor und verschiedene Nodes angehören können. Dieser wird immer als erstes erzeugt. Der Context übernimmt die Verknüpfung zwischen verschiedenen Nodes.

5.5.2. Nodes

Die Nodes sind Objekte, welche bestimmte Funktionalitäten übernehmen. Sie lassen sich in drei verschiedene Kategorien unterscheiden. Folgende Tabelle repräsentiert die wichtigsten Nodes jeder Kategorie.

SENSOR RELATED	MIDDLEWARE RELATED	RECORD
Device	Gesture Alert Generator	Recorder
Depth Generator	Scene Analyzer	Player
Image Generator	Hand Point Generator	Codec
IR Generator	User Generator	
Audio Generator		

Sensor related Nodes liefern Daten vom Sensor, welche über eine Meta-Klasse direkt ausgelesen werden können. **Middleware related Nodes** verarbeiten die Daten der Sensor related Nodes, ohne dass sie von Hand verknüpft werden müssen. Mit **Recordern** können alle Inputs auch als Stream aufgenommen und abgespeichert werden. So können zu Debugzwecken diese Daten bzw. Szenarien über *Mock Nodes* in anderen Codes verwendet werden. Diese funktionieren im Prinzip gleich wie die normalen Nodes, nehmen aber ihre Daten von Records, nicht von Sensoren. Neuer Code kann also direkt mit Records reproduzierbar getestet werden.

5.5.3. Meta Data Objects

Jeder Generator verfügt über ein Meta Data Objekt, welches bei Bedarf auch erzeugt werden muss. Dieses kapselt die wichtigen Outputs und Eigenschaften. Folglich werden Daten immer aus diesen Objekten ausgelesen und nicht aus den eigentlichen Generatoren.

5.5.4. Relevante Nodes

Depth Generator: Stellt eine Map bereit, bei welcher jedem Pixel ein Tiefenwert in Millimeter zugewiesen ist.

Scene Analyzer: Dieser Node fügt jedem Pixel eine Information zu, ob dies sich im Vordergrund befindet. Damit können Boden und Gegenstände erkannt werden.

User / Hand Generator: Hand bzw. User wird erkannt und verfolgt. Die Position kann also immer bestimmt werden.

5.5.5. Features

Cropping: Für die Performanceverbesserung kann OpenNI angewiesen werden, nur einen bestimmten Teil des Bildes zu verarbeiten.

Frame Sync: Nötig um mehrere Sensor gleichzeitig zu verwenden.

Mirror: Bild an der y-Achse spiegeln

Detection Pose: Vordefinierte Posen, die erkannt werden. Wird oft benutzt um einen neuen

User anzumelden bzw. eine Fehlinterpretation zu vermeiden.

Skeleton: Koordinaten verschiedener Eckpunkte ("joints") des Users. (Kopf, Schultern, Ellbogen, Hand, Hüfte, Knie, Füsse)

User Position: Tiefengenerator, optimiert für Distanz des Users

Error State: Meldet Fehlfunktion oder Ausfall eines Nodes

5.5.6. Daten generieren und lesen

Zuerst werden die entsprechenden Generatoren erzeugt und dem Context hinzugefügt. Diese produzieren jedoch noch keine Daten. Erst können sie noch konfiguriert werden. Anschliessend werden sie mit *StartGenerating()* gestartet und mit *StopGenerating()* wieder gestoppt. Ist ein Node aktiviert, erzeugt er kontinuierlich Daten, gibt diese jedoch erst nach einem Aktualisierungsbefehl frei (z.B. *WaitAndUpdateAll()*) In dieser Zeit wird effektiv gewartet bis neue Daten verfügbar sind. Nach 2 Sekunden wird ein Timeout ausgelöst. Die Tiefeninformationen können durch das entsprechende Meta Objekt als Map ausgelesen werden. Dazu muss lediglich das gewünschte Pixel durch die X- und Y-Koordinate angegeben werden.

5.5.7. Gesten erkennen

Nachdem der Context und der *Hand Generator*, sowie der *Gesture Generator* erzeugt wurden, müssen noch die Gesten hinzugefügt werden. Danach gibt es zwei Arten von Gesten, Fokus Gesten und weitere (Abschnitt 5.3.8). Wird eine sogenannte Fokus Geste ausgeführt, wird eine Hand als bekannt angemeldet und verfolgt. Danach können die weiteren Gesten verwendet werden. Wird diese Hand verloren, wird sie entsprechend wieder gelöscht.

Bei jeder Geste wird in C++ eine selbst definierbare CallBack Funktion aufgerufen, bei welcher folgende Informationen übergeben werden: Welche Hand, welche Geste und Position der Hand. Darin kann dann Individuell die Reaktion auf Gesten beschrieben werden. Es ist auch möglich, eigene Gesten zu implementieren. In C# ist dies mit Events gelöst.

5.5.8. Skeletterkennung

Skeletterkennung ist bereits implementiert. Wird ein User erkannt, wird dieser registriert. Danach ist eine Pose von Nötzen. Wird diese erfolgreich erkannt, wird der User kalibriert und dessen joints ausgelesen. Ist der User eine kurze Zeit nicht mehr vor der Kamera, wird er wieder gelöscht.

6. RooKi

6.1. Systembeschreibung

Das System RooKi besteht aus dem Putzroboter Roomba als Plattform, einem Notebook, welches die Betriebsssoftware ausführt und einer MS Kinect. Das Notebook ist zusammen mit der Kinect auf einer Holzplatte montiert, welche wiederum auf den Roomba geschraubt wurde. An diesem Holzbrett sind noch Akku und Speisungsboard montiert (Abb. 6.3a).

Kinect bildet den Input zum System. Natural Interaction stellt so den Input dar. Das heisst, das System reagiert auf Gesten und Bewegungen, welche mit der Kamera aufgenommen und in der Betriebsssoftware verarbeitet werden. Zusätzlich wird in gewissen Betriebsmodi die Umgebung als weiterer Input hinzugezogen. Die Verarbeitung der Bilder bezüglich Gesten geschieht hauptsächlich durch OpenNI und bildet somit die Schnittstelle zu Kinect.

Aus diesen Inputs werden Fahrbefehle für die Plattform Roomba errechnet, welche über eine serielle Schnittstelle übertragen werden. Diese nutzt das offizielle Roomba Open Interface. Ein USB Stecker emuliert einen seriellen RS-232 Port. Dieses Signal wird anschliessend von einem Level Shifter in TTL umgewandelt. Diese Verbindung funktioniert bidirektional.

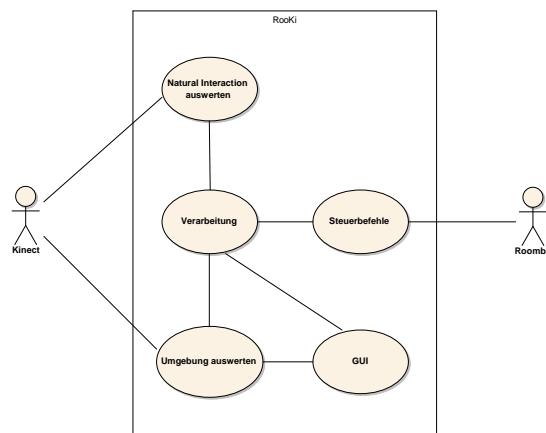
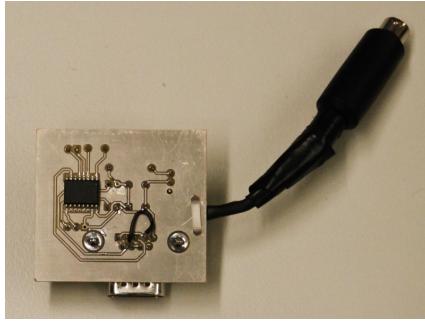
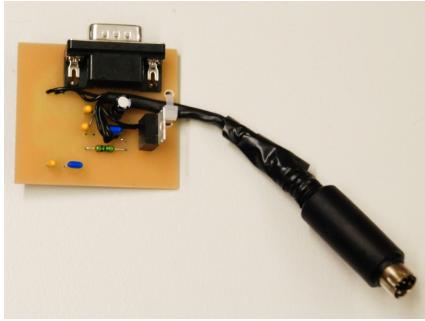


Abbildung 6.1: Use Case Diagram RooKi

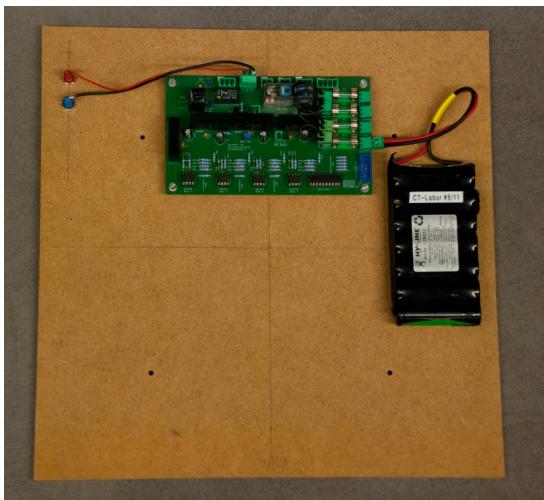
Roomba und Notebook werden von ihren eigenen Akkus gespeist. Roomba bietet über seine serielle Schnittstelle einen direkten Anschluss an seinen Akku. Die Leistung reicht jedoch nicht aus, um Kinect damit zu versorgen. Daher wurde Kinect über ein Stromversorgungsboard versorgt, welches für Eurobot entwickelt wurde. Dieses verfügt über einen 30V Akku und stellt verschiedene geregelte Spannungen zur Verfügung. Der 12V Ausgang bietet einen Ausgangstrom von 2A und ist daher mit Sicherheit ausreichend.



(a) Levelshifter von oben



(b) Levelshifter von unten

Abbildung 6.2: Levelshifter RS-232-TTL

(a) Bodenplatte des RooKi mit Speisungsboard und Akku



(b) RooKi

Abbildung 6.3: Aufbau des RooKi

6.2. Betriebsverhalten

Das Verhalten von RooKi ist in verschiedene Betriebsmodi unterteilt, welche mit Natural Interaction betreten und verlassen werden können. Des Weiteren können alle Statuswechsel mit Buttons auf dem GUI vorgenommen werden. Im Einschaltzustand befindet sich RooKi im Ruhezustand, das heisst er steht still und wartet auf Befehle.

RooKi verfügt grundsätzlich über drei verschiedene Funktionalitäten. Er kann selbstständig durch den Raum navigieren, indem er die Tiefenbilder von Kinect auswertet (siehe Abschnitt 6.3.2). Ein User kann als Ziel identifiziert werden, welchem dann RooKi folgt (siehe Abschnitt 6.3.3). Zusätzlich können durch Gesten einzelne Fahrbefehle erteilt werden (siehe Abschnitt 6.3.4). Die

Struktur der Software wurde auf einer State Machine aufgebaut.

Als Feedback für den User wurden State Notifications implementiert. In vielen Fällen sind diese nicht nötig, da RooKi nach einer Geste oft seine Tätigkeit unverzüglich aufnimmt und dadurch ersichtlich ist, dass RooKi den Befehl richtig interpretiert hat. Es gibt aber Fälle, bei denen sich lediglich der aktuelle State ändert. Für diesen Fall quittiert RooKi den State Wechsel mit einem akustischen Signal. Details sind Abb. 6.3 zu entnehmen.

