DHBW
Duale Hochschule
Baden-Württemberg

Autonome Navigation (im Indoor-Bereich) der selbstbalancierenden Plattform RobStep

Studienarbeit

für die Prüfung zum

Bachelor of Engineering

des Studiengangs Informationstechnik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Philip Hug & Simon Simon

Mai 2016

Bearbeitungszeitraum	5tes & 6tes Semester
Matrikelnummer, Kurs	4815162342, TINF13B3
Ausbildungsfirma	Firma GmbH, Firmenort
Betreuer	Prof. Dr. Markus Strand
Gutachter	-

Sperrvermerk

Die vorliegende Studienarbeit mit dem Titel *Autonome Navigation (im Indoor-Bereich) der selbstbalancierenden Plattform RobStep* enthält unternehmensinterne bzw. vertrauliche Informationen der Firma GmbH, ist deshalb mit einem Sperrvermerk versehen und wird ausschließlich zu Prüfungszwecken am Studiengang Informationstechnik der Dualen Hochschule Baden-Württemberg Karlsruhe vorgelegt. Sie ist ausschließlich zur Einsicht durch den zugeteilten Gutachter, die Leitung des Studiengangs und ggf. den Prüfungsausschuss des Studiengangs bestimmt. Es ist untersagt,

- den Inhalt dieser Arbeit (einschließlich Daten, Abbildungen, Tabellen, Zeichnungen usw.) als Ganzes oder auszugsweise weiterzugeben,
- Kopien oder Abschriften dieser Arbeit (einschließlich Daten, Abbildungen, Tabellen, Zeichnungen usw.) als Ganzes oder in Auszügen anzufertigen,
- diese Arbeit zu veröffentlichen bzw. digital, elektronisch oder virtuell zur Verfügung zu stellen.

Jede anderweitige Einsichtnahme und Veröffentlichung – auch von Teilen der Arbeit – bedarf der vorherigen Zustimmung durch den Verfasser und Firma GmbH.

Karlsruhe, Mai 2016

Philip Hug & Simon Simon

Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meine Studienarbeit mit dem Thema *Autonome Navigation (im Indoor-Bereich) der selbstbalancierenden Plattform RobStep* ohne fremde Hilfe angefertigt habe;
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe;
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe;
4. dass die eingereichte elektronische Fassung exakt mit der eingereichten schriftlichen Fassung übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Karlsruhe, Mai 2016

Philip Hug & Simon Simon

Abstract

Abstract normalerweise auf Englisch. Siehe: http://www.dhbw.de/fileadmin/user/public/Dokumente/Portal/Richtlinien_Praxismodule_Studien_und_Bachelorarbeiten_JG2011ff.pdf (8.3.1 Inhaltsverzeichnis)

Ein „Abstract“ ist eine prägnante Inhaltsangabe, ein Abriss ohne Interpretation und Wertung einer wissenschaftlichen Arbeit. In DIN 1426 wird das (oder auch der) Abstract als Kurzreferat zur Inhaltsangabe beschrieben.

Objektivität soll sich jeder persönlichen Wertung enthalten

Kürze soll so kurz wie möglich sein

Genauigkeit soll genau die Inhalte und die Meinung der Originalarbeit wiedergeben

Üblicherweise müssen wissenschaftliche Artikel einen Abstract enthalten, typischerweise von 100-150 Wörtern, ohne Bilder und Literaturzitate und in einem Absatz.

Quelle: http://de.wikipedia.org/wiki/Abstract Abgerufen 07.07.2011

Diese etwa einseitige Zusammenfassung soll es dem Leser ermöglichen, Inhalt der Arbeit und Vorgehensweise des Autors rasch zu überblicken. Gegenstand des Abstract sind insbesondere

- Problemstellung der Arbeit,
- im Rahmen der Arbeit geprüfte Hypothesen bzw. beantwortete Fragen,
- der Analyse zugrunde liegende Methode,
- wesentliche, im Rahmen der Arbeit gewonnene Erkenntnisse,
- Einschränkungen des Gültigkeitsbereichs (der Erkenntnisse) sowie nicht beantwortete Fragen.

Quelle: http://www.ib.dhbw-mannheim.de/fileadmin/ms/bwl-ib/Downloads_alt/Leitfaden_31.05.pdf, S. 49

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Theorieinhalte

1.1 Kinect

Bei der Microsoft Kinect handelt es sich um einen kombinierten Bildsensor für den Consumermarkt. Der von Microsoft ursprünglich konzipierte Einsatzzweck ist für die Integration von Gestensteuerung in Videospielen in Kombination mit einer Microsoft Xbox 360 Konsole. Erste Details des Gerätes wurden zunächst noch unter dem Workingtitle Project Natal veröffentlicht. Die Prototypen entstanden in Zusammenarbeit mit PrimeSense. Der sehr günstige Preis, mittlerweile 20 Euro für ein gebrauchtes Gerät, sowie quelloffene Treiber und Libraries machen die Kinect zu einem sehr beliebten Baustein zahlreicher Hobby- und Forschungsprojekten.

1.1.1 Fotosensor

Die Kinect verfügt über eine Kombination von Farb- und Tiefenkamera. Beim Farbsensor handelt es sich um einen herkömmlichen VGA-Sensor welcher Farbwerte im RGB-24Bit Farbraum liefert. Bilder werden zunächst mit einer Auflösung von $1280 * 960$ Pixel aufgenommen, anschließend allerdings auf $640 * 480$ Pixel per downsampling herunterreduziert. Das Aufnahmefeld erstreckt sich über 57 Grad Horizontal und 43 Grad Vertikal. [**kinect-georg**]

1.1.2 Tiefenbild-System

Tiefenbilder werden durch eine Kombination von IR-Laser Emitter und Sensor erzeugt. Der Emitter bestrahlt die Umgebung in einem Winkel von 58Grad Horizontal und 45Grad Vertikal mit einem pseudorandom IR-Muster. Diese Technik funktioniert in Bereichen mit viel natürlich Licht nur mit schweren Einschränkungen. Natürliches Licht enthält ebenfalls Infrarote Strahlung, welche zu Interferenzen mit dem IR-Muster des Emitters führt. Der Sensor kann somit das erzeugte Mesh nicht mehr eindeutig identifizieren und erzeugt somit ein extrem chaotisches Bild, welches sich durch starkes Rauschen äußert. Der Laser des Emitters arbeitet mit einer Leistung von 70mW und ist somit nit eyesafe. PrimeSense

hat eine patentierte Methode entwickelt welche den Einsatz ohne Schutzbrille ermöglicht. Laserlicht wird in der Regel mit einer Diode erzeugt. Dioden erzeugen punktgerichtete Strahlen. Dadurch ist bei der Betrachtung des Meshes im Bildzentrum ein Punkt mit höherer Intensität als die restlichen Lichtpunkte zu erkennen. Durch die scattering genannte Methode wird dieser zentrale Punkt auf 9 unterschiedliche Punkte durch Streuung verteilt. Dadurch wird auch die Intensität der einzelnen Scatter-Points auf $\frac{1}{9}$ reduziert, was es für das menschliche Auge unbedenklich macht. Der Emitter erzeugt zunächst ein Mesh mit einer Auflösung von $1200 * 960$ Pixeln. Im Nachhinein wird die Auflösung allerdings durch downsampling auf $640 * 490$ Pixel reduziert. Dies ist nötig da der USB-Stack das begrenzende Medium darstellt, und ansonsten die zeitgerechte Übertragung des Farb sowie des Tiefenbildes nicht sichergestellt werden kann. Microsoft gibt einen Funktionsbereich von 0,8m bis 3,5m an, welcher sich allerdings in praktischen Anwendungen auf 0,5 bis 3m eingependelt hat. Die Monochrom Kamera arbeitet mit einer nativen Auflösung von $1280 * 1024$ Pixel, welche allerdings bereits vor dem downsampling verkleinert wurde. Die Tiefenbilder enthalten Tiefenwerte im Wertebereich von 13Bit. [**kinect-hacking**]

1.1.3 Funktionsweise

light coding

Das Tiefensystem der Kinect unterstützt keine Laufzeitmessung. Stattdessen wird *structured light* verwendet. Dabei wird ein vorher definiertes Muster durch einen Emitter auf eine Fläche projiziert. Das Muster ist aus Punkten zusammengesetzt. Innerhalb des Musters gibt es keine Wiederholungen, was dazu führt dass jeder Punkt innerhalb des Bildes eindeutig identifziert werden kann. Bei der Kinect kommt entgegen des herkömmlichen Vorgehens zwei versetzte Bildsensoren zu verwenden nur ein einzelner zum Zug. Das Bild des fehlenden Sensors wird durch ein hart-codiertes Bild substituiert. Dieses Bild wird als Referenzobjekt für den Wertevergleich mit dem vom Sensor empfangenen Muster genutzt. Dabei werden per stereo triangulationsverfahren die räumlichen Unterschiede der in beiden bildern bekannten Punkte berechnet. Hierbei dienen die Intensitätsunterschiede der Punkte als Berechnungsgrundlage. Ist der vom Sensor empfangene Punkt intensiver als der selbe Punkt im Referenzbild, ist der Punkt näher am Sensor. Bei schwächerer Intensität liegt der reale Punkt tiefer im Raum als der Referenzpunkt.
[**kinect-georg**] [**alpha-centauri-ueberlicht**] [**kinect-uug-chem**]

