Hochschule Karlsruhe – Technik und Wirtschaft

Fachbereich Mechatronik

Bachelorarbeit

Ballspielender Roboter

vorgelegt von:
Johannes Zangl
geb. am 10.08.1995 in Karlsruhe
Matrikelnummer 54604
Studiengang Mechatronik

eingereicht am 02.06.2019

Erstbetreuer: Prof. Dr.-Ing. Joachim Wietzke
Zweitbetreuer: Prof. Dipl.-Ing. Jürgen Walter
Betreuer am Arbeitsplatz: Michael Meindl

Vorwort

Als Mensch einen Ball zu erkennen ist eine einfache Aufgabe. Die Form, Farbe und Größe ermöglicht es uns, sehr rasch zu erahnen, welches Objekt ein Ball sein könnte.

Doch sobald dies eine Automatismus übernehmen soll, fangen die Probleme an. Wie soll man einem Computer erklären, was in einem Bild der Ball ist. Und was passiert, wenn der Ball anders liegt und die alte Form nicht mehr erkennbar ist? Es gibt jede Menge Probleme, die mit einer einfach klingenden Aufgabenstellung einhergehen.

Diese Bachelorarbeit beschäftigt sich mit diesen Problemen und zeigt eine mögliche Lösung auf. Dabei leisteten der Erstbetreuer Dr.-Ing. Joachim Wietzke, der Betreuer am Arbeitsplatz Michael Meindl, und der Zweitbetreuer Dipl.-Ing. Jürgen Walter und anwesende Kommilitonen viel fachliche und soziale Unterstützung, ohne die das Projekt nicht so erfolgreich gewesen wäre.

Daher sei an dieser Stelle den Personen gedankt, welche die Bachelorarbeit in jeglicher Form unterstützt haben.

Diese Ausarbeitung dient in erster Linie als Dokumentation für diejenigen, die sich im Nachfolgenden mit dem Projekt auseinander setzen müssen und wollen. Dabei wird davon ausgegangen, dass die grundlegende Ansichtsart per PDF geschieht. Daher sind Hyperlinks im Text eingebunden, sodass die Einarbeitung ohne Suche nach Links geschehen kann.

Um die Links allerdings auch in analoger Form auffinden zu können, sind alle Links mit Link-Indizes versehen. Die Links finden sich nach den Indizes sortiert im Anhang.

Eigenständigkeitserklärung

Ich versichere, dass ich die Bachelorarbeit selbstständig angefertig und keine anderen Hilfsmittel als die von mir angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt und die vorliegende Arbeit an keiner Stelle zur Erlangung eines Abschlusses vorgelegt habe.

_____ _____

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Aufgabenstellung

Ziel der Thesis ist es, dass ein bereits existierender Roboter über optische Sensorik einen Ball erkennt und diesen in ein vorher definiertes Tor schiebt oder stoßt.

Der Roboter bewegt sich dabei nach dem Segway-Prinzip fort, balanciert also auf einer Achse und bewegt sich währenddessen fort. Der bestehende Roboter kann dabei bereits balancieren und sich zuverlässig bewegen.

Im einzelnen sind folgende Punkte zu bearbeiten:

1. Evaluierung von optischen Sensoren zur Erkennung des Balles.
2. Entwicklung eines Navigationskonzeptes im Raum.
3. Entwicklung eines Bewegungskonzeptes für die Ballbeförderung.
4. Programmierung der entsprechenden Software mit Einbindung der bestehenden Software.
5. Testen des Systems.
6. Dokumentation und Aufarbeitung.

Änderung der Aufgabenstellung

Der bestehende Roboter sollte während den ersten Wochen des Bearbeitungszeitraumes von anderen Studenten fertig gestellt werden. Dabei traten allerdings unerwartete Probleme auf, welche zu einer Verzögerung des Abschlusses geführt haben. Daher war es nicht möglich, auf diesen Roboter zurück zu greifen.

Als Alternative wurde daher ein anderer bestehender Roboter benutzt, welcher allerdings sich nicht nach dem Segway-Prinzip bewegt.

Zusammenfassung

Diese Arbeit beschreibt die Evaluierung, die Konzepterstellung und die Umsetzung eines selbstfahrenden ballspielenden Roboters.

Stand der Technik

OpenCV

Das Framework OpenCV ist eine Sammlung an Methoden um Bilder zu verarbeiten. Dabei können Bilder analysiert und verarbeitet werden, um Formen, Muster und Bildinhalte zu erkennen. Das Framework bietet zahlreiche Erweiterungen und wird in vielen Projekten eingesetzt.

Verfolgung eines Balles mit OpenCV

Im Internet finden sich viele Projekte, welche sich mit der Ballerkennung und Ballverfolgung in bewegten Bildern beschäftigen. Meist werden dafür kontrastreiche Bälle wie zum Beispiel Tennisbälle benutzt, da hier die Farben deutlicher erkennbar sind.

Dabei erkennen die Programme auch Bälle, wenn sie teilweise durch Finger verdeckt sind.

