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**]

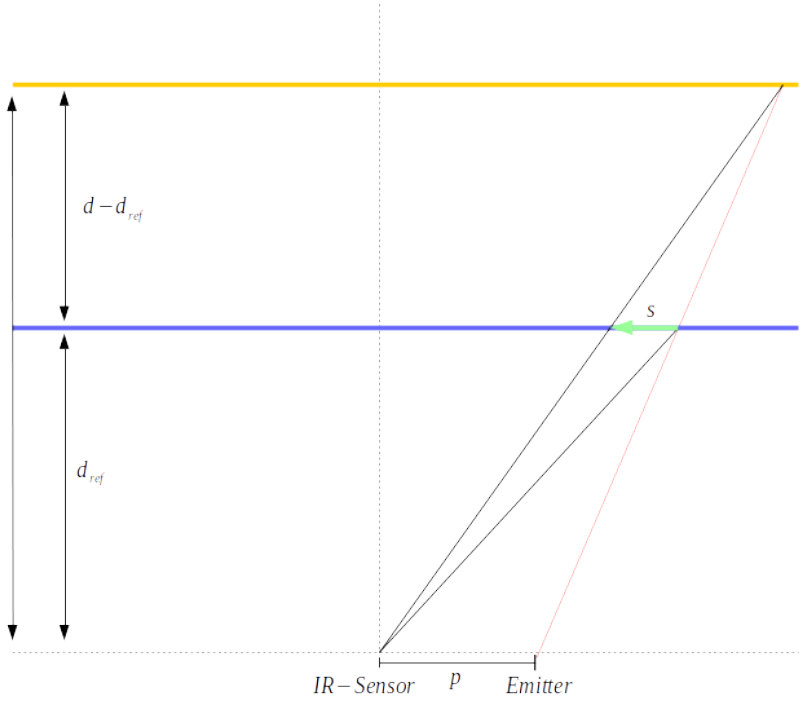


Abbildung 1.1: Illustration des light coding Prizips des Kinect Tiefenbildsystems.

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 OpenCV

1.3 Robot Operating System

1.3.1 basics

1.3.2 Topics & Nodes

1.4 Robstep

1.5 Robotino

2 Problemstellung

2.1 Vision

2.2 Erwartetes Ziel

2.3 Aufgabenverteilung

2.4 Zeitplan

3 Basis Konzept

- Grundlegende Idee -> Intelligenz aus Roboter herausnehmen in die Umgebung integrieren
- Fokus liegt hier nun auf Kommunikation der beteiligten Komponenten
- Grundmechanik auf ROS aufsetzen -> Eventbus -> liefert Basis für Kommunikation und für Eventhandling -> Erweiterungsfreundlich -> sehr gute Doku -> breite Unterstützung
- Wie soll die Kommunikation von statten gehen?
- ROS-System erklären Netzwerkkommunikation, Eventbus, Nodes Topics
- Welche Rolle spielt die Kinect
- Wie sieht die geplante Umgebung aus? -> Zeichnung

3.1 Kommunikation

- ROS nutzt netzwerktechnologien
- Medium W-LAN -> Flexibel -> Alle Komponenten einfach um WLAN erweiterbar bzw. schon damit ausgestattet
- Kommunikation innerhalb von ROS
- ROS nutzt hierfür Messages in verschiedenen einfach zu definierenden formaten

3.2 Eventhandling

3.3 Wahrnehmen der Umgebung

3.4 Feature Konzepte

Die Konzepte für die Wegfindung, die Analyse von Hindernissen und die Kommunikation mit der Plattform Robotino befinden sich in den jeweiligen Kapiteln.

4 Vorarbeiten

4.1 Betriebssystem

4.2 Installation der ROS Umgebung

4.3 Installation der Packete zur Nutzung des Microsoft Kinect Sensors

4.3.1 Treiberpakete

4.3.2 Frameworks

4.4 Bilder von Kinect bekommen

4.4.1 Ansprechen innerhalb von ROS

Topics & Nodes

zwischenspeichern der frames

image_transport? image_pipeline?