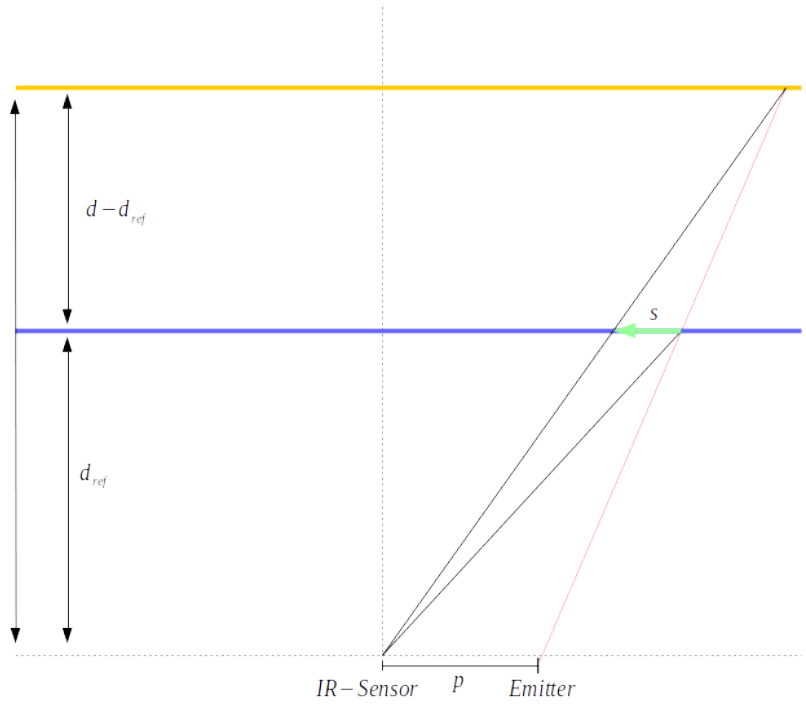


Abbildung 1.1: Illustration des light coding Prizips des Kinect Tiefenbildsystems.

Die Werte für d_ref (Tiefe des Referenzbildes) und p (horizontale Distanz zwischen Emitter und Monochromer Kamera) sind bekannt. Die Strecke s (horizontale Verschiebung zwischen Referenz- und Emitterbild) kann durch die eindeutig identifizierbaren Punkte errechnet werden. Die Tiefe d des Punktes im Emitterbild kann durch $d = \frac{p*d_ref}{s-p}$ berechnet werden. [**kinect-uug-chem**]

1.2 Robot Operating System

Das Akronym ROS steht für Robot Operating System. ROS ist kein wirkliches Betriebssystem auch wenn es einige Aufgaben eines OS übernimmt. Beispielsweise übernimmt es die Abstraktion von Hardware. Zusätzlich fungiert ROS wie eine Laufzeitumgebung. Es können Programme in verschiedenen Programmiersprachen wie C++, Python oder Lisp geschrieben werden. Diese Programme können die Infrastrukutur des ROS-Systems für

eigene Zwecke nutzen, erweitern das System gleichzeitig auch um dessen implementierte Funktionalität. Dies geschieht über das Message System von ROS.

1.2.1 topics

Die Kommunikation innerhalb des ROS-Systems erfolgt über eine peer2peer-artige Struktur. Dies erfolgt über einen eigenen Bus für jedes Thema, im ROS-System *topic* genannt. Die *topics* dienen gleichzeitig als Schlagwort für die Kommunikation. Die Kommunikation über *topics* findet zwischen *publisher* und *subscriber* statt. Dabei meldet der publisher dem zentralen Knoten des ROS-Systems dass Nachrichten über ein bestimmtes *topic* ausgetauscht werden müssen. Nun können andere Publisher Nachrichten mit dem Namen des Topics als *Betreff* auf den betreffenden Bus senden. Knoten für welche die Nachrichten auf dem BUS von interesse sind, melden sich als Subscriber am Bus an, und bekommen eine Benachrichtigung sobald eine neue Nachricht auf dem Bus liegt. Die Nachrichten im ROS-System als *messages* bezeichnet, haben einen fest definierten Inhalt. Die Definition eigener Nachrichtentypen ist ebenfalls möglich sowie das Nutzen vorbereiteter Nachrichten. Üblicherweise verfügt jedes Paket über eigene angepasste Nachrichtentypen. Ein Beispiel über den Aufbau einer ROS-*message* wird im Kapitel *Hindernisse* unter dem Abschnitt *Unterschiede Bilddatentypen* aufgeführt.

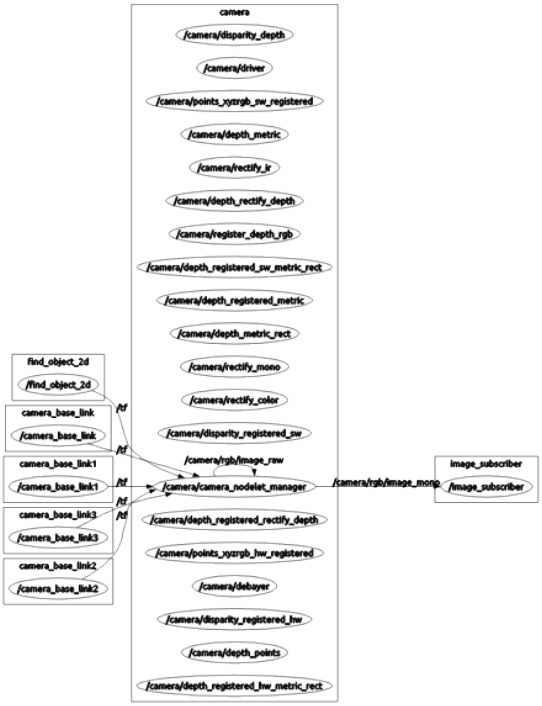


Abbildung 1.2: Überblick über die Topics sowie Subscriber/Publisher Nodes eines ROS-Systems.

1.2.2 nodes

Knoten bzw *nodes* sind ausführbare Programmdateien welche Aufgauben im ROS-System übernehmen. Sie erfüllen in der Regel genau einen Zweck, was dem modularen Aufbau des ROS-Systems zuspielt. Die zentrale Funktionalität wird über den ROS-Master gereitgestellt. Er stellt auch die Kommunikationsbusse für *topics* bereit.

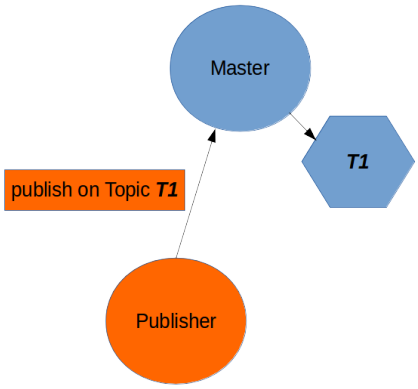


Abbildung 1.3: Initialer Publisher, sendet an Master dass er auf Topic T1 publishen will. Master erstellt darauf den Topic T1.

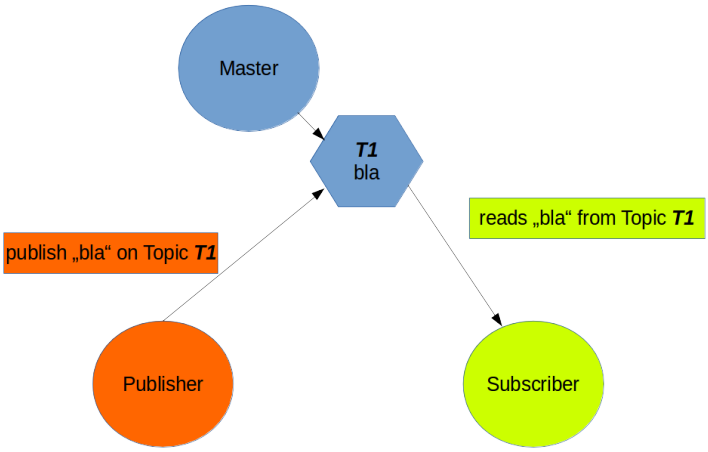


Abbildung 1.4: Publisher sendet *bla* an Topic T1, Subscriber liest von T1.

2 Problemstellung

In den meisten Projekten die sich mit autonomen Systemen befassen ist die Intelligenz sehr stark an den Roboter bzw. die Basisplattform gebunden. Die Aufgabe dieses Projektes soll es sein erste Versuche darin zu unternehmen eine physikalische Trennung zwischen Intelligenz und mobiler Plattform zu erreichen. Genauer soll die Steuerung des Roboters in eine periphere Infrastruktur verlagert werden.
Dadurch entstehen folgende Aufgabenbereiche:

- Die Kommunikation zwischen den beteiligten Elementen mobile Plattform, intelligente Infrastruktur, Benutzer sicherstellen.
- Die Integration des Parallelprojekts von Daniel Geiger TINF13B2
- Sinnvolle Architektur der Infrastruktur
- Integration Erfassender Komponenten in die Infrastruktur
- Basis für die Wegfindung der mobilen Plattform
- Erkennen von Hindernissen innerhalb des Bewegungsraums
- Reaktion auf erfasste Hindernisse
- Identifikation und Ortung der Plattform im Bewegungsraum

2.1 Erwartetes Ergebnis

Das erwartete Ergebnis dieses Projekts ist dass sich die mobile Plattform innerhalb einer begrenzten und definierten Umgebung autonom bewegen kann. Sie reagiert selbstständig auf auftauchende Hindernisse. Das autonome Bewegen soll durch einen benutzerseitigen Trigger initiiert werden.