In diesem Blog-Beitrag[02] wird erklärt, wie man Objekte mit OpenCV verfolgt. Die Programmiersprache ist hier allerdings Python.

Humanoide ballspielende Roboter

Es gibt regelmäßige Wettkämpfe[03], in denen humanoide Roboter bis zu einer Höhe von 50cm bis zu 150cm gegeneinander antreten.

Dabei ist im Regelwerk[04] die humanoide Form vorgeschrieben. Meistens werden in diese Roboter dann zwei Farb-Kameras eingebaut, welche über das OpenCV-Framework die Vorläufe erfassen. Die Kommunikation zwischen Robotern erfolgt laut den Regeln über Töne (Lautsprecher + Mikrofon).

Die Umsetzung hierbei ist sehr komplex, vor allem weil die Roboter miteinander spielen müssen, um Tore zu erzielen.

Viele dieser Projekte stellen ihren Quellcode öffentlich zur Verfügung[05], die Teams profitieren dabei gegenseitig von dem Wissen anderer. Durch den offenen Quellcode ist es möglich, einen stetigen Fortschritt bei der Entwicklung zu erzielen.

Objekterkennung im autonomen Fahren

Im Bereich vom autonomen Fahren werden eine Vielzahl unterschiedlicher Sensoren[06] eingesetzt. So werden Objekte auf mittlere bis große Reichweite mit Laserdistanzsensoren erkannt, während mittlere Objekte auch mit Kameras identifiziert werden. Im nahen Bereich wird dann meist auf Ultraschallsensoren oder andere Annäherungssensoren zurückgegriffen.

Durch die unterschiedlichen Sensoren ist es notwendig, dass diese über eine Sensorfusion[07] zusammengelegt werden und somit eine möglichst genaue Umgebung modelliert werden kann.

Bibliothek Intel RealSense SDK 2.0

Intel hat unter dem Namen RealSense[08] eine Bibliothek für Tiefenkameras entwickelt. In dieser werden Kommunikationsstandards festgelegt, welche für eine Vielzahl von Programmiersprachen eingesetzt werden kann, unter anderem unter C++11, C und Python.

In der Liste der unterstützten Plattformen[09] finden sich auch Linux-Distributionen wie das eingesetzte Ubuntu 18.04 LTS.

Evaluierung der Sensorik

Lidar-Sensoren (Sick TIM551)

Der getestete Sensor entspricht dem Sensor Sick TIM551[10].

Ein Lidar-Sensor sendet einen Laserimpuls aus und ermittelt durch die Lichtlaufzeit die Entfernung von Objekten. Durch einen drehenden und schwenkenden Spiegel ist es dadurch möglich eine sehr gut aufgelöste Punktewolke im Winkel von 270° zu erhalten.

Ein entscheidender Nachteil dieser Technik ist der derzeitig hohe Preis von über 3000€ pro Sensor. Der Sensor benötigt eine horizontale Befestigung und kann nur überhalb der Befestigungshöhe Daten aufzeichnen.

Bei der Inbetriebnahme ist aufgefallen, dass alle verfügbaren Treiber des Sensors für Linux-Distributionen auf ROS basieren. Sobald das System auch ohne ROS auskommen soll, wäre ein Einsatz mit erheblichen Aufwand verbunden, da ein eigener Treiber geschrieben werden müsste.

Der eingesetzte Treiber[11] heißt „sick_tim“ und ist nur für ROS erhältlich.

Durch die Funktionsweise ist es dem Sensor allerdings möglich, die Gegend auf wenige Zentimeter genau zu kartografieren. Die Orientierung im Raum ist dank 3D-Gyrosensor und eingebauten Kompass gewährleistet.

Als großer Nachteil hat sich allerdings herausgestellt, dass der Sensor unterhalb seiner horizontalen Achse keine Punkte wahrnehmen kann. Durch die Bauform wäre es also nicht möglich gewesen, den auf dem Boden liegenden Ball zu erfassen, ohne den Sensor falsch einzubauen.

Für den Test wurde die Kamera über USB an einen Computer angeschlossen. Auf diesem Computer musste ROS mit entsprechenden Zusatzprogrammen installiert werden.

Neben dem Kern von ROS musste dabei auch der Treiber für den Sensor und ein Visualisierungsprogramm für die optische Erkennung gestartet werden.

Ein Lidar-Sensor benötigt im Betrieb sehr wenig Rechnerressourcen. Allerdings würde eine Aufarbeitung der Daten für eine Erkennung eines Balles aufwendig werden, da aus der Punktewolke die Kontur eines Balles ermittelt werden könnte.

Tiefenbild-Kamera

TOF-Kameras[12], vom englischen Time of Flight, messen durch eine Laufzeitmessung eine einmalig beleuchtete Szene. Anders als bei Lidar wird die komplette Szene gleichzeitig gemessen, wodurch ein Bild einmalig vorhanden ist.

Die wohl bekannteste TOF-Kamera ist im Multimediaprodukt Microsoft Kinect erhalten. In dieser ist sowohl eine Farbkamera, als auch eine Tiefenbild-Kamera enthalten.