4.5 OpenCV

4.5.1 Installation

- Version 2.4.11 -> letzte Stabile V2.4 Release
- Build from Source

- cMake
- Auswahl der enthaltenen Module für Makefile
- make danach sudo make install

4.6 Robotino

4.6.1 robotino api2

- robotino api2
- eintrag in sources list
- apt-get update / install robotino-api2

4.6.2 robotino_pkg

- catkin workspace bauen
- apt-get install ros-indigo-navigation
- packet in catkin_ws/src
- robotino_node/CMakelists.txt/include_directories -> /usr/local/robotino/api2
- catkin_ws -> catkin_make
- catkin_make install
- source /catkin_ws/devel/setup.bash

5 Wegfindung

6 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.

6.1 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.

6.2 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.

6.2.1 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)

6.2.2 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.

(ADD)referenz und differenzbild(ADD)

6.2.3 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.

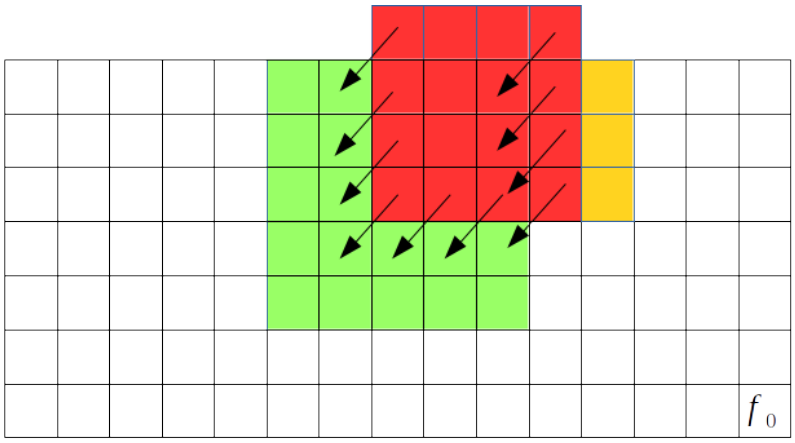


Abbildung 6.1: Visualisierung von Ausbreiten und Wandern, erster Frame f_0. Die Vektoren geben die Bewegung des quadratischen Objekts an.

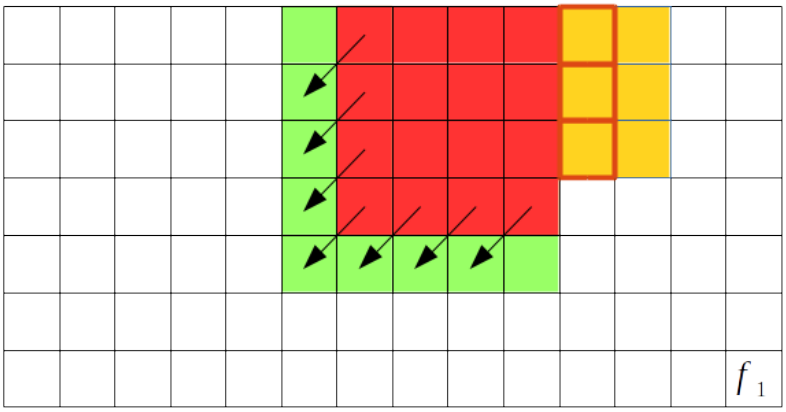


Abbildung 6.2: Visualisierung des zweiten Frames f_1. Besonderes Augenmerk sollte auf das Weiterziehen der rechten Kante des roten Quadrats und dem damit verbundenen Farbwechsel der zuvor roten Pixel, verdeutlicht durch die orange Umrandung.

6.2.4 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.