2.2 Aufgabenverteilung

Jeder der Studienarbeiter bearbeitet selbstständig einen Teilbereich des Gesamtprojekts. Andere Aufgaben werden in der Gruppe gelöst.

2.2.1 Gemeinsame Aufgabenbereiche

- Sinnvolle Architektur der Infrastruktur
- Die Integration des Parallelprojekts von Daniel Geiger TINF13B2
- Die Kommunikation zwischen den beteiligten Elementen mobile Plattform, intelligente Infrastruktur, Benutzer.

2.2.2 Teilbereiche Simon Simon

- Erkennen von Hindernissen innerhalb des Bewegungsraums
- Reaktion auf erfasste Hindernisse
- Identifikation und Ortung der Plattform im Bewegungsraum
- Integration Erfassender Komponenten in die Infrastruktur
- Interaktion mit mobiler Plattform hinsichtlich Hindernissen

3 Basis Konzept

Die Grundlegende Idee besteht darin die Intelligenz aus dem Roboter herauszunehmen und stattdessen eine intelligente Infrastruktur in eine definierte Umgebung zu integrieren. Dieser Ansatz löst das Platzproblem auf mobilen Plattformen. Auf bzw in Robotern verbaute Rechnersysteme sind primär Platz und Ressourcen-schonend. Die Leistung solcher Geräte ist allerdings sehr beschränkt. Mit dem Auslagern in die Umgebung stehen mehr Platz und somit auch mehr Leistung zur Verfügung. Dies eröffnet zudem andere Lösungsansätze für bekannte Probleme. Gleichzeitig wird die Kommunikation mit der mobilen Plattform deutlich Komplexer, da durch die geschaffene Entfernung Kennwerte wie Latenz enorm an Gewicht zulegen. Aus diesem Grund soll die Basis des Systems auf ROS aufbauen.

3.1 Kommunikation

Die Kommunikation innerhalb des Systems nutzt die von ROS bereitgestellten Komponenten. Diese nutzen Netzwerktechnologien und bauen auf den Netzwerkstack des Betriebssystems auf. Da bei diesem Projekt die Flexibilität der mobilen Plattform erhalten bleiben soll, wird als Übertragungsmedium *IEEE 802.11* (W-LAN / WIFI) benutzt.

3.2 Eventhandling

Das Eventhandling soll über einen zentralen Service innerhalb des ROS-Systems erfolgen. Ereignisse und Nachrichten werden in ROS in so genannten Topics verbreitet. Dieser Dienst soll die relevanten definierten Topics des Systems überwachen und entsprechend darauf reagieren.

3.3 Wahrnehmen der Umgebung

Die visuelle Schnittstelle des Systems wird eine Kinect Kamera von Microsoft sein. Diese Entscheidung wurde augrund der sehr guten Dokumentation des Gerätes getroffen. Zusätz-

lich findet die Kinect eine große Unterstützung in vielen Projekten. Dies geht zweifelsfrei auf die freien Treiber und Bibliotheken zurück. Ein weiterer Grund besteht darin dass nur sehr wenige Geräte sowohl über einen herkömmlichen RGB als auch Tiefensensor verfügen. Zuletzt bietet die Kinect v1 ein sehr gutes Preis/Leistungs - Verhältnis.

3.4 Aufbau der Infrastruktur

Abbildung 3.1: Schematischer Aufbau der geplanten Infrastruktur.

4 Wegfindung

5 Hindernisse

Die Interaktion mit seiner Umwelt ist ein essentieller Teil in der Anwendung autonomer Systeme. Besonders der Sicherheitsaspekt spielt hier eine primäre Rolle. Die Akzeptanz der potentiellen Nutzer wäre stark beeinträchtigt wenn die Unversehrtheit von umstehenden Lebewesen und Gegenständen nicht sichergestellt werden kann. Die Komplexität bei Umweltinteraktion zeigt sich besonders beim Auftreten dynamischer Events. Ein Beispiel hierfür ist das Erscheinen von Hindernissen im Bewegungsraum einer mobilen Plattform. Um auf solche Ereignisse angemessen zu reagieren müssen dynamische Objekte erkannt, verfolgt und identifiziert werden. Bei der Entkopptlung der Intelligenz vom Roboter selbst, eröffnet sich ein weiteres Problem. Die Unterscheidung zwischen einem dynamischen Objekt und der mobilen Plattform. Im folgenden Kapitel soll ein Konzept für ein solches Szenario erarbeitet und umgesetzt werden.

5.1 Definition von Hinderniss

Die Definition lautet wie folgt: Ëtwas, was das direkte Erreichen eines Ziels, das Weiterkommen be- oder verhindert.: [**duden-hinderniss**] Dabei wird offengelassen welche Ursache das Hinderniss sein kann, Gegenstände und Lebewesen werden hier nicht unterschieden, somit wird im weiteren Verlauf lediglich die neutrale Form Objekt genutzt. Genauer von Objekten welche den Bewegungsraum der Mobilen Plattform betreten bzw. sich bereits darin befinden. Zusätzlich müssen Objekte anhand ihres Verhaltens Klassifiziert werden. Dies ist notwendig da nicht auf jede Situation mit dem selben Vorgehen reagiert werden kann. Dadurch entsteht die Möglichkeit auf Basis der definierten Klassen eine Fallunterscheidung mit dafür angepassten Reaktionsroutinen zu entwerfen.

- dynamisches Objekt = Lebewesen oder Gegenstand welches den Bildbereich der Kamera betritt. Bewegt sich kontinuierlich auf einem festen Pfad durch den Bewegungsraum der Plattform.
- statisches Objekt = Lebewesen oder Gegenstand welches in den Bewegungsraum der Plattform betritt und dort an einem festen Punkt verweilt.

- chaotisches Objekt = Lebewesen oder Gegenstand welches ohne vorhersehbares Muster im Bewegungsraum der Plattform umherwandelt, und in unregelmäßigen Abständen für eine undefinierbare Zeit an einem fixen Punkt verweilt.
- Blitz Objekt = Lebewesen oder Gegenstand welches den Bewegungsraum der Mobilen Plattform nur sehr flach durchstreift i.e. den Bewegungsraum nicht tief betritt oder ihn lediglich tangiert, aber dennoch von der Raumüberwachung erfasst wird.

5.2 Objekte erkennen

Die Grundlage für jede Interaktion mit der Umgebung, unabhängig davon ob nun von einem Roboter oder einem Lebewesen gesprochen wird, bildet die Voraussetzung zu Erkennen und zu Verstehen was darin vorgeht. Bei Lebewesen wie Robotern geschieht dies über die Audio-Visuellen Schnittstellen des Körpers bzw. des Systems.

5.2.1 Konzeptionelle Lösungsansätze

Der Grundsätzliche Ansatz besteht darin die Umgebung mit einem Bildsensor zu überwachen und Veränderungen sichtbar zu machen und zu analysieren. Dies geschieht durch den Abgleich von Inhalten über eine Serie von Bildern. Dieses Verfahren wird in einigen Publikationen als Frame-Analysis bezeichnet. Die Unterschiede der implementierten Vertreter dieses Verfahrens liegen in den beobachteten Merkmalen der Bilder. Im Verlauf des Projekts wurden drei Varianten verglichen. Für die Implementierung wurde OpenCV genutzt. Beweggründe für diese Entscheidung können dem Theoriekapitel über OpenCV entnommen werden.
(ADD) Verweiss auf OpenCV-ref (ADD)

5.2.2 Sequential Images

Eine Variante lautet *sequential images*, dt. *auf einander folgende Bilder*. Dabei wird eine Folge von Bildern, im Regelfall zwei f_i & $f_i + 1$, im gesamten verglichen. Eine Möglichkeit dies zu tun besteht darin für jeden Pixel p_xy der beiden Bilder die Differenz der Grauwerte zu erzeugen. Grauwerte deshalb da *sequential images* monochrome bzw. Graufstufenbilder als Eingabe erfordern. Graustufenbilder eignen sich hierfür im Gegensatz zu Farbbildern aufgrund der Beschränkung auf 2 Maxima, schwarz und weiß, mit n vielen (grauen) Zwischenwerten. Dadurch laesst sich die Differenz durch eine simple Subtraktion der Grauwerte erreichen. Das RGB Farbmodell hingegen verfügt über 8 Maxima, von denen 6 Maxima jeweils über eine Komplementärefarbe innerhalb des Farbmodells verfügen. Dies

erschwert die Differenzbildung da nur in bestimmten Fällen durch eine Subtraktion der vorhandene Farbwert neutralisiert wird. Stattdessen würde man in vielen Fällen lediglich einen anderen Farbwert erhalten. Das Ziel der Differenzbildung besteht jedoch darin die gemeinsamen Merkmale die in beiden aufeinander folgenden Bildern auftauchen zu entfernen, und nur die einzigartigen Merkmale des nachfolgenden Bildes zu bewahren.
Sind die Grauwerte $g_p(f_i)$, $g_p(f_i+1)$ von Pixel p_xy in f_i und f_i+1 identisch wird der Wert für diesen Pixel auf 0 (schwarz) gesetzt. Dabei handelt es sich um den Idealfall. In der reellen Anwendung dieses Verfahrens werden die Werte nur stark reduziert da durch die Dynamik der Lichtverhältnisse die Grauwerte des Pixels zwar nur marginal dafür allerdings konstant schwanken und dadurch die Wahrscheinlichkeit für das Eintreffen des Idealfalls $g_p(f_i) = g_p(f_i+1)$ nur sehr gering ist.

Abbildung 5.1: Ausgangsbild in monochromatischer Form.