Bei ersten Tests wurde auf ein Nachbau zurückgegriffen, auf das Modell Asus Xtion[13], welches nicht mehr produziert wird.

Da das Modell Asus Xtion bereits vorhanden war, wurde diese Kamera benutzt, um erste Tests durchzuführen. Dafür wurde auf einem Computer die Software OpenNI[14] installiert. Mit dem OpenNI-Viewer konnte daraufhin das Tiefenbild angezeigt werden.

OpenNI fungiert hierbei als Treiber, um die Kamerabilder darzustellen.

Als Nachteil des Modells ist aufgefallen, dass die minimale erfassbare Distanz der Kamera 50cm ist. Dies ist für eine zuverlässige Ballerfassung zu wenig.

Zudem lassen sich durch die eingesetzte Software sehr wenig Daten von der Kamera erfassen.

Bei weiteren Tests wurde deshalb eine Kamera des Typs Intel SR300[15] getestet, welche zum Treiber RealSense2 kompatibel ist. RealSense2 ist eine von Intel entwickelte Schnittstelle, welche einen einheitlichen Standard für Kameras vorgibt. Diese Schnittstelle hat weitgehende Konfigurierungsmöglichkeiten, Kamerainformationen und methoden zur Bildverarbeitung.

Dadurch lässt sich zum Beispiel der konkrete Abstand eines Pixels berechnen.

Die Tiefenbilder erfassen dabei den Ball aus einem flachen Winkel sehr gut, wobei bei entsprechender Einstellung der Boden sehr gut gefiltert werden kann. Dadurch ist der Ball eindeutig erkennbar.

Als großer Nachteil ist aufgefallen, dass neuere Kameras aufgrund der Datenmenge sich nur über USB3 ansteuern lassen. Daher ist es nicht möglich, die Verarbeitung der Bilder durch einen Beaglebone Black zu bewerkstelligen.

Evaluierung

Erste Tests wurden mit der Kamera Asus Xtion2 gemacht. Diese Kamera wurde vorher bei dem Turtlebot in Kombination mit einem Lidar-Sensor eingesetzt.

Um das Verhalten der Kamera zu analysieren und der mögliche Einsatz zur Detektion eines Balles zu ermöglichen, wurde der Ball in einer Entfernung von 3 Meter platziert.

Danach wurde die Kamera auf 3 verschiedene Höhen gestellt, um eine geeignete Höhe der Montage zu ermitteln. Diese Höhe sollte als Referenz für die Montage an dem selbst balancierenden Roboter dienen. Das Kamerabild wurde über ein kleines C++-Programm an ein Bild ausgegeben.

Zuerst wurde die Kamera auf dem Boden gestellt.

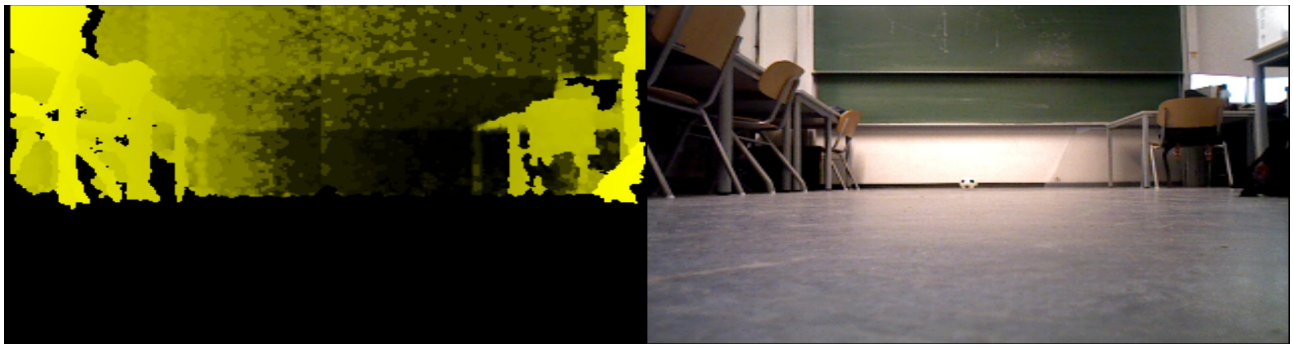


Bild 1: Evaluierung Asus Xtion, Boden

Der Ball ist auf dem Farbbild nicht gut erkennbar, weil die Wand des Raumes eine ähnliche Farbe hat. Auf dem Tiefenbild ist der Ball nicht erkennbar.

Im nächsten Schritt wurde die Kamera auf die Einbauhöhe des Turtlebots gestellt. Dies entspricht eine Höhe von 18cm zur Linse der Kamera.

Bild 2: Evaluierung Asus Xtion, 18cm

Der Ball ist im Tiefenbild klar erkennbar, der Boden ist als homogene schwarze Fläche sehr gut erfassbar. Andere Objekte im Bild wie Tisch- und Stuhlbeine sind klar erkennbar.

Im letzten Schritt wurde die Kamera mit einer Höhe von 23cm zur Linse aufgestellt.