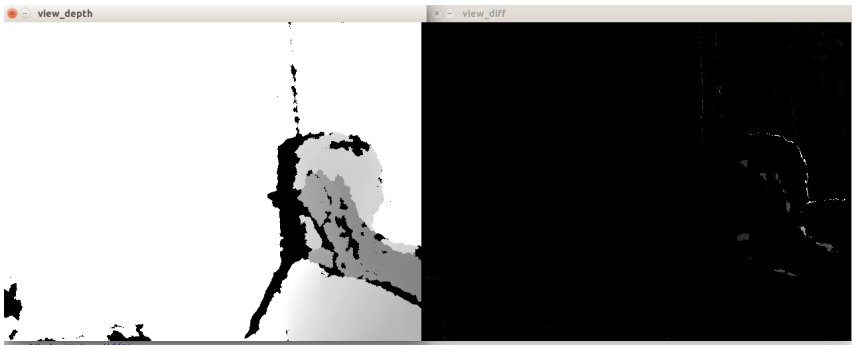


Abbildung 6.3: Darstellung des Tiefenbildes links und des Differenzbildes rechts. Durch das Rauschen im Tiefenbild wird die Differenz stark beeinträchtigt und praktisch nicht Nutzbar.

6.2.5 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.
(ADD)PAP und Codeschnipsel(ADD)