6.3. State Machine

Das System wurde mit einem State Pattern implementiert. In diesem bestehen alle States aus Singletons, welche von einer abstrakten Basisklasse abgeleitet sind. Diese Basisklasse ist einer Contextklasse bekannt, welche die Schnittstelle zur State Machine bildet. Demnach wird zur Initialisierung der State Machine eine Instanz dieser Context Klasse instanziert (hier RooKiCtrl). Die RooKiCtrl enthält zudem eine Referenz auf die Instanz des aktuellen Zustandes.

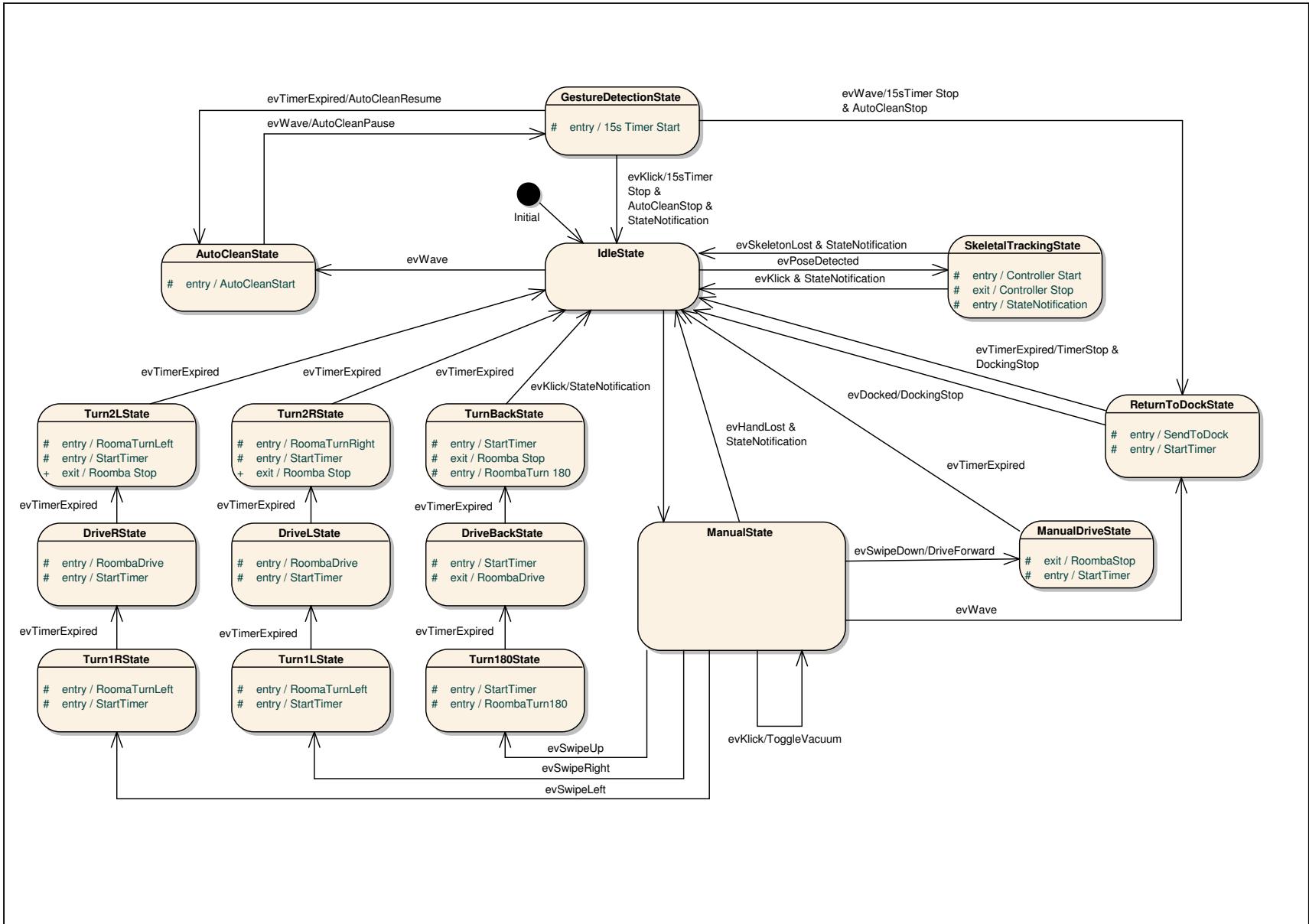
RooKiCtrl: Der Konstruktor initialisiert alle nötigen Komponenten und setzt den Initial Zustand. Als weitere public Methode bietet sie die Funktion *process* an. Diese reicht den Event an den aktuellen Zustand weiter. Dieser definiert den möglichen Folgezustand und die transition action. In der statischen Funktion *Change State* der Basisklasse werden nun die entsprechenden entry-, transition- und exit actions ausgeführt. Anschliessend wird die Referenz des aktuellen Zustandes RooKiCtrl übergeben und die State Machine verlassen.

RooKiState: Die Basisklasse stellt die Definition für alle abgeleiteten State Klassen dar und enthält einige Deklarationen, siehe Listing 6.1. Ein delegate namens *TransitionFunc* ermöglicht den Aufruf der transition Funktionen aus den States heraus, welche ebenfalls in RooKiState definiert sind.

Singleton: Ein einzelner State ist immer gleich aufgebaut. Beim ersten Zugriff auf das Property Instance wird die Instanz in Zeile 1 (Listing 6.1) erstellt. Bei einem Event wird nun die *process* Funktion der Klasse aufgerufen, wobei entschieden wird, ob das Event im aktuellen Status von Interesse ist. Wenn ja wird die *Change State* Funktion der Basisklasse aufgerufen, welche die entsprechenden Actions ausführt. Entry und exit Actions sind in der jeweiligen Klasse definiert. Der Konstruktor wird als private deklariert, damit keine weiteren Instanzen der Klasse erzeugt werden können.

Dadurch lässt sich die State Machine ideal abkapseln. Alle Aktionen, welche an einen Status gebunden sind, werden auch in dieser Klasse definiert. Von aussen ist die State Machine lediglich durch die *process* Methode zugänglich.

Die Events bestehen hauptsächlich aus Daten von der Kinect oder dem Auslaufen eines Timers.



```
1 static readonly AutoCleanState instance = new AutoCleanState();
2 BeispielState()
3 {
4     stateName = "BeispielState";
5 }
6 public static BeispielState Instance
7 {
8     get{return instance;}
9 }
10 public override RooKiState process(RooKiCtrl.Event e)
11 {
12     if ((RooKiCtrl.Event.evKlick == e) || (RooKiCtrl.Event.evWave == e))
13     {
14         return ChangeState(TransitionAction, BeispielFolgeState.Instance);
15     }
16     return null;
17 }
18 protected override void entryAction()
19 {
20 }
21 protected override void exitAction()
22 {
23 }
```

Listing 6.1: Singleton

6.3.1. Idle State

Dies ist der Standby und Initial Status des RooKi, in welchem keine Aktionen vorgenommen werden, bis der User eine Eingabe tatigt.

6.3.2. Auto Clean und Gesture Detection State

Dieser Status wird mit einer Geste Wave betreten. RooKi fahrt nun selbststandig durch den Raum. Objekte oder Wnde werden fruhzeitig detektiert und entschieden, ob Roomba diese umfahren kann. Wird die Geste Wave ausgefuhrt, halt der Roomba fur 15 Sekunden an und wartet im *Gesture Detection State* auf neue Befehle.

In dieser Zeit kann man per Click Geste zuruck in den *Idle State* wechseln oder mit einer Wave Geste den RooKi zu seiner Docking Station schicken. Lauft die Zeit jedoch ohne Ausfuhrung einer dieser beiden Gesten ab, fahrt er im *Auto Clean State* fort.

Funktionalitat

Jeder Pfeil symbolisiert einen Algorithmusablauf. Gibt es ein Objekt, das umfahren werden kann, wahlt RooKi den kurzesten Weg. In der Abb. 6.4 ist die rechte Kante des Objektes naher als die linke Kante. Nun fahrt RooKi bis zur Wand, dreht 90° nach rechts, fahrt bis zur Kante, dreht sich 90° nach links und fahrt geradeaus weiter.

Wird eine Wand detektiert dreht er sich nach rechts ab. Vorgesehen war eine 90° Drehung (Abb. 6.4a). Jedoch war dies nicht praxistauglich. Fahrt RooKi nicht genau senkrecht auf die Wand zu, kann es sein, dass er nach der Drehung von 90° nicht exakt parallel zur Wand steht. Da Kinect weder einen 180° Blickwinkel noch etwas unter 500mm erkennen kann, wurde RooKi in die Wand fahren. Aus diesem Grund wurde der Winkel >90° gewahlt (Abb. 6.4b). Als gute Erweiterung konnte man hier den Winkel zur Wand, auf die RooKi zufahrt bestimmen und dementsprechend den Drehwinkel anpassen.

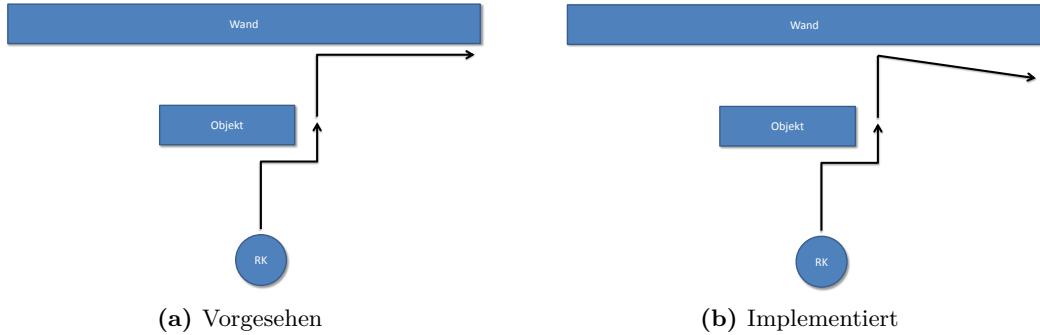


Abbildung 6.4: Auto Clean Verhalten (rechts)

In der Abb. 6.5 befindet sich RooKi näher an der linken Kante. Somit umfährt er das Objekt links. Da er sich an einer Wand immer gleich verhält, dreht er sich auch hier nach rechts.