Bild 3: Evaluierung Asus Xtion, 23cm

Zwar ist der Ball erkennbar, allerdings wird auch der Boden erkannt. Dass der Boden erkannt wird, macht die Ballerfassung sehr viel komplexer.

Daraufhin wurde getestet, wie nah der Ball sein kann, damit der Tiefenbild-Sensor das Bild erkannt. Dabei wurde ermittelt, dass der Ball minimal 50cm vom Sensor entfernt sein muss.

Störquellen

- Es wurde probiert, über Lichtquellen wie eine Infrarot-Maus, das Licht eines Kamerablitzes oder einer Fernbedienung die Tiefenbild-Kamera zu stören, da diese Infrarot nutzt. Es konnten dabei allerdings keine Störungen erkennbar sein.
- Bei Sonneneinfall fällt auf, dass das Tiefenbild einzelne Bildpunkte nicht erfassen kann. Dies kann Auswirkungen auf dem Ball haben.
- Bei einer weiteren Kamera hat sich gezeigt, dass die schwarzen Flecken des Balles dazu führen, dass keine Tiefeninformationen in diesem Bereich vorhanden sind.
- Leuchtstoffröhren können unter schlechte Umständen eine Auswirkung auf Tiefenbild-Kameras haben. Je nach Position und Stärke der Leuchtstoffröhren führen diese zu einer schlechteren Ballerkennung.

Ergebnis

Die Tests zeigten, dass durch die Tiefenbilder relativ simpel ein möglicher Ball erfasst werden kann. Zwar gibt es einige Nachteile und Störfaktoren, aber durch die Kombination aus Farbbild und Tiefenbild und die Möglichkeit der RealSense2 Schnittstelle besteht die Möglichkeit, diese Nachteile auszugleichen.

Für den Einbau auf den selbst balancierenden Roboter wurde nach einen kleinen, leichten und kompakten Sensor gesucht. Der Vergleich verschiedener Modelle hat die Entscheidung auf das Modell Intel D435i[16] gelenkt. Diese ist eine Weiterentwicklung des Modelles D415 und D435 und hat neben einer kompakten und leichten Bauform eine gute Auflösung und sehr gute Testberichte.

Farbkamera

In den getesteten Modellen der Tiefenbild-Kamera befinden sich auch Farb-Kameras. Diese erfassen Bilder im für den Menschen sichtbaren Farbspektrum.

Die Erfassung von Objekten ist ein durchaus komplexes Feld und nicht so einfach realisierbar. Einige Beispiele nutzen Tennisbälle oder Bälle für Hunde, die aus kontrastreichen Farben bestehen, damit diese einfacher gefunden werden können. Der vorliegende Ball war weiß, weswegen eine mögliche Erkennung des Balles durch das OpenCV-Framework problematisch war.

Bei Tests war es nicht möglich, den Ball vor einer weißen Ball zu erkennen. Die schwarzen Fünfecke konnten nicht als zuverlässige Referenz genutzt werden. Die Erfassung des Balles vor einem dunklen Hintergrund funktionierte mit entsprechenden Toleranzen gut, allerdings wären hier die Toleranzen für einen Feldeinsatz zu groß.

Bild 4: Farbkamera Balldetektierung hoher Kontrast

Einen kontrastreichen runden Gegenstand erkennt die Farbbild-Kamera sehr gut. Ein blauer Teller wird von dem Programm korrekt auch vor weißem Hintergrund erkannt.

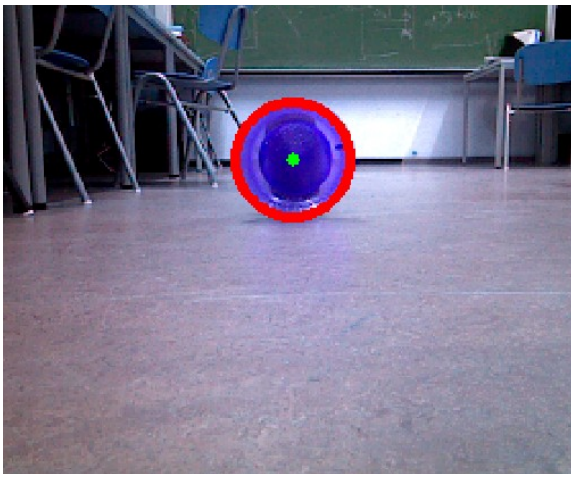


Bild 5: Farbkamera Detektion Teller

Während die getesteten Kameras durch starken Lichteinfall nicht beeinflusst worden sind, hatte zu schwache Beleuchtung einen sehr großen Einfluss.

Bild 6: Farbkamera Störung durch Licht

Eine zuverlässige Erfassung des Balles alleine mit einer Farbkamera ist daher nicht ohne weiteres möglich.

Nahbereichssensorik

Im näheren Umfeld des Roboters erwirken die evaluierten Sensoren keine oder eine schlechte Ballerkennung. Daher ist es notwendig, beim Nutzen solcher Sensoren für den näheren Bereich durch eine Sensorfusion mehrere Sensoren zu benutzen.

Für den Nahbereich wurden dafür Sensoren gesucht.