Abbildung 5.2: Gefiltertes Differenzbild erzeugt aus dem Referenzbild f_i und dem Nachfolger $f_i + 1$.

5.2.3 find Contour

Anders als *sequential images* arbeitet *find contour* mit Farbbilden. Die Grundmechanik besteht darin den Verlauf von Farbwerten zu verfolgen. Hierbei werden zwei Arten der Farbwertänderung detektiert. Beim Ausbreiten nimmt ein Pixel p_i den Farbwert seines direkten Nachbarn an $C(P_i) = C(P_i + 1)$. Das zweite Merkmal wird Wandern genannt. Hier wechselt der Farbwert von Pixel P_i zu $P_i + 1$, dabei nimmt P_i einen anderen Farbwert an $C(P_i + 1(f_1)) = C(P_i(f_0))$. Im Gegensatz zu *sequential images* detektiert *find contours* lediglich die Außenkanten - Konturen eines Objekts, dabei erzeugt das Ausbreiten die Vorderseite und Wandern die Rückseite des Objekts. In einigen Implementierungen werden Farbgruppen statt einzelnen Farbwerten genutzt, um die Funktionalität gegenüber Einwirkungen von Lichtquellen Robuster zu gestalten. Die Farbgruppierung enthält einen Referenzwert und weitere Werte ähnlicher Farben. Dadurch wird ein gewisser Toleranzbereich geschaffen, der unterschiedlich starke Beleuchtung oder beispielsweise Designfenster mit partieller Colorverglasung ausgleichen kann.

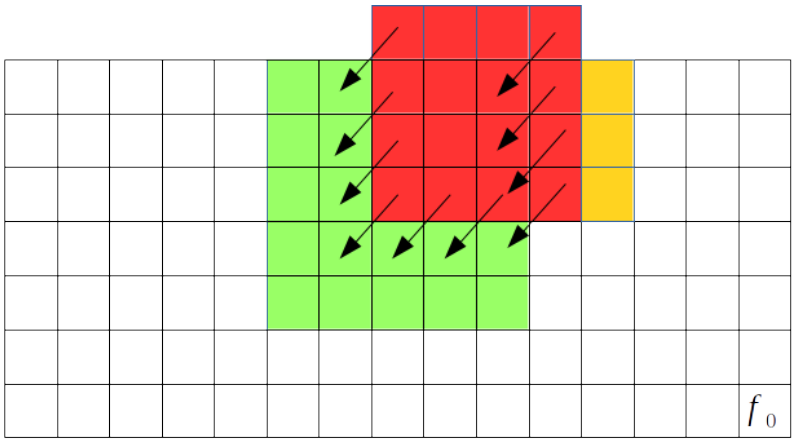


Abbildung 5.3: Visualisierung von Ausbreiten und Wandern, erster Frame f_0. Die Vektoren geben die Bewegung des quadratischen Objekts an.

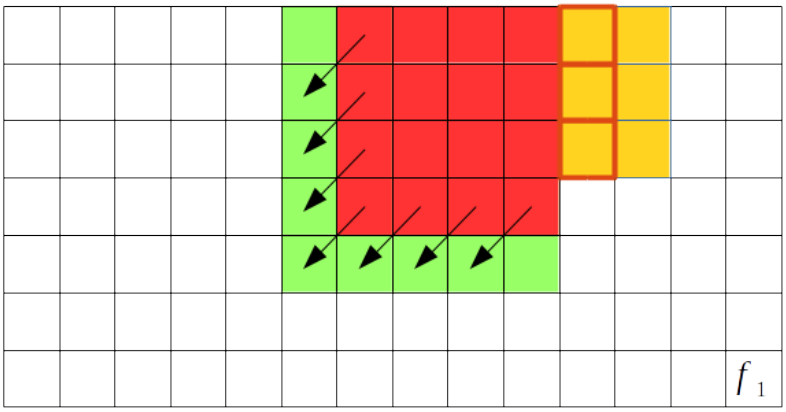


Abbildung 5.4: Visualisierung des zweiten Frames f_1. Besonderes Augenmerk sollte auf das Weiterziehen der rechten Kante des roten Quadrats und dem damit verbundenen Farbwechsel der zuvor roten Pixel, verdeutlicht durch die orange Umrandung.

5.2.4 Implementierung V1

Für die Implementierung der Objekterkennung wurde das *sequential images* Verfahren angewandt. Der Grund hierfür war die Idee, direkt die Tiefenbilder der Kinect zu verwenden, da diese schon im 13-Bit Mono Format vorliegen. Die Funktionalität ist in einen ROS node eingebettet. Das Erzeugen des Differenzbildes wird mit der *absdiff* Funktion der OpenCV-library erreicht. Dafür müssen zunächst die Bilddaten der Kinect aus dem ROS eigenen *imageConstPtr*-Format in das von OpenCV verwendete *CvImagePtr*-Format gewandelt werden. Hierfür wird die Konverter middleware cvBridge genutzt. **(ADD)Verweiß auf cvBridge in Vorarbeiten(ADD)**

```
1   void imageCallback(const sensor_msgs::ImageConstPtr& image)
2   {
3   sensor_msgs::ImageConstPtr imageInRos;
4   imageInRos = image;
5
6   cv_bridge::CvImagePtr imageInCV = cv_bridge::toCvCopy(imageInRos);
7
8   Mat matNow = imageInCV->image;
9
10  if(firstrun == 1){
11  matMinusOne = matNow;
12  firstrun = 0;
13  }
14
15  absdiff(matNow, matMinusOne, imageDiff);
16  matMinusOne = matNow;
17
18  imshow("view_diff", imageDiff);
19  }
```

imageCallback-V1

- Zeile 6: Konvertierung vom Datentyp imageConstPtr (ROS) zu CvImagePtr (OpenCV).
- Zeile 8: Kopieren der Bilddaten des aktuellen Frames in eine Mat-Speicherzelle. Vorbereitung für die Differenzenbildung, absdiff benötigt nur die Bilddaten (Mat-Datentyp). CvImagePtr enthalten noch zusätzliche Meta-Informationen über den aktuellen Frame. Mehr Details können den Datentypenbeschreibungen am Ende dieses Kapitels entnommen werden.
- Zeilen 10 bis 13: Während des Initialdurchlaufs liegt noch kein Vorgänger-Frame vor, dadurch wird in die Speicherzelle matMinusOne der aktuelle Frame geladen.
- Zeile 15: Bilden des Differenzframes mit *absdiff*. Die Funktionsattribute sind der aktuelle Frame, der Vorgängerframe sowie eine Speicherzelle des Typs Mat für die erzeugte Differenz.
- Zeile 16: Hier wird der aktuelle Frame in die Speicherzelle *matMinusOne* geladen, um bei der nächsten Iteration als Vorgängerframe zur Verfügung zu stehen.
- Zeile 18: imshow gibt den Inhalt einer übergebenen Mat-Speicherzelle (hier das Differenzbild - *imageDiff*) in einem Anzeigefenster mit dem Titel *view_diff* aus. Dies dient nur zu Debug-Zwecken um äußere Einflüsse wie die Ausleuchtung des Raumes und natürliches Licht und deren Auswirkungen auf das Differenzbild zu beobachten.

Die Beobachtung der ersten Implementierung zeigte dass das Tiefenbild der Kinect nicht für das *sequential images*-Verfahren geeignet ist. Das sehr starke Rauschen im Tiefenbild verhindert eine saubere Differenzbildung. Das Rauschen konnte auch durch Anwenden des Gauß-Filters nicht genug gemindert werden. Dies war der Anlass für eine zweite Implementierung.

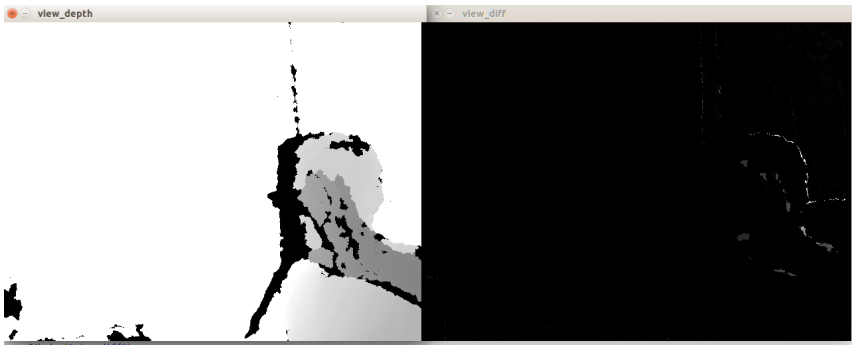


Abbildung 5.5: Darstellung des Tiefenbildes links und des Differenzbildes rechts. Durch das Rauschen im Tiefenbild wird die Differenz stark beeinträchtigt und praktisch nicht Nutzbar.

5.2.5 Implementierung V2

Die zweite Implementierung nutzt statt der Tiefenbilder das monochrome Bild des Kinect RGB-Sensors. Da RGB-Sensoren keine Bilder auf Basis eines IR-Meshes erzeugen, sondern lediglich die vorhandene Umgebungsbeleuchtung nutzen, weisen die damit erzeugten Bilder nahezu keine Interferrenzmuster durch natürliche IR-Strahlung auf. Der Differenzbildende Prozess hat sich im Vergleich zur erste Implementierung nicht verändert. Jedoch wurden Techniken zur Nachbearbeitung des Differenzbildes eingesetzt um das Ergebnis zu optimieren. Zum einen wurde die Threshold-Funktion von OpenCV verwendet welche den Kontrast des Bildes erhöht, um die Kanten sich bewegender Objekte deutlicher vom Hintergrund abzuheben. Anschließend wurde das durch die Threshold-Funktion entstandene Rauschen mit der Anwendung des Blur-Filters gemildert.