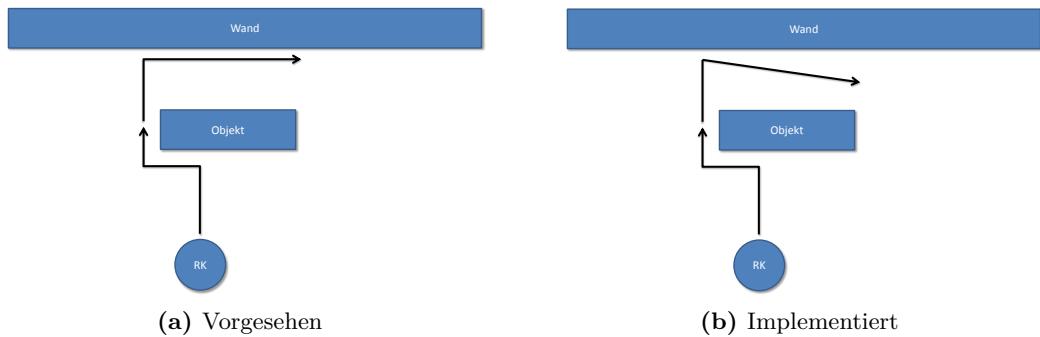
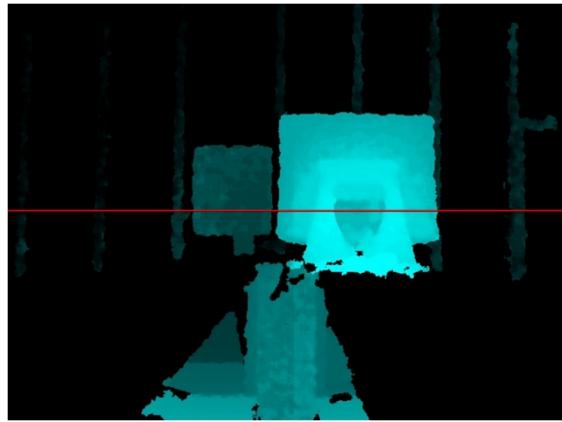


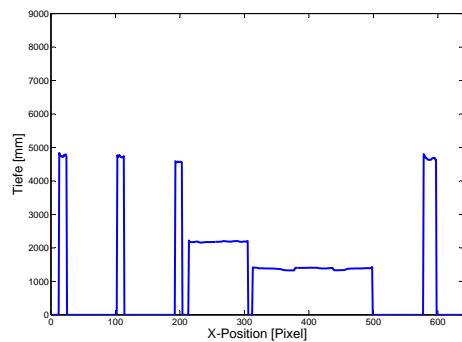
Abbildung 6.5: Auto Clean Verhalten (links)

Algorithmus

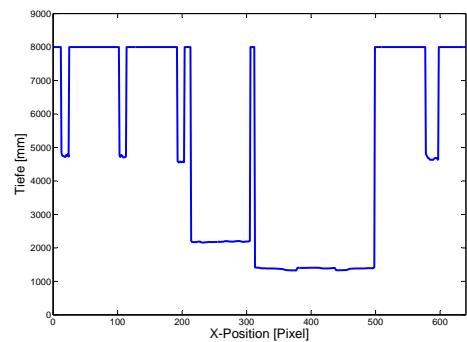
Es wird eine Zeile des Tiefenbildes (Momentan Zeile 239 - Mitte: Abb. 6.6a) über 10 Frames gemittelt (Abb. 6.6b). Dabei werden Werte die 0 sind nicht mit einberechnet. Danach werden alle Werte normalisiert, das heisst die Werte, die 0 oder grösser als der definierte Maximalwert (hier 8000) sind auf den Maximalwert gesetzt.



(a) Tiefenbild mit markierter Zeile



(b) Tiefenwerte über 10 Frames gemittelt



(c) Tiefenwerte normalisiert

Abbildung 6.6: Zeilenverarbeitung des Auto Clean Algorithmus

Nun kann die Zeile ausgewertet werden. Es ist wichtig, dass 8 Pixel breite Nullband zu beachten.

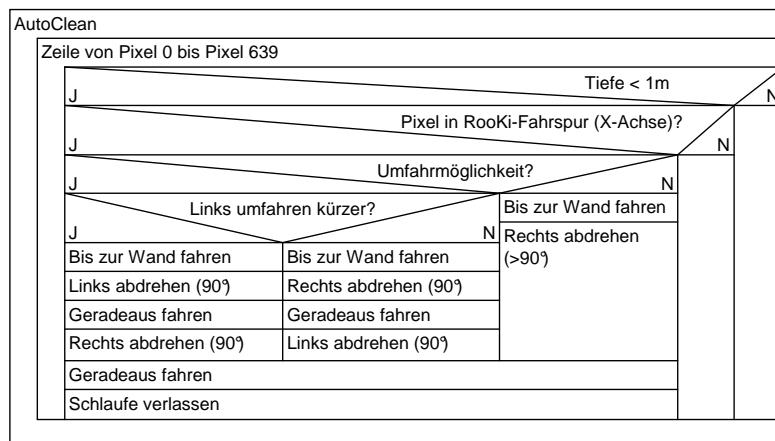


Abbildung 6.7: Auto Clean Struktogramm

Über- und Unterfahren

Das Unterfahren könnte ganz einfach gelöst werden, indem man eine weitere Zeile mit dem bestehenden Algorithmus auswertet. Das Problem besteht darin welche Zeile auszuwerten ist. Je nach Neigungswinkel der Kinect müsste eine andere Zeile ausgewertet werden. Dies könnte am besten mit einer zukünftigen OpenNI Version mit Motor-Unterstützung erzielt werden oder behelfsmässig mit anderen Treibern aus dem Internet, was jedoch eine nicht ganz saubere Lösung wäre. Auch einfach eine Kameraeinstellung festzulegen wäre nicht perfekt.

Die Funktion Überfahren ist nicht nötig, da der Roomba aufgrund seiner Bauweise nur einen sehr kleinen Steigungswinkelanstieg meistern kann. Dieser könnte kaum detektiert werden.

Erweiterungsmöglichkeiten

- Winkel bestimmen mit dem RooKi auf die Wand zufährt um den Drehwinkel anzupassen.
- Überprüfen ob der Platz neben dem zu umfahrenden Objekt breit genug für RooKi ist.
- Unterfahren: Weitere Zeile auslesen und mit gleichem Algorithmus auswerten.

6.3.3. Skeletal Tracking State

Mit der Pose (Abb. 6.8) wird die aktuelle Distanz ermittelt und als d_{soll} gespeichert. RooKi bewegt sich anschliessend so, damit RooKi immer die gespeicherte Distanz zum User beibehält und sich auch entsprechend in seine Richtung dreht. Dieser Modus kann mit einer Click Geste verlassen werden.

Die zu fahrende Strecke wird als Kreisbogen zur Sollposition berechnet (Abb. 6.9). Die folgenden grün markierten Angaben sind gegeben:

- d_{kinect} : Distanzwert zum Torsopunkt des Users
- d_{soll} : Die zuvor gespeicherte Solldistanz
- Δx : Abweichung des Users in X-Richtung in Pixel

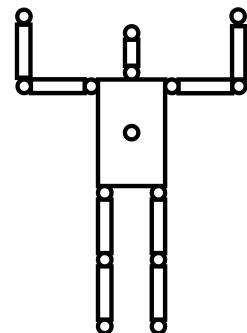


Abbildung 6.8: Detection Pose

Zuerst wird α und d_{ist} berechnet. Der horizontale Winkel der Kamera beträgt 57° . Mit einer Auflösung von 640 Pixeln in X-Richtung wird α als Verhältnis der Abweichung in Pixel zur Gesamtauflösung berechnet. Dies ist nur eine Annäherung, da die horizontale Abweichung vom Mittelpunkt nur in Pixel gegeben ist, reicht jedoch für diesen Fall problemlos. Mit diesem lässt sich anschliessend die direkte Distanz zum User berechnen.

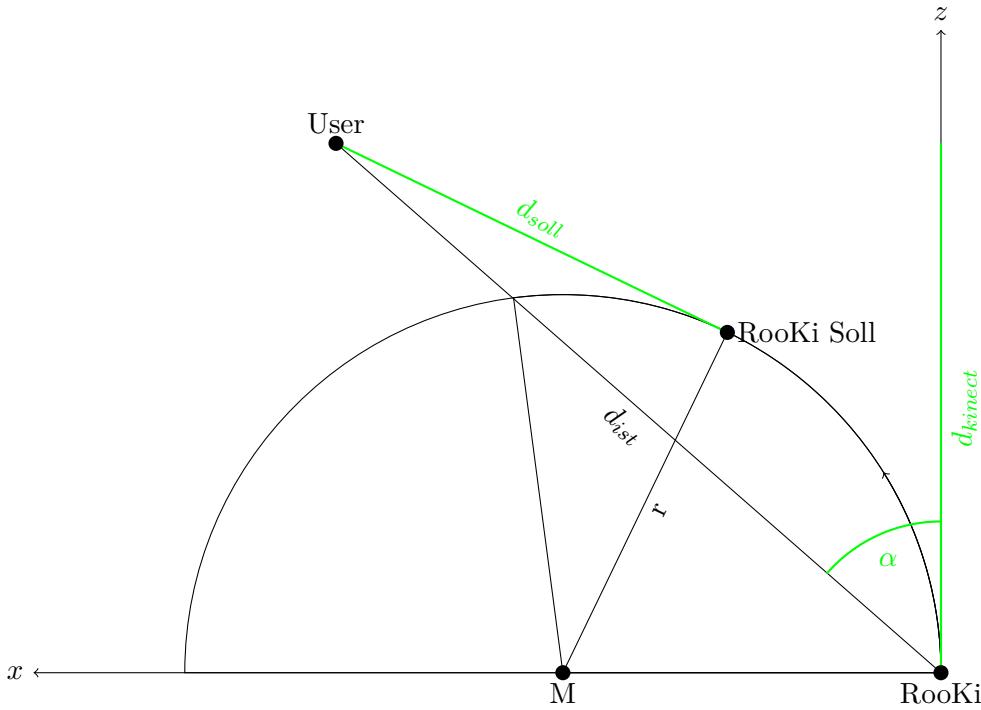


Abbildung 6.9: Kurvenradius

$$\alpha = \frac{\Delta x * 57}{640} \quad (6.1)$$

$$d_{ist} = \frac{d_{kinect}}{\cos(\alpha)} \quad (6.2)$$

Für den Kreisbogen muss erst die Sekantenlänge, welche Teil von d_{ist} ist, anhand des Sekanten-Tangenten Satzes berechnet werden. Dieser lautet in diesem Fall

$$d_{soll}^2 = (d_{ist} - d_{Sekante}) * d_{ist} \quad (6.3)$$

Mit den beiden Radien und der Sekante als Grundseite entsteht nun ein gleichschenkliges Dreieck, dessen Seitenlänge r anhand des Sinussatzes (Gl. 6.4) berechnet werden kann, da nun Sekantenlänge und die Basiswinkel ($90 - \alpha$) bekannt sind (γ ist der Winkel gegenüber von $d_{Sekante}$).

$$\frac{d_{Sekante}}{\gamma} = \frac{r}{90 - \alpha} \quad (6.4)$$

Löst man dies nun nach r auf und ersetzt den Winkel γ mit der Winkelsumme des Dreiecks minus zweimal die Basiswinkel erhält man die Gleichung für r .

$$\frac{d_{Sekante}}{180 - 2(90 - \alpha)}(90 - \alpha) = r \quad (6.5)$$

Löst man nun (Gl. 6.3) nach $d_{Sekante}$ auf und setzt man sie in (Gl. 6.5) ein, so erhält man eine Formel für r , welche nur von gegebenen Werten abhängt.

$$\frac{d_{ist} - \frac{d_{soll}^2}{d_{ist}}}{2 * \alpha}(90 - \alpha) = r \quad (6.6)$$

Zur Entscheidung der Fahrbefehle wird grundsätzlich unterschieden, ob die Distanz innerhalb einer Toleranz von 20cm liegt. Ist dies nicht der Fall, dreht sich RooKi auf der Stelle, damit der User immer in der Mitte des Tiefenbildes verbleibt. Ist der Distanzfehler hoch, wird entschieden, ob die hergeleitete Formel zur Anwendung kommt. Falls nicht, fährt RooKi gerade, da ansonsten der Radius des zu fahrenden Kreisbogens zu gross werden würde. Dieses Verhalten ist in Abb. 6.10 im Detail aufgezeigt.