Sharp Laserdistanzsensoren

Die Laserdistanzsensoren[17] erfassen den eindimensionalen Abstand eines Objektes. Dabei wird ein Laserstrahl ausgesendet und durch einen Sensor der Einfallswinkel des Signals gemessen. Durch den Einfallswinkel kann der Abstand des Objektes ermittelt werden.

Die Sensoren sind relativ preiswert und sind in verschiedenen Ausführungen zu erwerben. Dabei gibt es unterschiedliche Distanzspektren. Für den Einsatz wurden Sensoren mit der Reichweite 4-30cm und 10-80cm getestet.

Als Einsatzgebiet sollten die Sensoren auf der Stoßstange befestigt werden. Dabei würden allerdings mehrere Sensoren nebeneinander gleichzeitig den Abstand zum Ball messen. Um zu vermeiden, dass die Sensoren sich gegenseitig beeinflussen, wurde durch einen Testaufbau gemessen, wie die Sensoren sich beeinflussen.

Für den Testaufbau wurden die Sensoren mit der entsprechenden Einbauhöhe und der Distanz zueinander auf einem Karton angebracht. Der Ball wurde daraufhin an verschiedene Punkten platziert und die Spannung gemessen. Dabei wurde der Versuch einmal mit nur einem aktiven Sensor und einmal mit zwei aktiven Sensoren ausgeführt.

Der Ball wurde vor den Sensoren in verschiedenen Positionen gelegt. Dabei sind keine größeren Unstimmigkeiten aufgetreten. Die Detektion, ob ein Ball in der Reichweite der Sensoren ist, war in jedem Fall vorhanden.

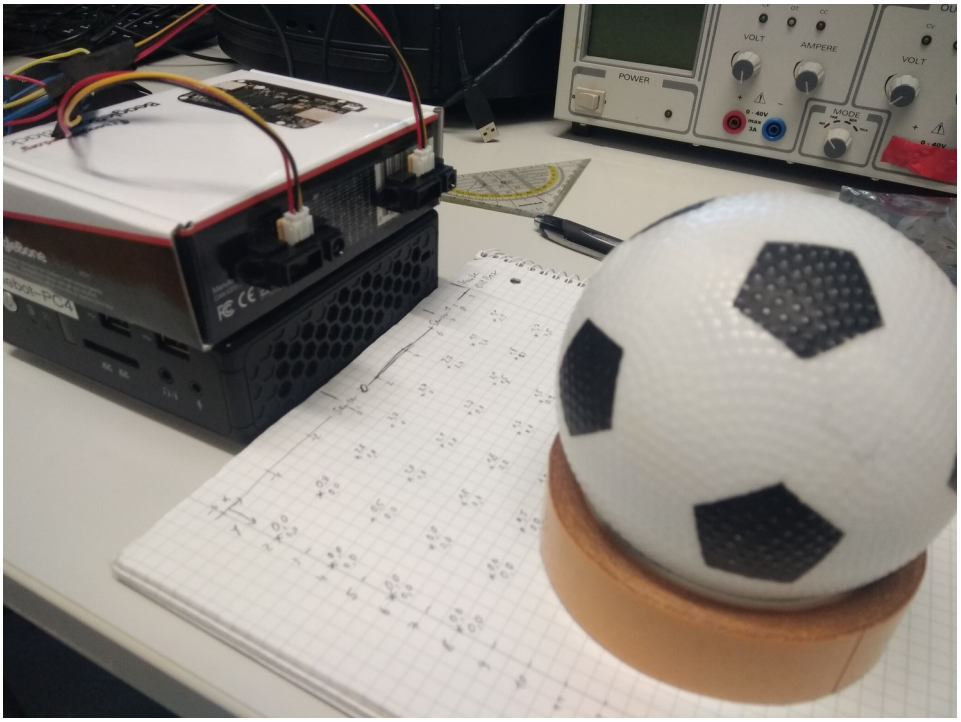


Bild 7: Analyse Sharp-Sensoren

Ultraschall

Durch Recherchen wurden auch Ultraschall-Sensoren[18] gefunden. Diese erzeugen durch ein Mikrofon Ultraschall-Wellen, welche sich über entsprechende Lautsprecher empfangen lassen. Über die Laufzeit dieser Wellen lässt sich ermitteln, wie weit das reflektierende Objekt entfernt ist.

Dabei geben diese Sensoren nicht an, wo das Objekt im Raum ist, sondern nur wie weit entfernt. Dies macht eine Ortung des Balles sehr komplex, da der Roboter durch den Raum bewegen müsste und anhand von Triangulation erfassen müsste, wo der Ball liegt. Dabei ließe sich nicht herausfinden, ob das Objekt tatsächlich der Ball ist.

Bei weiteren Recherchen sind einige unerfreuliche Eigenschaften von Ultraschall aufgefallen.

So ist es durch Alltagsgeräusche möglich, Ultraschallsensoren zu beeinflussen.

Induktive Näherungsschalter

Induktive Näherungsschalter[19] reagieren auf metallische Gegenstände. Da allerdings der Ball aus keinem Metall besteht, ist ein Einsatz eines solchen Sensors nicht sinnvoll.