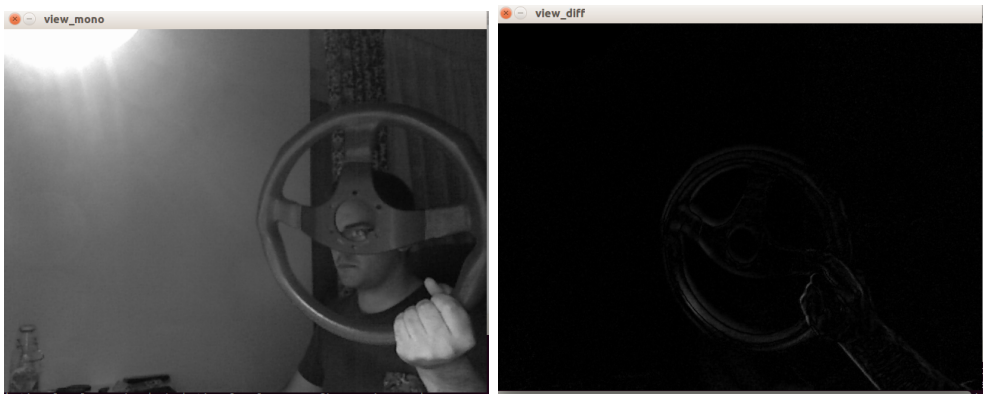
```
1   void imageCallback(const sensor_msgs::ImageConstPtr& image)
2   {
3   sensor_msgs::ImageConstPtr imageInRos;
4   imageInRos = image;
5
6   cv_bridge::CvImagePtr imageInCV = cv_bridge::toCvCopy(imageInRos);
7
8   Mat matNow = imageInCV->image;
9   if(firstrun == 1){
10  matMinus1 = matNow;
11  firstrun = 0;
12  }
13
14  absdiff(matMinus1, matNow, imageDiff);
15  matMinus1 = matNow;
16  Mat imageThresh;
17  threshold( imageDiff, imageThresh, 10, 255,0);
18  Mat imageBlur;
19  blur(imageThresh,imageBlur,cv::Size(10,10));
20
21      cv::imshow("view_mono", matNow);
22      cv::imshow("view_diff", imageDiff);
23      cv::imshow("view_thresh", imageThresh);
24      cv::imshow("view_blur", imageBlur);
25  }
```

imageCallback-V2

- Zeile 17: Anwendung der Threshold Funktion auf das erzeugte Differenzbild. Threshold ist eine Schwellwert Operation. Sie korrigiert die Farbwerte für jeden Pixel in Abhängigkeit seines aktuellen Wertes. Ist sein Farbwert unterhalb des Schwellwert wird er auf den Minimalwert reduziert. Liegt seine Intensität oberhalb des Schwellwertes wird er auf den Maximalwert korrigiert. Die ersten beiden Attribute geben die Speicherzellen für das Eingabe bzw. Ausgabe Bild an. Der dritte Übergabewert ist der Schwellwert selbst. Der Maximalwert auf den die Werte > Schwerllwert gesetzt werden sollen, ist das vierte Attribut. Zuletzt muss der Operations-Typ vorgegeben werden. Hier wurde mit 0 die *Binary*-Operation benutzt welche das oben beschriebene Verhalten triggert.
- Zeile 19: Die Blur-Funktion der OpenCV-library kann zum Filtern von Rauschen genutzt werden. Der Effekt ähnelt dem Weichzeichnen, was einen verschmierenden Effekt hat. Dies wird durch das Anwenden eines Filters auf jeden Bildpunkt erreicht. Hier wurde der *normalized box filter* genutzt. Er berechnet für jeden Pixel den Durchschnitt der Farbwerte seiner Nachbarpixel, und gleicht den eigenen Farbwert

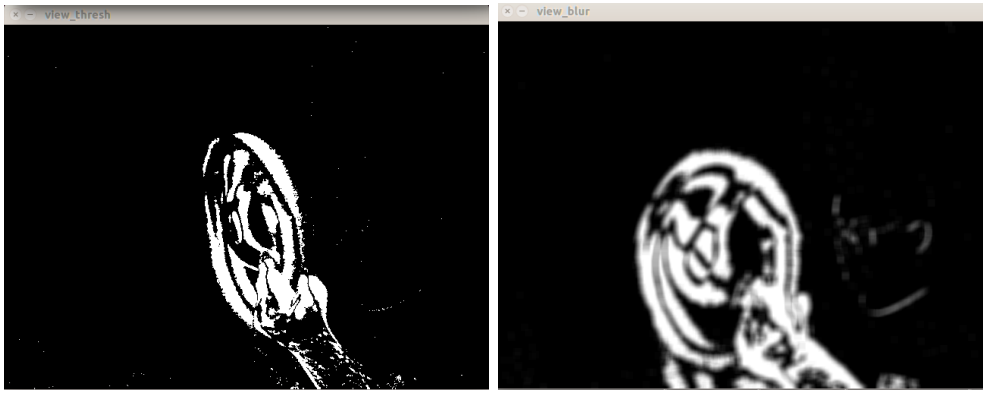
daran an. Das Attribut cv:Size() gibt den Kern(Nullvektor einer Matrix) des Filters an.

[**cv-filter**] [**unger13**] [**cv-thresh**]



(a) monochrom (b) Differenz

Abbildung 5.6: Darstellung des monochromen Ursprungsbild der Kinect, rechts davon das damit erzeugte Differenzbild.



(a) Threshold (b) Blur

Abbildung 5.7: Die Linke Abbildung zeigt das Ergebnis der Threshold-Funktion welche auf das Differenzbild angewendet wurde. Die Filterung des Threshold-Bilds mit der Blur-Funktion ist auf der rechten Seite dargestellt. Zu Beachten ist das Fehlen der Rauschanteile.

5.2.6 Unterschiede Bilddatentypen

Die folgenden Tabellen sollen die Unterschiede der beiden oben erwähnten Datentypen verdeutlichen.

Name	Inhalt	Basisdatentyp
header	Informationen über Frame	std_messages
height	Framehöhe	uint32
width	Framebreite	uint32
encoding	Bildkodierung	String
is_bigendian	least significant bit	uint8
step	Zeilenlänge in Bytes	uint32
data[]	Bildpunkte 2D-Array (step * Zeilenanzahl)	uint8

Tabelle 5.1: Tabellarische Darstellung des ROS eigenen Datentyps *ImageConstPtr*.

Der Datentyp des headers std_messages ist selbst ein synthetischer Datentyp, er kann wiederum Werte der Typen *int, String, double* enthalten.

Name	Inhalt	Basisdatentyp
source	Bildquelle	String
encoding	Bildkodierung	String
image[]	Bildpunkte als 2D-Array $P_i(X/Y)$	Mat

Tabelle 5.2: Tabellarische Darstellung des OpenCV Datentyps *CvImagePtr*.

Der Datentyp Mat ist ein Array des Typs *unsigned int 8*

5.3 Objekte Identifizieren

Die herkömmliche Herangehensweise einen Roboter mit visueller Wahrnehmung auszustatten besteht darin, direkt auf dem Chassis des Roboters eine Kamera zu verbauen. Durch die Trennung von Roboter und optischen Sensoren eröffnet sich ein zusätzliches Problem. Da der Roboter sich innerhalb des Bildbereichs bewegen soll wird er zwangsläufig selbst als mobiles Hindernis wahr genommen. Um diesen Fall zu umgehen müssen alle sich bewegenden Elemente des Bilds identifiziert werden. Die grundsätzliceh Aufgabe besteht nun darin die mobile Plattform von Objekten zu unterscheiden. Das Open-Source Projekt

Find_Object enthält die hierfür benötigten Funktionen und wurde bereits in die ROS Infrastruktur integriert.

5.3.1 Identifizieren der mobilen Plattform

Die Identifikation von Objekten mit *find-object* erfolgt durch das Registrieren der gesuchten Objekte. Die Registrierung wird über die GUI durchgeführt. Dazu wird zunächst ein Foto des Objekts angefertigt. Das Programm ermittelt nun Markante Punkte innerhalb des Bildes. Durch das Zuschneiden des Bildes auf die Außenmaße des Objekts, bekommt man ein Template des registrierten Objekts. Das eigentliche Idntifizieren von Objekten innerhalb eines Frames erfolgt durch das abgleichen seiner markanten Punkte mit denen des Objekttemplates.