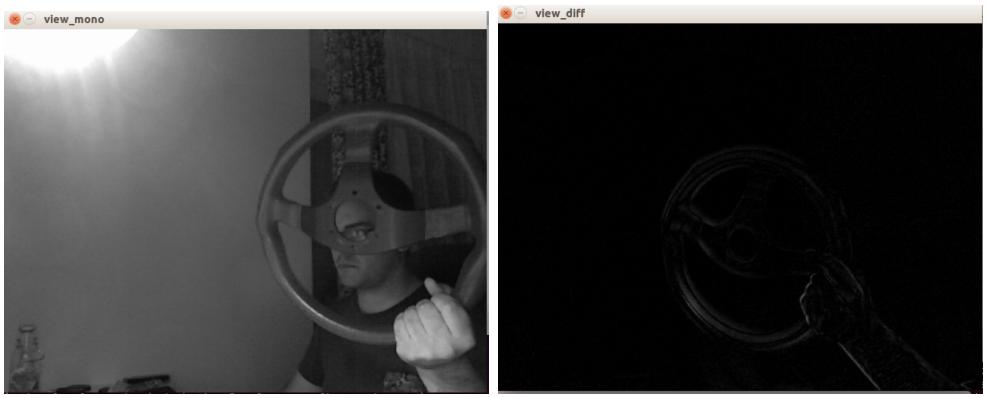
```
1   void imageCallback(const sensor_msgs::ImageConstPtr& image)
2   {
3   sensor_msgs::ImageConstPtr imageInRos;
4   imageInRos = image;
5
6   cv_bridge::CvImagePtr imageInCV = cv_bridge::toCvCopy(imageInRos);
7
8   Mat matNow = imageInCV->image;
9   if(firstrun == 1){
10  matMinus1 = matNow;
11  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

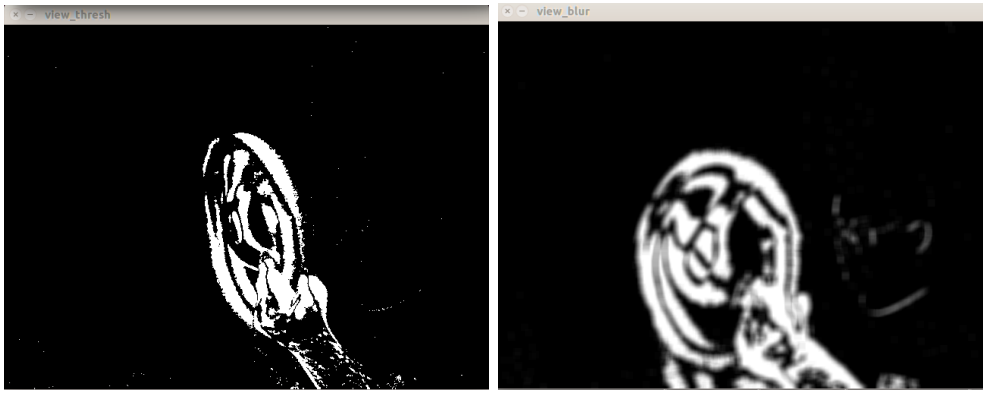
daran an. Das Attribut cv:Size() gibt den Kern(Nullvektor einer Matrix) des Filters an.

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



(a) monochrom (b) Differenz

Abbildung 6.4: Darstellung des monochromen Ursprungsbild der Kinect, rechts davon das damit erzeugte Differenzbild.



(a) Threshold (b) Blur

Abbildung 6.5: 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.

6.2.6 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 6.1: 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 6.2: Tabellarische Darstellung des OpenCV Datentyps *CvImagePtr*.

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

6.3 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.

6.3.1 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.

(ADD)Bilder von Frame und Objekt(ADD)

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 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**]

7 Reaktion auf Hindernisse

- bezug auf Klassifzierung
- Fallunterscheidung verdeutlichen
- Klassifikation anhand welcher Messwerte/Merkmale

7.0.2 Klassifikation

7.0.3 Bewegungsanalyse

7.0.4 dynamische Objekte

- Stop
- Bewegungsanalyse
- Hindernis kommt direkt auf Roboter zu -> weiterhin stehen bleiben
- Hindernis stoppt und bewegt sich nicht mehr weiter -> Umgehung bestimmen, umfahren
- Hindernis stoppt und bewegt sich weiter -> Roboter bleibt solange stehen bis Hindernis den Bewegungsraum verlassen hat oder sich nicht mehr weiter im Raum bewegt(stillstand)

7.0.5 statische Objekte

7.0.6 chaotische Objkte

7.0.7 Blitz Objekte

8 Interaktion mit mobiler Plattform

8.1 Plattformwechsel

- Robstep
- warum weg von Robstep
- warum eignet sich der Robotino besser?

8.2 Kommunikation mit mobiler Plattform

8.2.1 Integration des Robotino in ROS

8.2.2 Analyse der Topics und Nodes

- robotino_node
- robotino_local_movement
- rosrun robotino_local_move robotino_local_move_client_node x, y, Rotation in Grad, Timeout in Sekunden
- Befehl zum Unterbrechen von Befehlen und sofortiges Anhalten.

8.2.3 Konzeption der Kommunikation

nötige Funktionen

mögliche Befehle & erwartetes Verhalten

Visualisierung des Kommunikationsprozesses

8.3 Abbilden der Steuerbefehle auf Anweisungen des Eventhandlers

9 Prototyp

9.1 Testdurchführung

9.2 Evaluation

10 Fazit

Anhang

(Beispielhafter Anhang)

A. Assignment

B. List of CD Contents

C. CD

B. List of CD Contents

- **Literature/**
  - **Citavi-Project(incl pdfs)/** ⇒ *Citavi (bibliography software) project with almost all found sources relating to this report. The PDFs linked to bibliography items therein are in the sub-directory 'CitaviFiles'*
    - bibliography.bib ⇒ *Exported Bibliography file with all sources*
    - Studienarbeit.ctv4 ⇒ *Citavi Project file*
    - **CitaviCovers/** ⇒ *Images of bibliography cover pages*
    - **CitaviFiles/** ⇒ *Cited and most other found PDF resources*
  - **eBooks/**
  - **JournalArticles/**
  - **Standards/**
  - **Websites/**
- **Presentation/**
  - presentation.pptx
  - presentation.pdf
- **Report/**
  - Aufgabenstellung.pdf
  - Studienarbeit2.pdf
  - **Latex-Files/** ⇒ *editable LaTeX files and other included files for this report*
    - **ads/** ⇒ *Front- and Backmatter*
    - **content/** ⇒ *Main part*
    - **images/** ⇒ *All used images*
    - **lang/** ⇒ *Language files for LaTeX template*