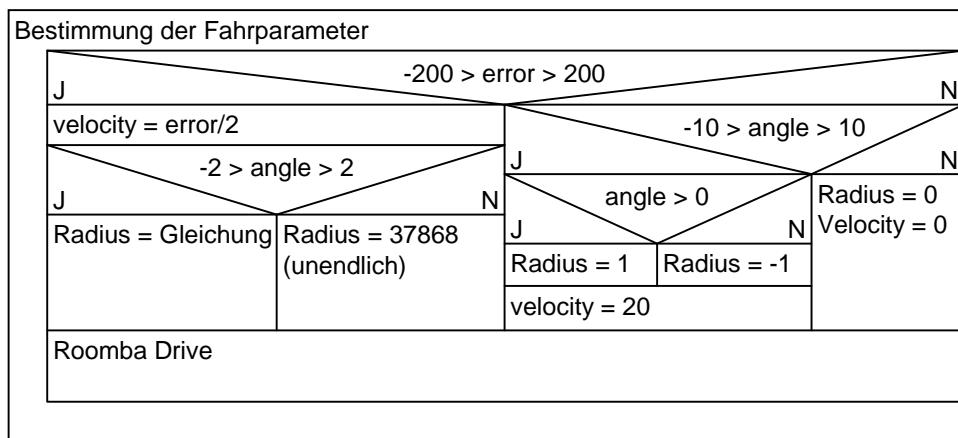


Abbildung 6.10: Entscheidungsprozedur des Fahrbefehls

6.3.4. Manual State

RooKi kann angewiesen werden, einzelne Aktionen auszuführen. Dazu muss als erstes eine Click Geste ausgeführt werden. Anschliessend kann der entsprechende Befehl erteilt werden. Ist dieser ausgeführt, kehrt RooKi in den *Idle State* zurück.

Swipe Left/Right: Handbewegung nach links bzw. nach rechts. Für diesen Befehl werden drei Zustände durchlaufen. In die entsprechende Richtung drehen, einen Meter vorwärts fahren und

sich wieder zum User zurückdrehen. Beim Betreten jedes Zustandes wird ein Timer geladen. Dessen Ablaufen löst ein Event aus, welcher die State Machine dazu veranlasst, den nächsten Zustand zu betreten.

Swipe Up/Down: Analog zu Left/Right. Bei Down fährt RooKi einen Meter vorwärts. Roomba erhält den Befehl mit einer Geschwindigkeit von 400mm/s zu fahren. Gemessen wurde aber eine Geschwindigkeit von 365 mm/s. Somit wird als Fahrzeit gerundet 2800 ms verwendet. Nachdem RooKi den Fahrbefehl erhalten hat, wird ein Timer initialisiert, dessen Ablaufen RooKi wieder stoppt und in den *Idle State* versetzt. Wird die Hand nach oben bewegt, dreht sich RooKi um 180°, fährt ebenfalls einen Meter und dreht sich zum User zurück. Dieses Verhalten wurde so gewählt, damit sich während des Fahrens RooKis IR-Sensoren an der Vorderseite befinden. Diese können Gefälle wie eine Treppe detektieren und verhindern ein ungewolltes Herunterfallen. Da sich der Roomba im Safe Mode befindet, stoppt er in diesem Fall unverzüglich und unabhängig von den Fahrbefehlen.

Wave: RooKi wird angewiesen, auf seine Docking Station zu fahren. Mehr dazu unter Abschnitt 6.3.5.

Klick: Aktiviert seine Putzroutine, bei welcher er anfängt zu saugen und seine Bürsten einschaltet. Sind diese bereits eingeschalten, werden sie wieder deaktiviert. Dies wird anhand einer Spiegelvariable in der Klasse Roomba realisiert, da sich der aktuelle Zustand nicht zuverlässig anhand der seriellen Schnittstelle auslesen lässt.

Für die Parameter Geschwindigkeiten und Zeiten wurde ein XML File namens RooKi.exe.config angelegt. Damit können diese Werte auch nach dem kompilieren noch angepasst werden, indem das entsprechende File verändert wird. Folgende Werte können angepasst werden:

- TurnTime: Dauer einer 90° Drehung
- DriveTime: Dauer der Fahrt eines einzelnen States mit Drehung (links, rechts und rückwärts)
- ManualDriveTime: Dauer während des Vorwärtsfahren
- DockTime: Dauer während RooKi seine Docking Station sucht, bis er ohne Erfolg in den *Idle State* wechselt
- ManualDriveSpeed: Geschwindigkeit in allen manuellen Fahrbefehlen

6.3.5. Return to Dock State

Aus dem *Manual State* oder *Gesture Detection State* ist es möglich, mit einer Wave Geste den *Return to Dock State* zu erreichen. In diesem wird Roombas eigener Algorithmus verwendet, um zu seiner Docking Station zurückzukehren. Während dessen wird zyklisch ausgelesen, ob Roomba seine Docking Station bereits erreicht hat. Wenn dies der Fall ist, oder ein Timer von einer Minute ausläuft, kehrt RooKi automatisch in seinen *Idle State* zurück.

Dieser State ist noch erweiterbar. Da das Holzbrett des RooKi über die Aussenkante von Roomba reicht, verdeckt dies die Bumper. Diese werden für Roombas Suche nach der Docking Station benötigt, da er blind fährt. Dieser State sollte in diesem Stand der Software nur dann betreten werden, wenn sich RooKi in der Nähe seiner Docking Station befindet, da RooKi sonst bei der Kollision mit dem ersten Hindernis nicht mehr weiter kommt.

6.4. Recorder / NiViewer

OpenNI bietet einen Recorder an, mit welchem die Daten aller Generatoren gespeichert werden können. Dies bietet die Möglichkeit, ein Szenario aufzunehmen. Dieses File wird im selben Ordner abgelegt, in welchem RooKis Programm gestartet wird und trägt den Namen rookirecord.oni. Dies kann anschliessend untersucht werden oder durch den Player von OpenNI als Ersatz für Kameradaten verwendet werden. Für die nachträgliche Analyse der Daten bietet OpenNI bereits ein Open Source Tool namens NiViewer an.

NiViewer bietet verschiedene Möglichkeiten an, Tiefenbilder von Kinect zu analysieren. Diese Werte können entweder direkt von der Kamera oder von einer vorgängigen Aufnahme stammen. Die Anwendung wechselt automatisch beim Start in den Vollbildmodus und ist durch Tasten bedienbar.

- 1-9: Voreingestellte Ansichten
- m: Spiegelt die aktuelle Ansicht
- s / x: Startet und stoppt Capture
- c: Einzelnes Frame in .raw file speichern (für jedes Frame ein eigenes file, dessen Namen mit automatischem Index versehen sind.)
- p: Pointermode: Eine Leiste erscheint mit Angabe über Frame Nummer, Timestamp, Skala der Tiefenwerte und Tiefenwert beim Mauszeiger.
- f: Toggle Fullscreen
- o: Pause/Resume

- 1 / L: 1/10 Frames vorwärts
- k / K: 1/10 Frames rückwärts
- ';' : Ein einzelnes Frame von Kamera lesen und bei diesem stoppen
- ']' / '[' : Playback Geschwindigkeit erhöhen/verringern
- '?': Hilfe

Die aus NiViewer erzeugten raw Files eines Frames können anschliessend mit Matlab mit Hilfe von folgendem Code extrahiert und analysiert werden. Wird NiViewer im ursprünglichen Zustand aus dem Ordner von OpenNI gestartet, muss darauf geachtet werden, dass das Programm als Administrator ausgeführt wird, da sonst keine Frames gespeichert werden können. 640 und 480 entspricht der Auflösung des Bildes und sollten angepasst werden, falls das Bild mit einer anderen Auflösung aufgenommen wurde. x ist in diesem Fall ein 640x480 Array, welches alle Tiefenwerte enthält.

```
1 % Open file
2 id = fopen('Depth_1.raw', 'r');
3 % Read in the data.
4 x = fread(id, [640,480], 'short');
5 % Close file
6 fclose(id);
```

Listing 6.2: Lesen von .raw Files

6.5. Klassen

Beim Programmstart instanziert das GUI ein RooKiCtrl-Objekt. Dies wiederum instanziert die Finite State Machine (FSM). Zur FSM gehören alle State-Klassen.

6.5.1. NuiSensor

Die NuiSensor Klasse wird von RooKiCtrl instanziert. Sie stellt das Verbindungsglied zwischen der Applikation und der Kinect dar. Hier wird das Tiefenbild, sowie das Property „TorsoDistance“ berechnet und bereitgestellt. Zusätzlich sind hier auch alle Properties für die Gesten hier zu finden. Ebenfalls werden alle Gesten initialisiert, sowie deren Parameter bestimmt.

6.5.2. DriveCtrl

Alle Fahralgorithmen des RooKi sind in der DriveCtrl-Klasse zu finden.

Algorithmen

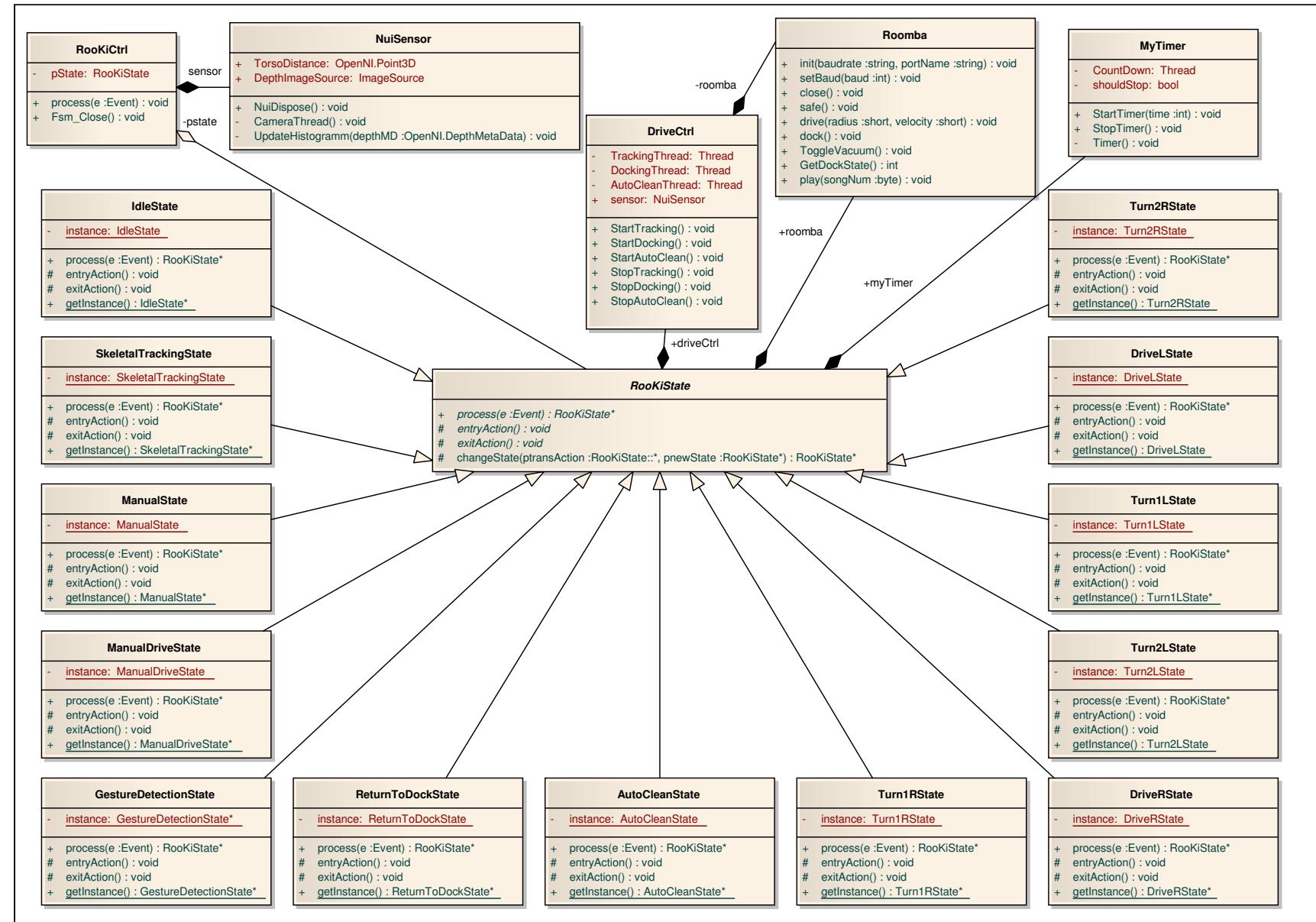
- Skeletal Tracking
- Auto Clean
- Docking