Kapazitive Näherungsschalter

Kapazitive Näherungsschalter[20] reagieren auf geänderte elektrische Kapazitäten von Objekten vor dem Sensor. Dabei beträgt der Schaltabstand bis 40mm, durch die Änderung auf einen leitenden Gegenstand erhöht sich die Distanz auf 80mm. Aufgrund der geringen Distanz ist der Einsatz von kapazitiven Näherungsschaltern nicht sinnvoll.

Entscheidung für die Ballerkennung

Die evaluierten Sensoren wurden miteinander verglichen. Durch den beschränkten Platz auf dem selbst balancierenden Roboter ist dabei die Wahl auf die Nutzung folgender Systeme gefallen:

Balldetektierung

Um den Ball zu Erkennen wurde sich dafür entschieden, einen Sensor einzusetzen, der sowohl Tiefenbilder als auch Farbbilder zur Verfügung stellen kann.

Dies hat folgenden Hintergrund

- Die verfügbaren Sensoren sind sowohl leicht als auch kompakt.
- Die Sensoren lassen sich nicht so stark von einem Kippen des Roboters beeinflussen.
- Durch die Tiefenbild-Kamera ist es auch möglich, kleinere Hindernisse wie Stuhlbeine zu erkennen.
- Entsprechende Sensoren sind bereits für unter 200€ zu erhalten.

Dabei wurden allerdings folgende Nachteile in Kauf genommen:

- Der Roboter kann die Bildverarbeitung nicht auf dem existierenden Einplatinencomputer durchführen. Im Handel gibt es zwar verschiedene Einplatinencomputer mit entsprechender Rechenleistung, diese sind allerdings noch nicht erprobt.
- Im Nahbereich der Kamera ist es nicht möglich, den Ball und Hindernisse zu erkennen. Daher ist es notwendig, eine zusätzliche Nahbereichssensorik zu benutzen.

Nahbereichssensorik

Um den Ball auch dann zuverlässig zu erkennen, wenn dieser näher als der Mindestabstand der Tiefenbild-Kamera liegt, kommen Laser-Distanzsensoren von Sharp zum Einsatz.

Diese geben die Daten allerdings nur als analoge Spannungswerte aus, welche über einen AD-Wandler dem Computer mitgeteilt werden müssen. Zum Einsatz kam hier ein Beaglebone Black, auf welchem ein Python-Skript die ADC-Daten umwandelt.