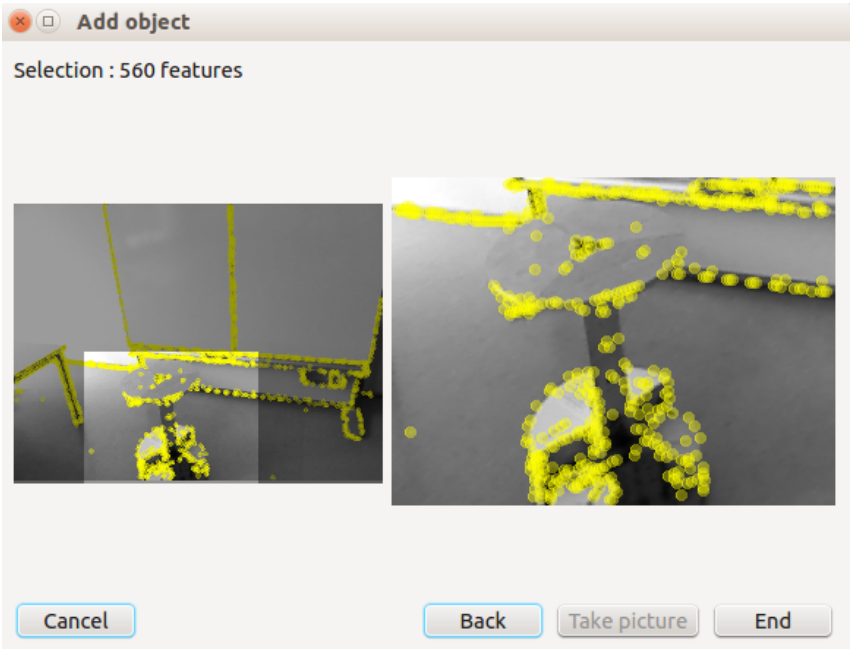


Abbildung 5.8: Darstellung der Templateerstellung mit Find-Object

Da sich die mobile Plattform innerhalb des Raumes bewegen soll, müssen zum einen perspektivische Unterschiede sowie durch Orientierungsänderungen verursachte Rotationen des Roboters abgefangen werden. Dies geschieht durch das Anfertigen mehrerer Objekttemplates.

Die Unterscheidung zwischen der mobilen Plattform und eines unbekannten Objekts erfolgt durch die Kombination der Objekt-Erkennung und der Identifikation des Robotino. *Find-Object* bietet die Möglichkeit Informationen über identifizierte Objekte in ein dediziertes

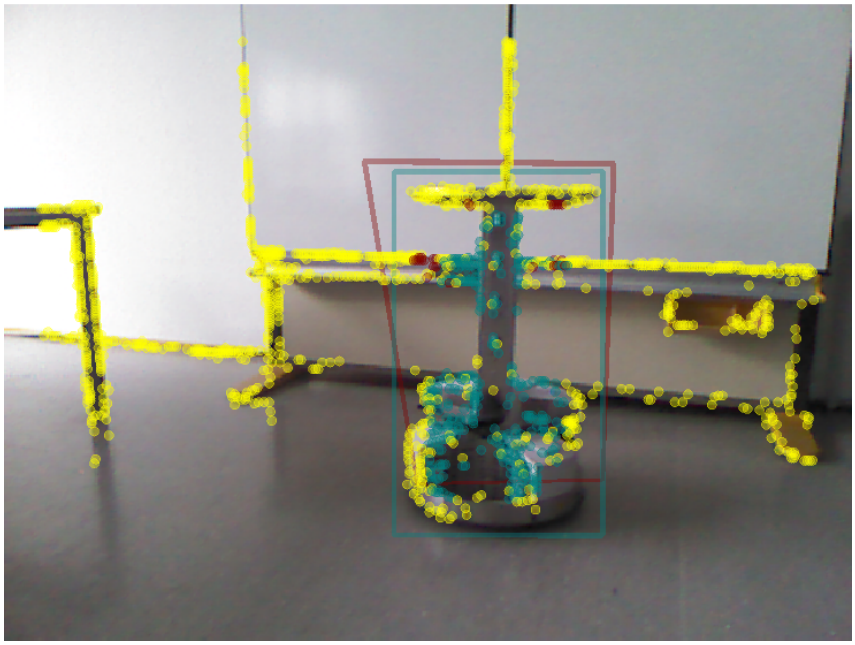


Abbildung 5.9: Hier wurde der Robotino durch 2 Templates identifiziert. Das Anwenden von mehreren Templates gleichzeitig erhöht die Fehlerresistenz.

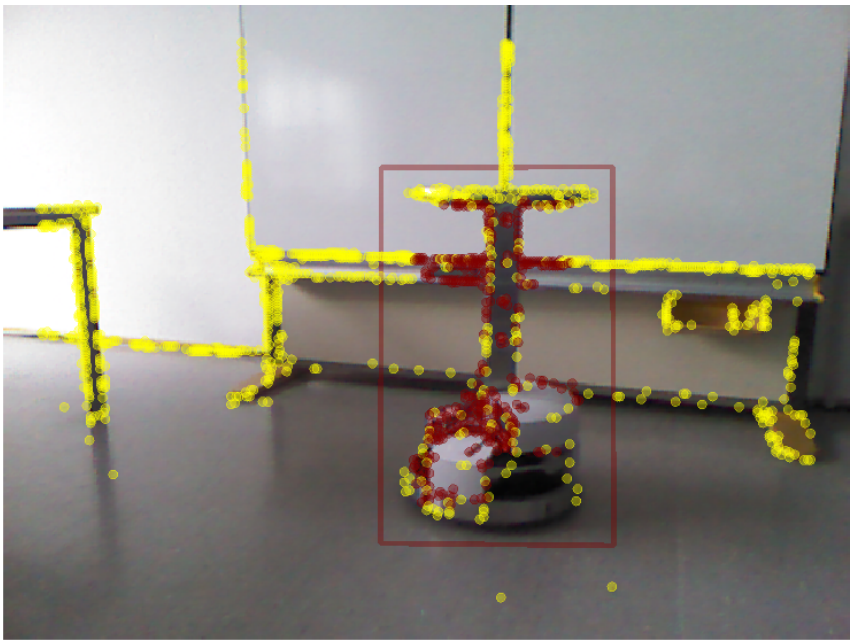


Abbildung 5.10: Hier wurde der Robotino durch 1 Template identifiziert.

ROS Topic zu schreiben. Der Informationsgehalt enthält neben der Objekt-ID die Koordinaten des Objekts innerhalb des Frames. Die Abprüfung erfolgt wiederum über einen eigenen ROS-node. Dieser bezieht sich die Koordinaten der mobilen Plattform vom *objects*-Topic, sowie das Differenzbild des *Image_Subscriber*-Knotens. Nun vergleicht er die Mittelpunkte der Objekte mit den von *Find-Object* ermittelten Koordinaten des Robotinos. Sollten diese Punkte nicht identisch sein handelt es sich nicht um die mobile Plattform sondern um ein Hindernis. Für die Mittelpunktbestimmung werden die Funktionen *find-contours* und *moments* der OpenCV-library genutzt.

floodfill algorithmus

Zuerst war der Lösungsansatz für die Objektkoordinatenbestimmung, die mit dem *Image_Subscriber*-Node detektierten Objkete mit einer eindeutigen Farbe zu füllen. Im nächsten Schritt wurden zufällig Punkte des Frames welche den definierten Farbwert aufweisen, mit den Koordinaten des Robotinos verglichen. Kommt es zu einer Übereinstimmung handelt es sich bei diesem Objekt um die mobile Plattform selbst. Kommt es zu keiner Übereinstimmung befindet sich ein Störobjekt im Bewegungsraum des Roboters.
Für das ausfüllen der Objektflächen eignet sich der Floodfill-Algorithmus. Er arbeitet von einem vorgegebenen Startpunkt aus und vergleicht die Farbwerte der Nachbarpunkte mit denen des Startpunktes. Stimmen die Farbwerte des Referenzpunktes und dessen abgeprüften Nachbar überein färbt er beide Punkte mit einer gemeinsamen Farbe ein. Das Voranschreiten im Frame ist abhängig von der Implementierungsweise. Eine Variante hierfür ist das Nutzen eines Stacks. Hierbei werden alle Koordinaten der geprüften Nachbarn in den Stapelspeicher geschrieben. Wurden alle Nachbarn eines Punktes, 4er und 8er Nachbarschaft möglich, geprüft, wird der oberste Eintrag des Stacks als Startwert der nächsten Iteration benutzt. Trifft der Fall zu dass der Farbwert eines Nachbarn nicht mit der des Referenzpunktes übereinstimmt handelt es sich um eine Kante, welche in unserem Fall die Grenze eines Objekts darstellt. In diesem Fall arbeitet der Algorithmus in dieser Richtung nicht mehr weiter, der Punkt wird wieder aus dem Stack genommen und verworfen. Der Fest definierte Startwert des Algorithmus sorgt dafür dass er für den in diesem Projekt erdachten Zweck nicht nutzbar ist. Wenn der Startwert falsch gewählt wird, werden nicht die Flächen der Objekte gefüllt, sondern die Fläche zwischen Ihnen. Dadurch würden keine Störobjekte im Raum identifiziert werden, was zu Kollisionen zwischen fahrender Plattform und Objekten führen kann.[**lode-floodfill**]