6.5.3. Roomba

Diese Klasse stellt die Schnittstelle zwischen der Applikation und dem Roomba dar. In ihr werden die RS-232 Befehle für den Roomba in einfachen Funktionen bereitgestellt.

6.5.4. MyTimer

MyTimer stellt einen einfachen Countdown dar. Man kann ihn mit einem Wert in Millisekunden starten. Nach Ablauf der Zeit wird ein Event ausgelöst.



6.6. GUI

Startet man das Programm RooKi, öffnet sich das GUI (Abb. 6.11). Das GUI hat eine feste Auflösung von 982x580 Pixel. Zu Beginn sind die Event Buttons inaktiv und es wird im linken Bereich kein Tiefenbild angezeigt.

Unter dem Punkt *Recording* lässt sich der Aufnahmemodus ein- oder ausschalten. Während des Betriebes kann diese Einstellung nicht mehr verändert werden.

In der ComboBox *COM Port* werden alle, momentan zur Verfügung stehenden, COM Ports aufgelistet. Hier ist derjenige Port zu wählen, an dem der Roomba angeschlossen ist.

Daneben muss man die Baudrate wählen, die am Roomba eingestellt wurde. Normalerweise ist dies *115200* kann aber auf *19200* umgestellt werden.

Sind alle Einstellungen getätigter, kann man die Session mit dem *Start* Button beginnen.

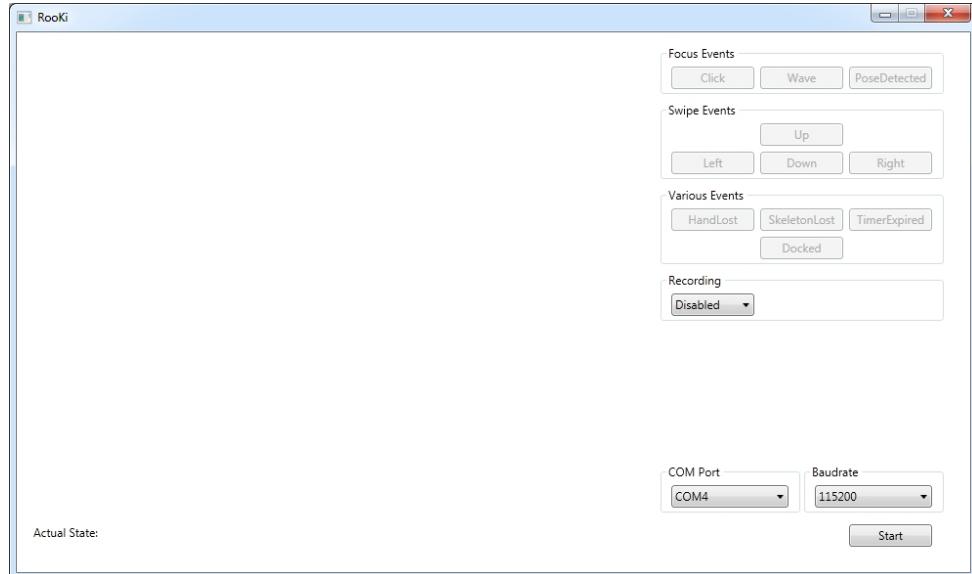


Abbildung 6.11: GUI nach Öffnen

Wurde die Session gestartet und waren alle Einstellungen korrekt, erscheint im linken Bereich des GUI das aktuelle Tiefenbild, welches die Kinect aufnimmt. Darunter wird im Bereich *Actual State:* der momentane State in der sich die State Machine befindet angezeigt.

Nun sind die Event Buttons aktiv und können benutzt werden um Statewechsel zu bewirken ohne Gesten auszuführen. Die Bereiche *Recording*, *COM Port* und *Baudrate*, sowie der *Start* Button sind nun inaktiv. Um das Programm zu beenden ist das rote Kreuz oben rechts zu benützen.

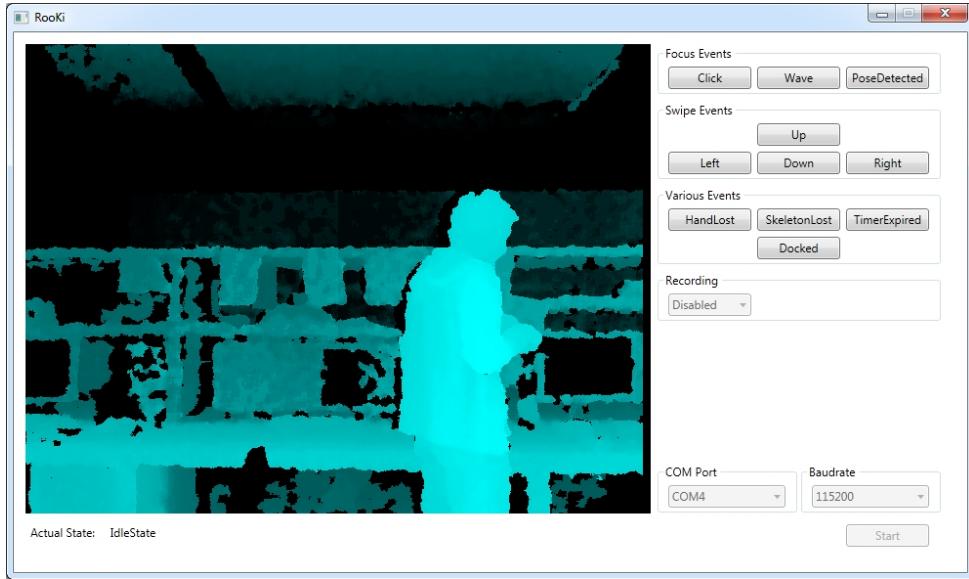


Abbildung 6.12: GUI nach Start

Event Buttons

Click	Simuliert eine Click-Geste (Hand vor und zurück)
Wave	Simuliert eine Wave-Geste (4 mal Winkbewegung vollführen)
PoseDetected	Simuliert dass ein User von openNI erkannt wurde (User macht DetectionPose)
Up	Simuliert eine Swipe-Up-Geste (Hand nach oben bewegen)
Left	Simuliert eine Swipe-Left-Geste (Hand nach links bewegen)
Down	Simuliert eine Swipe-Down-Geste (Hand nach unten bewegen)
Right	Simuliert eine Swipe-Right-Geste (Hand nach rechts bewegen)
HandLost	Simuliert das Verlieren einer Hand durch OpenNI
SkeletonLost	Simuliert das Verlieren eines Skelettes (User) durch OpenNI
TimerExpired	Simuliert den Timer-Abgelaufen-Event
Docked	Simuliert das Erreichen der Docking Station des Roombas

Tabelle 6.1: Event Buttons

6.7. Systemvalidierung

6.7.1. FSM

In der State Machine wurde in allen States die Reaktion auf alle möglichen Events gemäss State Chart geprüft. Dabei wurde geprüft, ob alle Statusübergänge korrekt vorgenommen werden oder der Event ignoriert wird, falls dieser in diesem State nicht relevant ist. Zusätzlich wurde

anhand einer Konsolenausgabe sichergestellt, ob alle Entry-, Exit- und Transition Actions richtig ausgeführt werden. Diese Konsolenausgabe wurde in der Releaseversion entfernt.

Diese Tests sind alle erfolgreich bestanden worden.

6.7.2. Roomba

Alle verfügbaren Befehle der Roomba Klasse wurden auf ihre Funktionalität überprüft. Die meisten Übergabeparameter werden vom Programmierer beschrieben, weshalb ein ungültiger Wert nicht geprüft oder abgefangen werden muss. Zu hohe Geschwindigkeiten oder Drehradien für Fahrbefehle werden schon in den einzelnen Softwaremodulen abgefangen. Sollte Roomba doch einen höheren Wert empfangen, wird dieser durch Roomba automatisch auf sein spezifiziertes Maximum gesetzt.

Der Befehl *play()* zeigt manchmal ein Fehlverhalten. Dieser sollte eine zuvor definierte Abfolge von Tönen wiedergeben. Diese Songs werden nicht immer wiedergegeben. Dies tritt besonders oft auf, wenn mehrere Songs in kurzer Zeit abgespielt werden müssen. Der Fehler konnte nicht nachvollzogen werden, da die zugehörigen Befehle korrekt implementiert wurden und weder kurz zuvor, noch danach andere Befehle gesendet werden.

Alle restlichen Tests sind alle erfolgreich bestanden worden.

6.7.3. Nui Sensor

Die Events der NuiSensor Klasse wurden alle mit Konsolenausgaben überprüft. Das Tiefenbild, sowie das Property *TorsoDistance*, welches für den *Skeletal Tracking State* benötigt wird, wurden live getestet und zeigten keine Anomalien.

Bei der Swipe Gesten mussten noch einige Parameter angepasst werden. Dies geschah nach dem Trial and Error Verfahren, da keine Dokumentation zu den Parametern besteht. Es wurden akzeptable Einstellungen gefunden. Je nach Anwendung müssen diese angepasst werden.

6.7.4. Auto Clean State

Das Auslesen der Zeilen, das Mitteln und das Normalisieren wurden mithilfe von MATLAB überprüft. Die verschiedenen Verzweigungsmöglichkeiten wurden durch gestellte Szenarien überprüft und haben das erwartete Verhalten gezeigt.