Finaler Aufbau

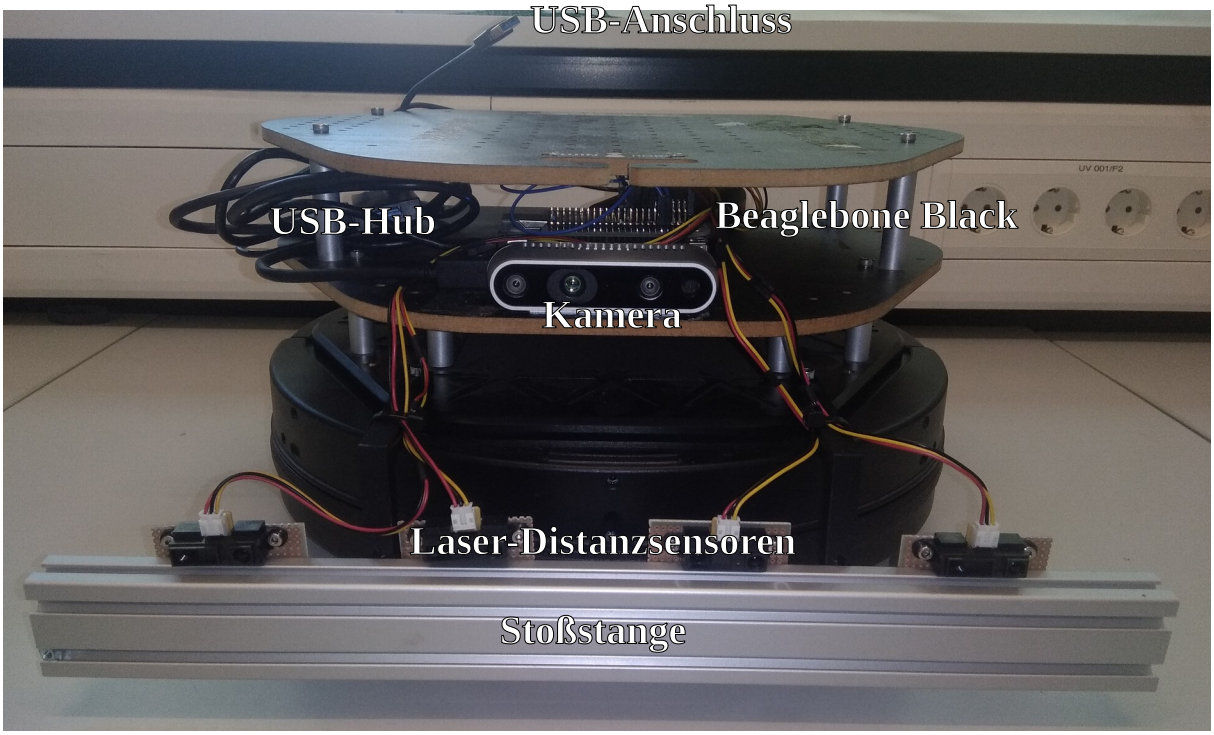


Bild 8: Turtlebot Aufbau Frontansicht

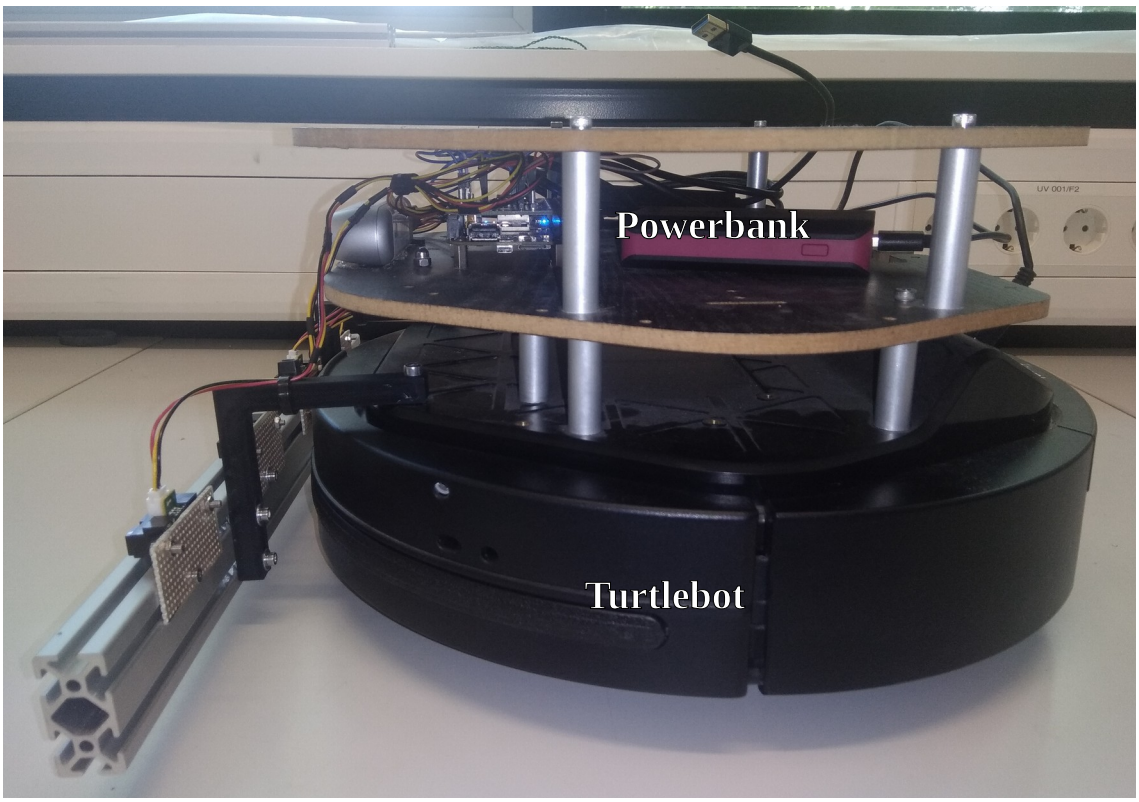


Bild 9: Turtlebot Aufbau Seitenansicht

Für den Aufbau des Roboters werden folgende Komponenten benötigt:

- Laptop oder Computer: Auf diesem ist das Hauptprogramm installiert
- USB3.0-Hub: Der Hub hat eine externe Spannungsversorgung, welche mit einer PowerBank betrieben werden kann. Dadurch wird der Laptop oder Computer entlastet.
- PowerBank: Energiequelle für alle Geräte, die an dem USB-Hub angeschlossen sind.
- BeagleBone Black: Einplatinencomputer, welcher mit AD-Wandlern die Laserdistanz-Sensoren auswerten und per Python-Skript an den Computer senden. Der BeagleBone Black ist per USB-Verbindung an den Hub angeschlossen.
- Kombinierte Tiefen- und Farbbild-Kamera Intel D435i: Dieser kombinierte Sensor erfasst das Tiefenbild und das Farbbild und schickt diese über die USB-Verbindung an den Laptop. Die Kamera ist an den USB-Hub angeschlossen.
- Turtlebot: Der Turtlebot ist über eine USB-Verbindung am USB-Hub angeschlossen.