find_contours und moments

Statt der *Floodfill*-Implementierung im OpenCV Paket werden nun die ebenfalls im OpenCV-Paket enthaltenen Funktionen *find_contours* und *moments* benutzt.

```
1  void imageCallback(const sensor_msgs::ImageConstPtr& image)
2  {
3  sensor_msgs::ImageConstPtr imageInRos;
4  imageInRos = image;
5
6  cv_bridge::CvImagePtr imageInCV = cv_bridge::toCvCopy(imageInRos);
7
8  Mat matNow = imageInCV->image;
9
10
11 if(firstrun == 1){
12 matMinus1 = matNow;
13 firstrun = 0;
14 }
15
16 ///calculate absolute differential and filter noise. Finaly convert to
       threshold image for improved contrast.
17 absdiff(matMinus1, matNow, imageDiff);
18 matMinus1 = matNow;
19 Mat imageBlur;
20 blur(imageDiff,imageBlur,cv::Size(15,15));
21 Mat imageThresh;
22 threshold( imageBlur, imageThresh, 10, 255,0);
23
24 ///detect edges
25 vector<vector<Point> > contours;
26 vector<Vec4i> hierarchy;
27 findContours( imageThresh, contours, hierarchy, CV_RETR_EXTERNAL,
       CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
28
29 ///join contours to polygons
30 vector<Moments> mu(contours.size() );
31 for( int i = 0; i < contours.size(); i++ )
32 { mu[i] = moments( contours[i], true ); }
33
34 ///calculate the masscenter of each polygon, used as centralpoint of
       object
35 vector<Point2f> mc( contours.size() );
36   for( int i = 0; i < contours.size(); i++ )
37      { mc[i] = Point2f( mu[i].m10/mu[i].m00 , mu[i].m01/mu[i].m00 ); }
38
```

```
39  ///highlight the object-edges and display the masscenter as coloured
        circle
40  Mat drawing = Mat::zeros( imageBlur.size(), CV_8UC3 );
41    for( int i = 0; i< contours.size(); i++ )
42       {
43         Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255),
               rng.uniform(0,255) );
44         drawContours( drawing, contours, i, color, 2, 8, hierarchy, 0,
               Point() );
45         circle( drawing, mc[i], 4, color, -1, 8, 0 );
46       }
47
48  imshow( "view_contour", drawing );
49  imshow("view_mono", matNow);
50  imshow("view_diff", imageDiff);
51  imshow("view_thresh", imageThresh);
52  imshow("view_blur", imageBlur);
53  }
```

imageCallback-V2

- Zeilen 17, 18, 19: Das Weichzeichnen durch *blur();* wurde vor die Schwellwert-Operation gezogen. hierdurch wird ein besseres Rauschfiltern erreicht.
- Block ab Zeile 24: *findcontours* detektiert die Kanten im übergeben Threshold-Bild. Die gefunden Kanten werden in einem Vector-Array gespeichert.
- Block ab Zeile 29: Hier werden die detektierten Kanten mittles *moments* zu Polygonen zusammengefügt.
- Block ab Zeile 34: Für jedes Polygon wird nun das Massezentrum berechnet, welches als Zentralpunkt des Objekts verwendet wird.

[**cv-findcontours**]

6 Reaktion auf Hindernisse

Reaktion kommt vom alt-lateinischen *reactio* und bedeutet Rückhandlung. Genauer bedeutet es sein eigenes Verhalten an Etwas anzupassen. Lebewesen reagieren auf Umwelteinflüsse auf der Basis von Annahmen und Erfahrung. Diese Erfahrung wird durch wiederholtes Analysieren unterschiedlichster Situationen gewonnen. Die Annahmen bestehen aus dem Vergleich und der Kombination von Erfahrungen. Ein Roboter selbst, ist dazu allerdings nicht in der Lage. Ein solches Verhalten kann allerdings antrainiert werden. Dazu müssen die Merkmale zum deuten der Situation und die adäquate Verhaltensweise vorgegeben werden. Diese Informationen können durch das Modellieren einer solchen Situation erhoben werden.

6.1 Modellierung von Situationen

Eine Situation kann als geschlossenes System betrachtet werden. An einer Situation sind unterschiedliche Elemente beteiligt. Jedes Element verhält sich unterschiedlich. Jedes Element weist zudem verschiedene Verhaltensmuster auf. Ein System enthält Variablen. Jede Variable kann verschiedene Zustände annehmen. Somit lässt sich eine Abbildung von Elementen auf Variablen durchführen. Die Verhaltensmuster eines Elements sind die Zustände einer Variable. Ein Beispiel hierfür lässt sich anhand des Roboters und seines Verhaltens aufzeigen. Der Roboter selbst ist eine Variable. Der Roboter soll sich innerhalb eines Raumes auf dem Boden bewegen. Sein Zustand ist während dem er sich bewegt *moving*. Er bewegt sich jedoch nicht die ganze Zeit, sondern nur wenn er einen Auftrag bekommt, sich an eine Position zu begeben. Wenn er sich nicht bewegt wartet er auf einen Auftrag. Somit ist ein weiterer Zustand *waiting*.

Die Abbildung von Hindernissen innerhalb des Systems ist allerdings nicht ohne weiteres möglich. Dies liegt daran dass sich Störobjekte nicht die ganze Zeit im Raum befinden. Sie betreten den Raum, bewegen sich darin und verlassen Ihn eventuell wieder. Dieser Fall kann nicht als System dargestellt werden, da dessen Variablen ein fester Teil davon sind. Ein Lösungsansatz für dieses Problem wäre eine Unterteilung in verschiedene Situationen und damit auch in verschiedene Systeme. Der Wechsel zwischen den Teilsystemen kann Anhand anschließend durch Fallunterscheidungen erfolgen. Als Basis für die Fallunterscheidung

wird nun eine Klassifikation der Hindernisse vorgenommen und darin deren erwartetes Verhalten definiert.

6.2 Klassifikation

Eine Klassifikation beschreibt zunächst das zu modellierende Setting und dessen erwartete Ereignisse. Anschließend werden auf Basis der beschriebenen Szenarios Muster abgeleitet. Ebenfalls offenbaren sich Merkmale wie messbare Einheiten, welche für Systemwechsel herangezogen werden können.
In diesem Fall werden die Ereignisse welche durch das Auftauchen von Hindernissen getriggert werden beleuchtet. Ein Fall kann zu diesem Zeitpunkt bereits abgeleitet werden. Dadurch dass Hindernisse auftauchen muss es ein System geben in dem kein Hindernis im Raum existiert. Dieser Fall kann als initiales System angesehen werden. Objekte welche den Bewegungsraum des Roboters betreten sind zu diesem Zeitpunkt in Bewegung. Sie können sich auf unterschiedliche Art bewegen. Zum einen auf einem geraden Pfad. Zum anderen verfolgen sie keinen ersichtlichen Pfad, stattdessen bewegen sie sich willkürlich durch den Raum. Statt den Raum zu verlassen kann ein Objekt auch die Bewegung unterbrechen und eine fixe Position innerhalb des Raumes annehmen. Zuletzt besteht die Möglichkeit dass ein Objekt die Grenzen des Raums nur streift, und ihn nur für einen kurzen Zeitpunkt betritt und sofort wieder verlässt.
Anhand des Settings können nun folgende Klassen von Hindernissen geschaffen werden.

- **lineare Objekte** bewegen sich auf einem festen geraden Pfad durch den Raum.
- **dynamische Objekte** bewegen sich auf keinem ersichtlichen Pfad durch den Raum.
- **statische Objekte** sind entweder lineare oder dynamische Objekte welche die Bewegung abgebrochen haben und an einer festen Position innerhalb des Raumes verweilen.
- **periphere Objkete** streifen den Bewegungsraum des Roboters nur oberflächlich, und verlassen ihn sofort wieder.

6.2.1 Konzept Bewegungsanalyse

Die Klassifizierung der aufgeführten Objekte basiert auf deren Bewegungsmuster. Eine Voraussetzung für die Implementierung dieser Klassen besteht darin die Bewegung von Objekten nachvollziehen zu können. Die Bewegung selbst stellt sich als Positionsänderung zwischen zwei Frames dar. Das Problem liegt hier nun darin eine Verbindung zwischen den beiden Koordinaten über mehrere Frames hinweg zu realisieren, ohne dabei die

Objektzugehörigkeit zu verlieren. Dies ließe sich durch eine Zuteilung eines Identifikators zu jedem Objekt realisieren. Dies kann auf die selbe Weise geschehen wie die Identifikation der mobilen Plattform. Da für jedes beliebige Objekt ein Template erstellt werden müsste, mehrere sogar wenn perspektivische Verschiebungen ausgeglichen werden müssen, wäre der Aufwand zu groß um diesen Lösungsansatz praktisch umzusetzen.
Ungeachtet der Identifikation der Objekte stellt das Verfolgen der Objektmittelpunkte eine umsetzbare Lösung dar. Die Verschiebung ließe sich in Bildpunkten messen. Als Datenstruktur ist ein Vector ideal. Orientierung und Länge des Vectors überschneiden sich mit Richtung und Geschwindigkeit des Objekts. Die Richtung des Objects liese sich über die Verschiebung in der 4er oder 8er Nachbarschaft ablesen, die Geschwindigkeit über die Differenz von n-Pixel pro Frame. **(ADD)SCHEMA(ADD)**

6.2.2 Konzept Hindernis umfahren

Das Umfahren von Hindernissen ist nur bei statischen Objekten sinnvoll. Das Verhalten mobiler Objekte lässt sich nur schwer schätzen. Für die Modellierung von Ausweichrouten eignen sich Baiser Kurven. Die benötigten Start und Endpunkte der Kurve sind bereits bekannt. Hierfür können die aktuelle Position des Roboters und die Zielkoordinate des Roboters benutzt werden. Baiser Kurven werden über Hilfspunkte aufgespannt. Je mehr Hilfspunkte bekannt sind desto stärker wird die Kurve gekrümmt. Geeignete Hilfspunkte können auf einfache Art berechnet werden. Das Objekt liegt auf der Verbindungsachse zwischen mobiler Plattform und Zielpunkt. Nun kann vom Mittelpunkt des Störobjekts ein Vector aufgespannt werden der orthogonal auf die Verbindungsachse zeigt. Dieser Vector wird nun um den Durchmesser des Objekts Verlängert. Dadurch kann nun eine Kurve aufgespannt werden, die weit genug um das Hindernis herumführt, sodass der Roboter keine Gefahr läuft hängen zu bleiben. **(ADD) Schema(ADD)**

7 Interaktion mit mobiler Plattform hinsichtlich Hindernissen

7.1 Plattformwechsel

Das Ziel war es eine Infrastruktur zur autonomisierung der mobilen Plattform Robstep zu implementieren. Jedoch wurde dies durch technische Einschränkungen seitens des Robsteps verhindert. Für die Kommunikation öffnet der Robstep ein eigenes WLAN. Jedoch kann sich jeweils nur 1 Gerät mit diesem Netz verbinden. Da die Bewegung der mobilen Plattform über einen externen Rechner abgesetzt werden, besteht keine Möglichkeit eine direkte Verbindung mit dem ROS aufzubauen. Stattdessen wurde ein Plattformwechsel zu einem Robotino durchgeführt. Dieser Roboter öffnet ebenfalls ein eigenes WLAN, erlaubt aber das gleichzeitige Verbinden mehrerer Geräte. Zusätzlich wurden für den Robotino bereits Pakete in ROS bereitgestellt, was die Implementierung von Bewegungsbefehlen erübrigt.