Es gibt noch Möglichkeiten, die RooKi nicht meistern kann. Zum Beispiel wenn zwei Objekte nahe beinander sind, aber ein Spalt zwischen ihnen vorhanden ist, würde er durch den Spalt

zu fahren versuchen, da nicht überprüft wird ob die Breite der Umfahrstrecke genug breit für RooKi ist.

6.7.5. Manual State

Die in den Settings definierten Zeiten und Geschwindigkeiten wurden überprüft. Folgende Messungen wurden vorgenommen.

- Geschwindigkeit: In allen States wird eine Geschwindigkeit von 400mm/s gewählt. Gemesen wurde aber 364mm/s. Diese Abweichung ist von Rooma gegeben und könnte an der zusätzlichen Last der Holzplatte und des Notebooks liegen.
- Drehradius: In allen Drehstatus wird eine Drehung von 90° angestrebt. Die Drehzeiten wurden empirisch festgelegt wobei sich herausstellte, dass der Drehwinkel nicht exakt eingestellt werden kann. Bei jeder Drehung entsteht ein Fehler von rund 10°. Die Tests bezüglich der Drehrichtungen konnten jedoch erfolgreich durchgeführt werden und sind bestanden.
- Toggle Vacuum: Es konnte jederzeit korrekt getoggled werden.
- Return to Dock: Es wurde getestet, ob Roomba die Suche nach seiner Docking Station aufnimmt, in den *Idle State* zurückkehrt, falls Roomba sich angedockt hat und ob RooKi seine Suche nach einer Minute wieder beendet, falls die Station nicht gefunden wurde. Alle Verhaltensmuster werden korrekt durchgeführt.

Es ergeben sich also Abweichungen durch Roomba. Das System ist jedoch korrekt implementiert.

6.7.6. Skeletal Tracking State

Die Funktion hat einige durch OpenNI und Roomba gesetzte Limitationen, welche die Bewegungsfreiheit des Users einschränken. Zum einen darf sich der User nicht schneller als 0.5m/s von RooKi fortbewegen, da dies die maximale Fahrgeschwindigkeit des Roboters ist. Eine Messreihe mit fünf Messungen hat gezeigt, dass OpenNI den User bei einer Distanz von mehr als 4.5m nicht mehr sehen kann und RooKi somit in den *Idle State* zurückfällt. Wenn der User sich also mit einer Geschwindigkeit von RooKi fortbewegt, welche höher ist als RooKis Fahrgeschwindigkeit, muss darauf geachtet werden, dass man innerhalb dieser Distanz bleibt.

Auch die Bewegungen in X-Richtung sind limitiert. So sollte sich der User auch in X-Richtung nicht zu schnell bewegen. Auch wenn der User innerhalb der 57° Sichtwinkel bleibt und nahe genug bei RooKi steht, besteht dennoch die Möglichkeit, dass RooKi den User verliert. Hier wird ersichtlich, dass Kinect und OpenNI dafür ausgelegt sind, dass die Kamera still steht. Wenn

sich RooKi dreht und sich der User gleichzeitig bewegt, erhöht sich die Wahrscheinlichkeit des Verlusts des Users stark. Werden die Schritte jedoch in langsamem Schritttempo ausgeführt, kann RooKi diesen folgen.

Folgende Fälle konnten aber in verschiedenen Umgebungen erfolgreich durchgeführt werden. Vorgabe für alle Tests war, dass sich RooKi an seiner Sollposition befindet und keine Hindernisse seine Fahrt behindern.

- Einzelner grosser Schritt diagonal zu RooKi. Hier fuhr RooKi einen Kreisbogen, ohne am Anfang oder Ende seine Richtung durch eine Drehung zu ändern.
- Diagonal zu RooKi weglauen. Hier wird ein immer grösser werdender Kreisbogen gefahren, bis sich RooKi in einer Linie zum User befindet und nur noch die Distanz nachregelt.
- Bewegt man sich aus dem Bild, oder wie einleitend beschrieben zu schnell, meldet OpenNI den User erfolgreich ab, hält an und wechselt in den *Idle State*.
- Läuft man um RooKi herum, ohne seine Distanz zu ihm mehr als 20cm zu ändern, dreht sich RooKi auf der Stelle.

7. Projekt Review

Alles in allem lässt sich sagen, dass das Projekt ein Erfolg war. Wie es nun mal in Projekten üblich ist, genossen wir viele motivierende Höhepunkte, hatten aber auch mit Misserfolgen und Tiefpunkten zu kämpfen.

Zu Beginn hatten wir viele Anfangsschwierigkeiten zu überstehen. Auf den Labor PCs versuchten wir alle Treiber und SDKs zu installieren, was aber aus ungeklärten Gründen nicht funktionierte. So entschlossen wir uns, auf den privaten Notebook zu arbeiten. Auch der Levelshifter funktionierte nicht mehr und musste mehrere Male überarbeitet werden. Als dies überstanden war konnten wir schnell erste Erfolge erzielen und das Projekt vorantreiben.

Die Herausforderung bestand besonders in der Grösse der zu schreibenden Software. In Praktika der HSR werden kleine Codeausschnitte zur Übung geschrieben. Ein grosses Projekt zu erstellen erfordert viel mehr Struktur. Diese Erfahrung, wie dies umgesetzt wird, war eine wertvolle Erfahrung.

A. CD Inhalt

- Source Code RooKi
- RooKi Helpfile
- Dokumentation in PDF und LaTeX Source
- TestGUI Kinect for MS
- Rohdaten der Distanzmessung mit MATLAB Script für die Auswertung
- RooKi Version zur Ausgabe der Werte der Mittelwert- und Normalisierungs-Funktion des *Auto Clean State*
- MATLAB Script zur Validierung der Mittelwert- und Normalisierungs-Funktion des *Auto Clean State*
- Verwendete Treiber und Framework Versionen
- Roomba Schnittstellenspezifikation
- Pflichtenheft
- Aufgabenstellung

B. Dokumente



Studienarbeit

SA-ESW11.02

HS 2011/12

Aufgabenstellung

für

Herr Yves Boillat

Herr André Reumer

Steuerung eines Roomba Roboters mit Hilfe von MS Kinect

1. Einführung

Kinect ist die Bezeichnung für ein Eingabegerät von Microsoft, welches für die Spielkonsole Xbox 360 entwickelt wurde. Mit Kinect ist es möglich, die Xbox durch Bewegungen (Gestures) und Sprachbefehle zu steuern.



Abbildung 1 Kinect HW

Die Kinect Hardware beinhaltet eine Farbkamera und eine monochrome Kamera, die zusammen mit einem IR-Laser die Distanz zu der Szenerie bestimmt (3D depth-camera). Mit diesen Informationen wird die Szenerie in allen drei Dimensionen erfasst und ausgewertet.

Neben den visuellen Sensoren besitzt es insgesamt vier Mikrophone für die Sprachbefehle. Durch die mechanische Anordnung der Mikrofone ist eine Lokalisierung der Geräuschquelle möglich.

Microsoft stellt eine API bereit, die Funktionen wie Gesichtserkennung, Bewegungsanalyse (skeleton tracking) und Spracherkennung bereitstellt. Eine OpenSource-Version ist ebenfalls erhältlich.

2. Aufgabenstellung

Das Ziel dieser Arbeit ist, die von Kinect verwendeten Technologien zu verstehen und Kinect an einem Computer zu betreiben. Mit den Daten, die der Kinect erfasst, soll der Putzroboter Roomba mittels Gestik gesteuert oder angewiesen werden, sich selbstständig und intelligent im Raum zu bewegen. Mit Hilfe des 3D Sensors sollen der vom Roboter durchfahrene Raum visualisiert und die im Raum befindlichen Gegenstände erkannt werden. Ebenfalls ist zu unterscheiden, wann eine Unebenheit im Boden ein unüberwindbares Hindernis darstellt oder wann dieses über oder unterfahren werden kann.



Abbildung 2 Roomba Staubsauger

Im Folgenden sind die einzelnen Aufgaben aufgelistet:

- Einarbeitung in die Kinect Hard- und Software (Beinhaltet die Kenntnis über die Technologien der 3D Kamera und die verwendeten Bildverarbeitungs-Algorithmen)
- Die Möglichkeiten mit Kinect quantitativ beschreiben (Informationen über Genauigkeit, Zuverlässigkeit, usw... ermitteln)
- Evaluation der vorhandenen SW-Bibliotheken und deren Vor- und Nachteile
- Entwickeln einer SW für die Aufgabenstellung
- Die entwickelte Applikation testen

3. Ablauf

Zu Beginn der Studienarbeit sind ein Projektplan sowie ein detailliertes Pflichtenheft zu erstellen, welches von allen beteiligten Parteien zu genehmigen ist. Anschliessend sind mögliche Lösungsansätze zusammenzutragen, im Vergleich objektiv einander gegenüberzustellen und zu bewerten. Die am besten geeignete Variante soll schliesslich ausgearbeitet, implementiert und detailliert getestet werden.

Bei mehrköpfigen Teams sollen die verschiedenen Tätigkeiten und Teilaufgaben innerhalb des Teams sinnvoll verteilt werden. Die konkrete Aufteilung ist im Projektplan klar und deutlich festzuhalten. Weitere Einzelheiten werden an den regelmässigen Besprechungen festgelegt und sind zu protokollieren.

4. Bericht

Über die Arbeit ist ein Bericht zu verfassen, dessen Textteil 60 Seiten nicht überschreiten soll. Im Bericht müssen alle wesentlichen gemachten Überlegungen, Abklärungen, Berechnungen und Untersuchungen detailliert (in Text und Bild) dokumentiert werden.

Der Bericht muss gut leserlich geschrieben und übersichtlich gegliedert sein. Er soll mindestens die folgenden Kapitel umfassen: Inhaltsverzeichnis, allgemeine (für nicht Fachleute) verständliche Einleitung (Abstract), Original der Aufgabenstellung, kommentierter Zeitplan (Soll) und Arbeitsfortschritt (Ist) inklusive Aufteilung der Arbeiten und Pflichtenheft. Der Aufbau des übrigen Teils des Berichtes mit einer Analyse der Aufgabenstellung und Bewertung der Lösungsmöglichkeiten, einer Konzept- und Realisierungsbeschreibung, sowie der zugrunde liegenden theoretischen Betrachtung ist der Aufgabe entsprechend zu gestalten. Zum Bericht gehören immer auch eine Systembeschreibung mit einer klaren Festlegung der Systemgrenze, Angaben darüber, wie das ganze System zu erstellen ist, sowie eine kommentierte Auflistung der zugehörigen Daten.

Der Bericht ist in 3 Papier-Exemplaren, sowie in elektronischer Form (in Source und als PDF-Dokument) abzugeben. Der Bericht und alle anfallenden Daten sind auf 3 CDs/DVDs zu archivieren und ebenfalls abzugeben.

5. Termine

KW 38 Montag, 19.09.2011 Ausgabe der Aufgabenstellung, Beginn der Arbeit

KW 51 Freitag, 23.12.2011 / 13:00 Abgabe Bericht, Ende der Arbeit

6. Besonderes

Während der ganzen Arbeit haben alle Studenten ein stets aktuelles, persönliches Laborjournal in Form eines Heftes zu führen. Darin sind Arbeitszeiten, Tätigkeiten, Erkenntnisse, Beschlüsse, usw. in chronologischer Reihenfolge festzuhalten. Das Laborjournal kann vom Betreuer jederzeit eingesehen und auch zur Gesamtbewertung beigezogen werden. Das Laborjournal ist am Ende der Arbeit zusammen mit dem Bericht abzugeben.

Wenn Programmcode erstellt wird, muss ein Versionsverwaltungstool eingesetzt werden.

Anschaffungen und Leistungen Dritter müssen rechtzeitig angemeldet und im Voraus bewilligt werden. Über die wöchentlichen Besprechungen ist ein Kurzprotokoll zu verfassen.

Der Erfolg in der Studienarbeit ist stark von der Zusammenarbeit aller Beteiligten abhängig. Eine gute Koordination und Kommunikation ist wichtig und muss von allen Seiten aktiv gefördert werden.

7. Bewertung

Bewertet werden: Arbeits-Methodik inkl. Projektplanung und -durchführung, technischer Inhalt, Resultat, Schlussbericht und Präsentationen. Ebenso werden die individuellen Laborjournale zur Bewertung beigezogen.

8. Organisatorisches

Betreuung der Arbeit: Prof. Reto Bonderer, Philipp Hörler

Industriepartner: IMES, HSR Rapperswil

Betreuung des Labors: Caspar Naef

Arbeitsplatz: Labor 6.007

Regelmässige Besprechungen: jeweils am Montag von 10:10 – 11:00 im 6.007

Rapperswil, September 2011

R. Bonderer

PFLICHTENHEFT

Steuerung eines Roomba Roboters mit Hilfe von MS Kinect

Autoren: Andre Reumer, Yves Boillat

Betreuer: Prof. Reto Bonderer

Aufgabenstellung: SA-ESW11.02

Themengebiet: Elektrotechnik – Embedded Software Engineering

Version: 1.0

Letzte Änderung: 18. Oktober 2011



Inhaltsverzeichnis

1	Zielbestimmung.....	3
1.1	Musskriterien	3
1.2	Sollkriterien	3
1.3	Wunschkriterien.....	3
1.4	Abgrenzungskriterien	3
2	Bewertungskriterien des Roomba und der Kinect	4
2.1	Untersuchung der Roomba Schnittstelle	4
2.2	Untersuchung der Kinect SDK	4
2.3	Untersuchung der OpenNI Plattform	5
3	Technische Rahmenbedingungen	5
3.1	Plattform	5
3.2	Schnittstellen.....	5
4	Produkteinsatz.....	5
4.1	Zielgruppen.....	5
4.2	Umgebungsbedingungen	5
5	Meilensteine.....	6
6	Beteiligte Personen	6
7	Freigabe.....	7

Änderungsindex

Version	Änderungen	Name	Datum
1.0	Zur Unterzeichnung aller Parteien	AR, YB	18.10.2011
0.5	Erste Version zur Überprüfung von allen Parteien	AR, YB	06.10.2011

1 Zielbestimmung

Das Ziel dieser Semesterarbeit (SA) besteht aus einer Untersuchung der Kinect. Hierbei werden verschiedene Eigenschaften evaluiert.

Daraus soll eine Schlussfolgerung gezogen werden, wie gut sich das Gerät zur Steuerung eines Roomba Roboters eignet. Der Roomba kann zwar schon selbstständig den Boden eines Raumes reinigen, hat aber Defizite im Fahrverhalten. Dies soll mit Kinect deutlich verbessert werden.

Gestützt auf den Untersuchungsresultaten wird im Anschluss eine Software entwickelt. Diese steuert den Roomba mittels 3D-Sensor und Gesten. Sie soll dann zu Demonstrationszwecken eingesetzt werden können.

1.1 Musskriterien

- Die Untersuchung muss zeigen, ob sich Kinect für diese Anwendung eignet.
 - Untersuchung der Roomba Schnittstelle (Siehe Kapitel 2.1)
 - Untersuchung der Kinect SDK und OpenNI Plattform (Siehe Kapitel 2.2)
 - Distanzlimitationen
 - Lichteinfluss
 - Gestenlimitationen
 - Motor
 - Software
- Es muss eine Software geschrieben werden, die den Roomba mit Hilfe von Kinect steuert.

1.2 Sollkriterien

- Während des Einarbeitens soll die Technologie der 3D Kamera studiert und verstanden werden.
- Grundsätzlich werden zwei Softwarebibliotheken untersucht. Die offizielle MS Kinect SDK, sowie OpenNI. Alle Eigenschaften werden mit beiden Bibliotheken untersucht und die Resultate verglichen. Somit soll eine Aussage über die Vor- und Nachteile gemacht werden können.
- Die zu schreibende Software kann Gegenstände im Raum erfassen, die es umfahren muss. Bei Unebenheiten des Bodens soll entschieden werden, ob das Hindernis über- oder unterfahren werden kann.
- Der Roboter soll mittels Gestik gesteuert werden können.
- Die entwickelte Applikation soll getestet werden.

1.3 Wunschkriterien

- Algorithmus zur effizienten Reinigung eines Raumes. Das heisst in einer möglichst kurzen Strecke eine möglichst grosse Raumabdeckung zu erzielen.
- Untersuchung der Audiofunktionen der Kinect (Siehe Kapitel 2.2)

1.4 Abgrenzungskriterien

- Im Rahmen dieser SA wird nicht eine marktreife Konstruktion gebaut. Der Aufbau des Notebooks und der Kinect auf dem Roomba ist als Laboraufbau anzusehen.
- Die Software dient dem Demonstrationszweck und ist nicht zwingend direkt auf ein Embedded System portierbar.

2 Bewertungskriterien des Roomba und der Kinect

2.1 Untersuchung der Roomba Schnittstelle

Spezifikationen

- Schnittstellenaufbau
- Sensoren

Betriebsmodi

- Geschwindigkeiten
- Befehlsmöglichkeiten

2.2 Untersuchung der Kinect SDK

Distanzlimitationen

- Maximale Distanz, welche noch erkannt wird (grosses Objekt)
- Minimale Distanz, welche noch erkannt wird (grosses Objekt)
- Winkelverschärfung des Tiefensensors
- Erfassungswinkel (X- und Y-Achse) des Tiefensensors
- Oberflächeneinfluss
 - Spiegel
 - Glas
 - Metall
 - Holz

Lichteinfluss

- Kein Licht
- Gegenlicht

Gestenlimitationen

- Skeletterkennung notwendig?
 - Voraussetzungen für erfolgreiches Erkennen

Motor

- Ansteuerung
- Limitationen
 - Drehwinkel (Horizontal)
 - Neigungswinkel (Vertikal)

Audio

- Benötigte Software
- Verschiedene Sprachen möglich?
- Wörter bestimmen möglich?

Software

- Gestenerkennung implementiert?
- Skeletterkennung implementiert?
- Bereitstellungsart der Tiefeninformationen für Implementierung in einem Programm
 - Komplettes Bild
 - Einzelne Punkte

2.3 Untersuchung der OpenNI Plattform

- Die gleichen Kriterien wie bei der Kinect SDK sind zu evaluieren
- Kriterien mit denjenigen der Kinect SDK vergleichen

3 Technische Rahmenbedingungen

3.1 Plattform

Die Untersuchung, sowie die Software beziehen sich auf das Kinect System für die XBOX 360 von Microsoft und den Staubsaugerroboter Roomba von iRobot.

Die Software läuft auf einem Notebook, an dem Kinect und der Roomba angeschlossen sind.

3.2 Schnittstellen

Die Kinect stellt eine USB-Verbindung zur Verfügung. Der Roomba verfügt über eine serielle Schnittstelle (RS-232), welche mit einem „Serial to USB Converter“ mit dem Notebook verbunden wird.

4 Produkteinsatz

Das Produkt dient der Untersuchung, sowie als Demonstration einer Verbindung von Kinect mit einem Roomba.

4.1 Zielgruppen

Die Zielgruppen der SA sind weitere Projekte, welche sich mit Kinect und dem Roomba beschäftigen, sowie auch die Schule.

4.2 Umgebungsbedingungen

Für diese SA gelten Laborbedingungen. Temperaturschwankungen oder spezielle Umgebungen werden nicht berücksichtigt.

Pflichtenheft Steuerung eines Roomba Roboters mit Hilfe von MS Kinect.

5 Meilensteine

Termin	Merkmal
Montag, 17.10.2011	Erfassung der Leistungsfähigkeit von Kinect abgeschlossen und es liegt einen begründeten Entscheid für Kinect SDK oder OpenNI vor.
Montag, 24.10.2011	Grundstruktur und Klassen der Software sind definiert.
Montag, 31.10.2011	Die Software-Architektur (Design) steht fest und soll in der Besprechung freigegeben werden. Nach der Freigabe kann mit der Implementation begonnen werden.
Samstag, 19.11. 2011	Die entwickelte Software soll an den Robolympics demonstriert werden.
Freitag, 23.12.2011 13:00	Abgabe Bericht, Ende der Arbeit

6 Beteiligte Personen

Name	Funktion	Zugehörigkeit	E-Mail-Adresse
Prof. Reto Bonderer	Auftraggeber	HSR	rbondere@hsr.ch
Philipp Hörler	Projektbetreuer	HSR	phoerler@hsr.ch
Andre Reumer	Auftragnehmer	Student HSR	areumer@hsr.ch
Yves Boillat	Auftragnehmer	Student HSR	yboillat@hsr.ch

Pflichtenheft

Steuerung eines Roomba Roboters mit Hilfe von MS Kinect

7 Freigabe

Auftraggeber

Prof. Reto Bonderer

Rapperswil, 7.11.2011 R. Bonderer
Ort, Datum Unterschrift

Projektbetreuer

Philipp Hörler

7. November 2011 P. Hörler
Ort, Datum Unterschrift

Auftragnehmer

Andre Reumer

Rapperswil, 7. 11. 2011 A. Reumer
Ort, Datum Unterschrift

Yves Boillat

Rapperswil, 07.11.2011 Y. Boillat
Ort, Datum Unterschrift

C. Messungen

C.1. Messwerte Distanzmessungen Kinect for Microsoft

Wenn nicht anderst angegeben sind die Messwerte als mm anzusehen.

	0.8m	1m	2m	3m	3.8m
	805	1019	2009	2980	3710
	815	1016	2045	3143	3975
	824	1013	2045	1920 (Boden)	1878 (Boden)
	809	1013	2021	3006	3800
	818	1010	2033	3059	3200 (Boden)
	809	1004	2033	3115	3975
	809	1019	2021	3059	3885
	820	1013	2009	3087	3200 (Boden)
	811	1016	2033	3059	3885
	809	1004	2009	3059	3885
	820	1004	2001	1941 (boden)	1899 (Boden)

C.2. Messwerte Distanzmessungen OpenNI

Hierbei ist zu beachten, dass ab einer Distanz von 3m die untersten zwei Messpunkte den Boden detektierten. Ab vier Metern die untersten vier Werte. Diese wurden in der statistischen Betrachtung nicht miteinbezogen. Wenn nicht anderst angegeben sind die Messwerte als Millimeter anzusehen.