7.2 Kommunikation mit mobiler Plattform

Die Kommunikation zwischen ROS-nodes und Robotino verläuft innerhalb des ROS-Systems. Im Robotino ist ein eigener Rechner verbaut, auf dem ein ROS-Knoten zur Ansteuerung von Sensoren und des Fahrsystems betrieben wird.

7.2.1 Integration des Robotino in ROS

Die Kommunikation mit der mobilen Plattform wird auf einem Rechner mit physikalischer Trennung zum Roboter initiiert. Dazu werden die folgenden Knoten und Dienste des ROS-Pakets benutzt.

robotino_node

Der robotino_node wird auf dem Clientsystem installiert. Er initiiert die Verbindung zum lokalen ROS-Knoten auf der mobilen Plattform. Er stellt wiederum Topics bereit über die indirekt mit dem lokalen Hardware-Knoten des Robotinos gesprochen werden kann.

robotino_local_movement_server

Der robotino_local_movement_server ist Teil des Robotino Pakets *robotino_local_movement*. Er stellt eine Verbindung zum *robotino_node* her in dem er Steuerungsbefehle empfängt und sie an das entsprechende Topic des *robotino_node* sendet.

robotino_local_move_client_node

Mit diesem node werden Steuerungsbefehle an den local-move-server gesendet. Jeder Befehl umfasst genau eine Bewegung welche aus 4 Teilen zusammengesetzt wird. Translation auf X-Achse; Translation auf Y-Achse; Rotation in Grad, Timeout des Befehls in Sekunden(Ist der Befehl innerhalb des Timeouts nicht erledigt wird er trotzdem beendet). Die Translationen sind jeweils in Metern angegeben.

7.3 Eventhandling

Der eventhandler soll als Brücke zwischen Objekterkennung und Robotino arbeiten. Die nodes *image_subscriber* und *find_object* publishen daten auf dedizierten Topics. Ein weiterer node lauscht auf beiden Topics, sobald neue Nachrichten anliegen vergleicht er die Koordinaten des Robotinos mit denen der Objektmittelpunkte. Besteht in einem Fall keine Übereinstimmung published er auf den boolschen Wert true auf das Topic /obstacle. Der eigentliche eventhandler empfängt Nachrichten des Topcis /obstacle. Sobald er die Nachricht *true* erhält, published er eine Nachricht an das Topic */cmd_vel* um den Bewegungsablauf des Robotinos zu unterbrechen.

```
1
2  bool obstacle_detected = false;
3  geometry_msgs::Twist msg;
4
5
6  ///get obstacle-state
7  void obstacleCallback(const std_msgs::Bool& obstaclemsg)
8  {
9    obstacle_detected = obstaclemsg;
10 }
11
12 int main(int argc, char **argv)
13 {
14   ros::init(argc, argv, "listener");
15
16   ros::NodeHandle n;
17
18     ///init subscriber to receive obstacle-messages
19   ros::Subscriber sub = n.subscribe("\obstacle", 10, obstacleCallback)
         ;
20
21   ///init publisher to push stop-command to Robotino
22   ros::Publisher cmd_vel_pub_ = n.advertise<geometry_msgs::Twist>("/
         cmd_vel", 1, true);
23
24 ///keep checking if obstacles are found. if true -> publish stop
       message
25 while (ros::ok())
26   {
27   if(obstacle_detected == true){
28     msg.linear.x = 0;
29     msg.linear.y = 0;
30     msg.angular.z = 0;
31     cmd_vel_pub_.publish(msg);}
32     ros::spinOnce();
33
34   }
35   return 0;
36 }
```

- Block ab Zeile 6: Topic */obstacles* nach neuen Nachrichten abfragen.
- Block ab Zeile 25: Hier wird, sofern über das Topic */obstacles* eine Nachricht mit dem boolschen Wert true vorliegt, eine Nachricht an das */cmd_vel*-Topic verschickt.

Diese Nachricht setzt sich aus einer X-Achsen und Y-Achsen Translation und einer Rotation zusammen. Alle 3 Werte sind auf 0 gesetzt. Dies überschreibt den aktuellen Bewegungsauftrag des Robotino mit den Werten 0. Dadurch ist sichergestellt dass die mobile Plattform bei detektierten Störobjekten anhält.

Um über den Seperation-Node die Koordinaten der Objektmittelpunkte mit denen des Robotinos zu vergleichen muss zunächst der Image-Subscriber node erweitert werden. Hierfür wird ein publisher eingefügt, der die berechneten Objektmittelpunkte über das Topic *(objectsXY)* verteilt.

```
1   ros::Publisher objectCenter = n.advertise<std_messages::
        Float32MultiArray>("/objectsXY", 1, true);
```

Diese Publisher Deklaration wird in die Main-Methode des image-subscriber nodes eingefügt

```
1  float msg[2] = 0.0, 0.0;
2  for(i=0, i<2, i++ ){
3  msg[i] = mc[i];
4  }
5  objectCenter.publish(msg);
```

Hierüber werden die Koordinaten des Objektzentrums in ein fließkomma-Array geschrieben und über eine 32-Bit Floatmessage gepublished. Die Konvertierung ist notwendig, da es keine direkte Message für Vector-Elemente gibt.

```
1  void objectCenter_callback(const std_msgs::Float32MultiArray&
       objCenterFloat){
2
3    for(i=0, i<2, i++){
4    objectCenterXY[i] = objCenterFloat[i];}
5  }
6
7  void robotinoCenter_callback(const std_msgs::Float32MultiArray&
       roboCenterFloat){
8
9    for(i=0, i<2, i++){
10   robotinoCenterXY[i] = roboCenterFloat[i];}
11 }
```

Durch die Callback-Funktionen werden jeweils die Koordinaten der Mittelpunkte des Robotinos sowie des Objekts in lokale float-Arrays geschrieben.

```
1    for(i=0, i<2, i++){
2    if(objectCenterXY[i] != robotinoCenterXY[i]){
3    msg = true;
4      obstacleBool.publish(msg);}}
```

Der Koordinatenabgleich wird durch eine Konditionalabfrage innerhalb einer for-Schleife durchgeführt. Wenn einer der Werte nicht mit denen des Robotino übereinstimmt wird dies über den Wechsel von false auf true in der boolschen Variable msg signalisiert. Anschließend wird der aktualisierte Wert auf das Topic */obstacle* gepublished.

8 Fazit

8.1 Projekt Fazit

- Nicht alle Ziele erreicht
- Erreichtes noch nicht 100 Prozentig sauber oder durchdacht, vieles bedarf noch Optimierung
- Allerdings sicher Grundstein für weitere aufbauende und vertiefende Projekte geschaffen.
- Trotz hoher Komplexität des Projekts sehr gutes Betätigungsfeld für Studienarbeit
- hohe Ziele spornen an, mit niedrigeren Zielen nicht den selben jetzigen Stand.

8.2 persönliches Fazit Philip Hug

8.3 persönliches Fazit Simon Simon

Das erste was mir bei der Reflektion in den Kopf kommt, ist die unterschätzte Komplexität des Projekts. Die angestrebten Ziele waren für praktische Neueinsteiger im Bereich der Robotik zu ambitioniert. Besonders der Einstieg stellte sich als sehr frustrierend heraus. Die Motivation litt stark darunter, da selbst um einfache Bilder aus der Kinect zu bekommen viel Theoretische Vor- und Einarbeitung nötig waren. Jedoch stellten sich die ersten Ergebnisse schneller ein als es zu diesem Zeitpunkt erwartet wurde. Dies wirkte sehr befriedigend und weckte die Lust weiter zu machen. Durch das breite Spektrum des Projekts besteht die Belohnung für die Arbeit darin in vielen Bereichen (Robotik, Bildverarbeitung, ROS) neue Kenntnisse erwerben zu dürfen. Zusätzlich profitierten auch Fähigkeiten in anderen Disziplinen von der Arbeit. Beispielsweise das Programmieren, durch meinen Arbeitsplatz im Consulting, fern der Softwareentwicklung, nutzte ich gerne die Chance mich in diesem Sektor zu betätigen und weiterzuentwickeln. Zuletzt bleibt noch Herrn Dipl.Ing. Michael Schneider und Herrn Enrico Hühneborg für

ihre tatkräftige Unterstützung und unkomplizierte Versorgung mit Arbeitsmaterial zu danken!