	0.52m	1m	2m	3m	4m	5m	6m
	510	969	1947	2942	3864	4906	5915
	520	994	1947	2942	3821	4838	5915
	523	985	1969	2942	3952	4977	5915
	516	991	1958	2967	3952	4906	6124
	527	1000	2039	3157	4192	5364	6468
	520	997	2003	3046	4093	5202	6335
	520	1011	1969	2942	3952	4838	5915
	539	1017	2027	3019	3101	3129	3073
	532	1023	2015	3019	3157	3216	3246
	548	1027	1843	1843	1833	1833	1824
	544	1048	1843	1843	1843	1833	1904

Folgende Messungen wurden mit NiViewer aufgenommen. Sie zeigen die Abweichung der Tiefeinwerte vom Mittelwert in Millimeter auf verschiedene Distanzen.

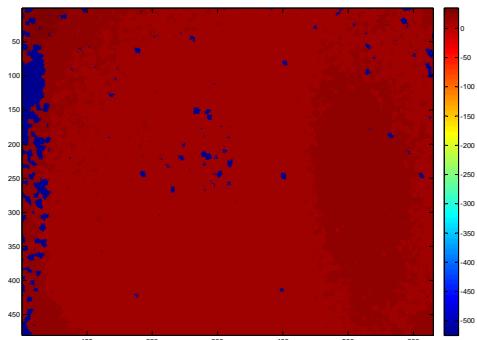


Abbildung C.1: 0.5m Distanz

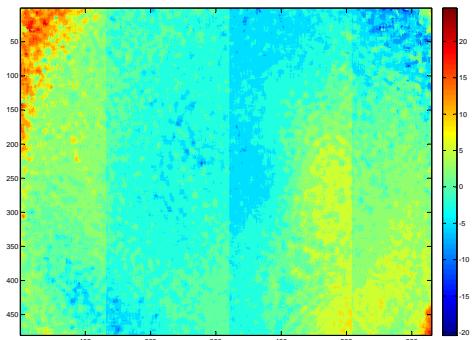


Abbildung C.2: 1m Distanz

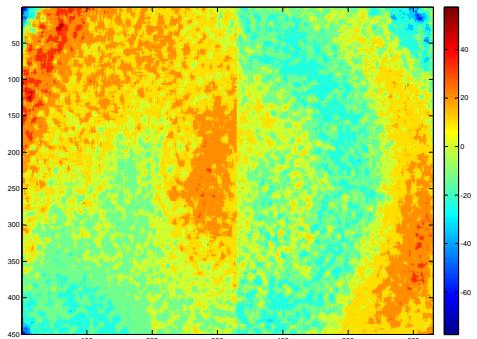


Abbildung C.3: 2m Distanz

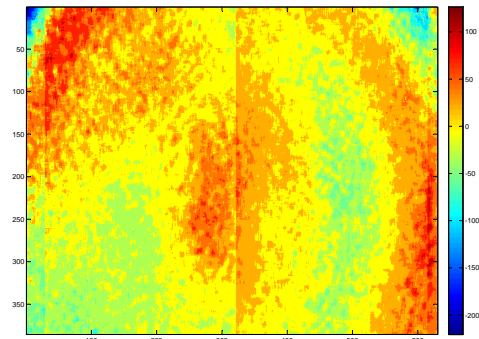


Abbildung C.4: 3m Distanz

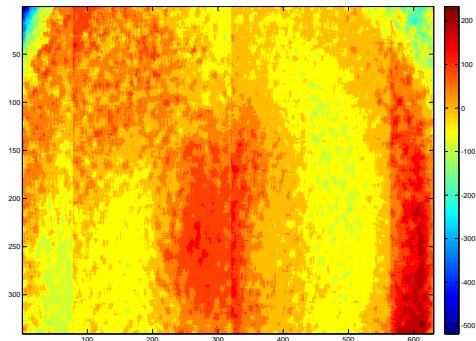


Abbildung C.5: 4m Distanz

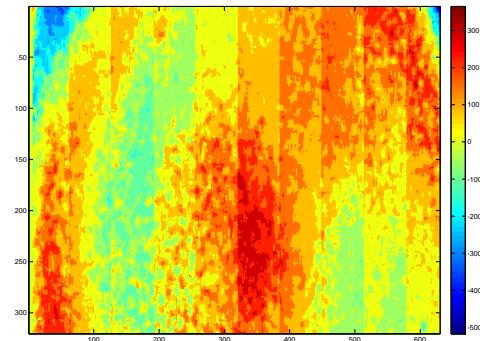


Abbildung C.6: 5m Distanz

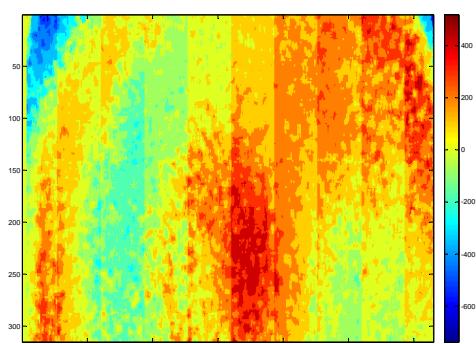


Abbildung C.7: 6m Distanz

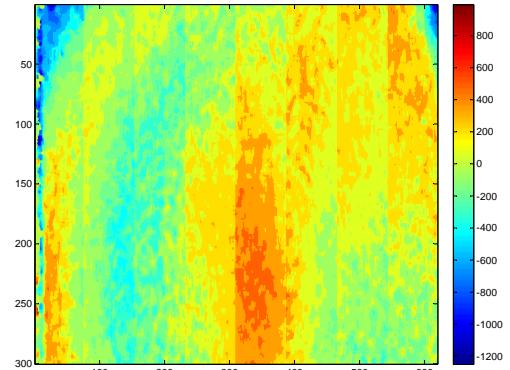


Abbildung C.8: 7m Distanz

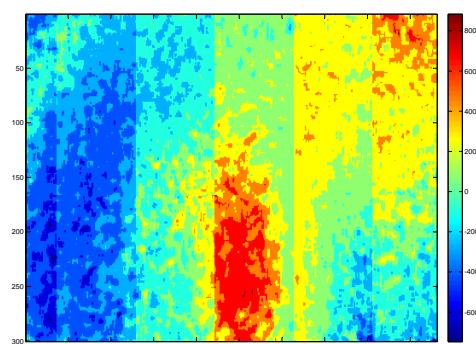


Abbildung C.9: 8m Distanz

D. Roomba

Packet	Name	Bytes	Value Range	Units
7	Bumps Wheeldrops	1	0 - 15	
8	Wall	1	0 - 1	
9	Cliff Left	1	0 - 1	
10	Cliff Front Left	1	0 - 1	
11	Cliff Front Right	1	0 - 1	
12	Cliff Right	1	0 - 1	
13	Virtual Wall	1	0 - 1	
14	Overscurrents	1	0 - 29	
15	Dirt Detect	1	0 - 255	
16	Unused 1	1	0 - 255	
17	Ir Opcode	1	0 - 255	
18	Buttons	1	0 - 255	
19	Distance	2	-32768 - 32767	mm
20	Angle	2	-32768 - 32767	degrees
21	Charging State	1	0 - 5	
22	Voltage	2	0 - 65535	mV
23	Current	2	-32768 - 32767	mA
24	Temperature	1	-128 - 127	deg C
25	Battery Charge	2	0 - 65535	mAh
26	Battery Capacity	2	0 - 65535	mAh
27	Wall Signal	2	0 - 4095	
28	Cliff Left Signal	2	0 - 4095	
29	Cliff Front Left Signal	2	0 - 4095	
30	Cliff Front Right Signal	2	0 - 4095	
31	Cliff Right Signal	2	0 - 4095	
32	Unused 2	1	0 - 255	
33	Unused 3	2	0 - 65535	
34	Charger Available	1	0 - 3	
35	Open Interface Mode	1	0 - 3	
36	Song Number	1	0 - 4	
37	Song Playing?	1	0 - 1	
38	Oi Stream Num Packets	1	0 - 108	
39	Velocity	2	-500 - 500	mm/s
40	Radius	2	-32768 - 32767	mm
41	Velocity Right	2	-500 - 500	mm/s
42	Velocity Left	2	-500 - 500	mm/s
43	Encoder Counts Left	2	0 - 65535	
44	Encoder Counts Right	2	0 - 65535	
45	Light Bumper	1	0 - 127	
46	Light Bump Left	2	0 - 4095	
47	Light Bump Front Left	2	0 - 4095	
48	Light Bump Center Left	2	0 - 4095	
49	Light Bump Center Right	2	0 - 4095	
50	Light Bump Front Right	2	0 - 4095	
51	Light Bump Right	2	0 - 4095	
52	Ir Opcode Left	1	0 - 255	
53	Ir Opcode Right	1	0 - 255	
54	Left Motor Current	2	-32768 - 32767	mA
55	Right Motor Current	2	-32768 - 32767	mA
56	Main Brush Current	2	-32768 - 32767	mA
57	Side Brush Current	2	-32768 - 32767	mA
58	Stasis	1	0 - 1	

Abbildung D.1: Roomba Sensordaten

E. Verzeichnisse

Abbildungsverzeichnis

2.1. Projektstrukturplan	10
3.1. Pinbelegung Roomba	15
4.1. Kinect Aufbau	17
4.2. Kinect IR-Bild	18
4.3. IR-Bild Ausschnitt	18
4.4. Schattenbildung im Tiefenbild	19
4.5. Schattenbildung	19
4.6. Nullband	20
4.7. Messung des Lichteinfluss vor dem Fenster	21
4.8. Vergleich mit und ohne Sonnenlicht	21
4.9. Demonstration des Sonnenlichteinflusses	21
5.1. Kinect for Microsoft Architektur	22
5.2. Struktur OpenNI	23
5.3. TestGUI	24
5.4. Messung der Minimaldistanz	25
5.5. Messung über weitere Distanzen	27
5.6. Anordnung der Messpunkte	28
5.7. Standardabweichung der Tiefenwerte	28
5.8. Standardabweichung der Tiefenwerte	29
6.1. Use Case Diagram RooKi	39
6.2. Levelshifter RS-232-TTL	40
6.3. Aufbau des RooKi	40
6.4. Auto Clean Verhalten (rechts)	45
6.5. Auto Clean Verhalten (links)	45
6.6. Zeilenverarbeitung des Auto Clean Algorithmus	46
6.7. Auto Clean Struktogramm	46
6.8. Detection Pose	47
6.9. Kurvenradius	48
6.10. Entscheidungsprozedur des Fahrbefehls	49
6.11. GUI nach Öffnen	55
6.12. GUI nach Start	56
C.1. 0.5m Distanz	73
C.2. 1m Distanz	73
C.3. 2m Distanz	73

C.4. 3m Distanz	73
C.5. 4m Distanz	74
C.6. 5m Distanz	74
C.7. 6m Distanz	74
C.8. 7m Distanz	74
C.9. 8m Distanz	74
D.1. Roomba Sensordaten	75

Tabellenverzeichnis

2.1. Meilensteine	10
5.1. Elevation Angle der Buttons	24
5.2. Minimaldistanzen Kinect for Microsoft	25
5.3. Minimaldistanzen OpenNI	26
5.4. Mittelwerte und Standardabweichungen Kinect for Microsoft	27
5.5. Mittelwerte und Standardabweichungen OpenNI	28
5.6. Mittelwert und Standardabweichung OpenNI über ganze Map	29
6.1. Event Buttons	56

Listings

6.1. Singleton	43
6.2. Lesen von .raw Files	52