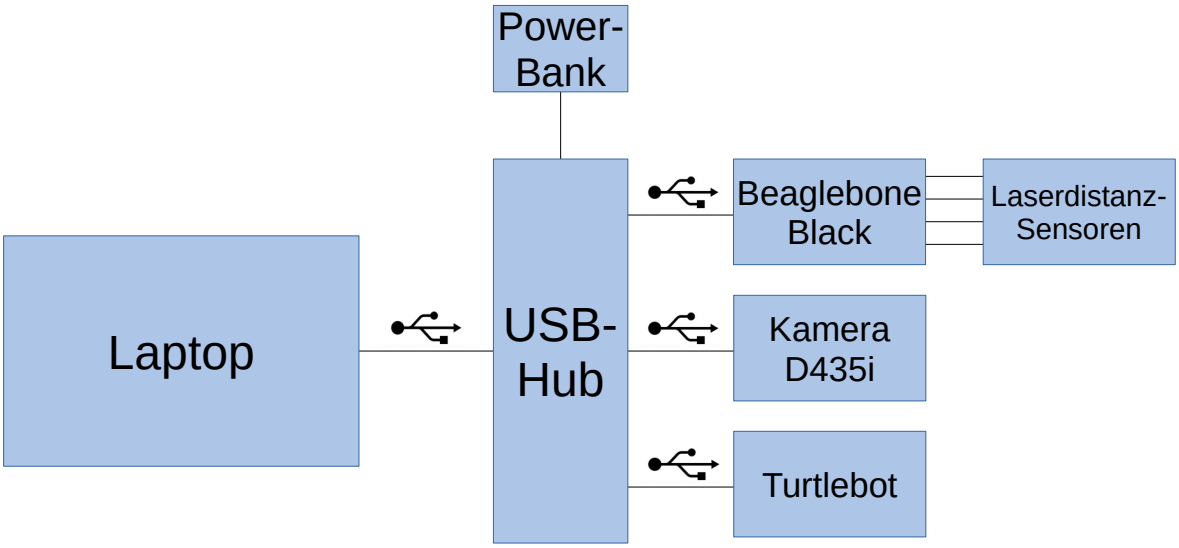


Bild 10: Turtlebot Skizze der Hardware

Turtlebot

Der Turtlebot besteht aus einer fahrenden Grundeinheit, welche durch hinzufügen mehrere Ebenen erweitert werden kann.

Der Roboter kann über eine serielle Kommunikation angesteuert werden. Dabei ist der Anschluss eine USB-Buchse. Der Turtlebot registriert sich als serielle Schnittstelle am Computer, da im Roboter ein FTDI-Chip[21] installiert ist. Dieser fungiert als Gateway zwischen UART-Signalen und dem USB-Protokoll.

Über ein seriellen Treiber lassen sich dem Turtlebot Befehle[22] senden. Diese Befehle haben eine bestimmte Syntax. Die Syntax wird dabei auf der Webseite erklärt. Zum Steuern des Roboters ist ein extra Treiber verfügbar, welcher die Installation von ROS erfordert. Durch das serielle Protokoll kann man allerdings auch seinen eigenen Treiber benutzen.

Prinzip der Ballerkennung

Mit dem menschlichen Auge ist es sehr einfach möglich, Objekte zu erkennen und diese zu unterscheiden. Den Ball kann man dabei sehr leicht durch seine Form, Größe und Höhe in dem Bild erkennen. Doch da die Software ohne menschliche Hilfe auskommen muss, ist es Notwendig, einen Algorithmus zu schreiben, der den Ball zuverlässig erkennt.

Die Tiefenbild-Kamera wandelt die Tiefenbild-Informationen in eine Punktenwolke um. Diese Punkewolke kann über Transformationen in ein Farbbild überführt werden, welches der Mensch erfassen kann. In der Software ist es notwendig, dass die Punktewolke analysiert wird und daraus Rückschlüsse gezogen werden, wo der Ball ist.

Während meiner Überlegungen zur Bildverarbeitung habe ich mir die Vorgehensweise von Menschen angesehen. Dabei stellte ich mir die Frage, wie ein Mensch die Informationen aus dem Bild verarbeitet. Objekte werden je nach Größe, Form und Farbe zusammen gefasst. Dabei ist grundlegend die Voraussetzung, dass die Objekte sich vom Hintergrund abheben.

Dieses Vorgehen kann man nutzen, um es auf die Bildverarbeitung umzusetzen:

Durch das Tiefenbild ist es möglich, Objekte aufgrund ihrer Entfernung voneinander abzugrenzen. Das Abgrenzen von Objekten ist in der visualisierten Version des Tiefenbildes durch die verschiedenen Farben möglich. Das Framework OpenCV bietet Möglichkeiten, ein bestehendes Bild aufgrund verschiedener Farben in Polygone zu zerlegen. Dabei wird jedes zusammenhängendes Objekt als eigenes Polygon in eine Liste gelegt.

Jedes Polygon besteht aus einer Liste an Punkten, welche die Kontur des Objektes festlegen.

Ein Polygon, welches ein Ball darstellt, ist allerdings nicht rund und kann nicht allein aufgrund der Form erkannt werden. Jegliche Ansätze der Erkennung des Balles durch die reine Form des Polygons ist gescheitert, da die Form nicht immer die gleiche ist. Mögliche Einflussfaktoren sind Sonneneinfall, Beleuchtung, Distanz und der relative Winkel des Balles zur Kamera.

Die Erkennung war also nicht durch die reine Form möglich, sondern musste durch die erfassten Daten aus dem Tiefenbild erfasst werden.

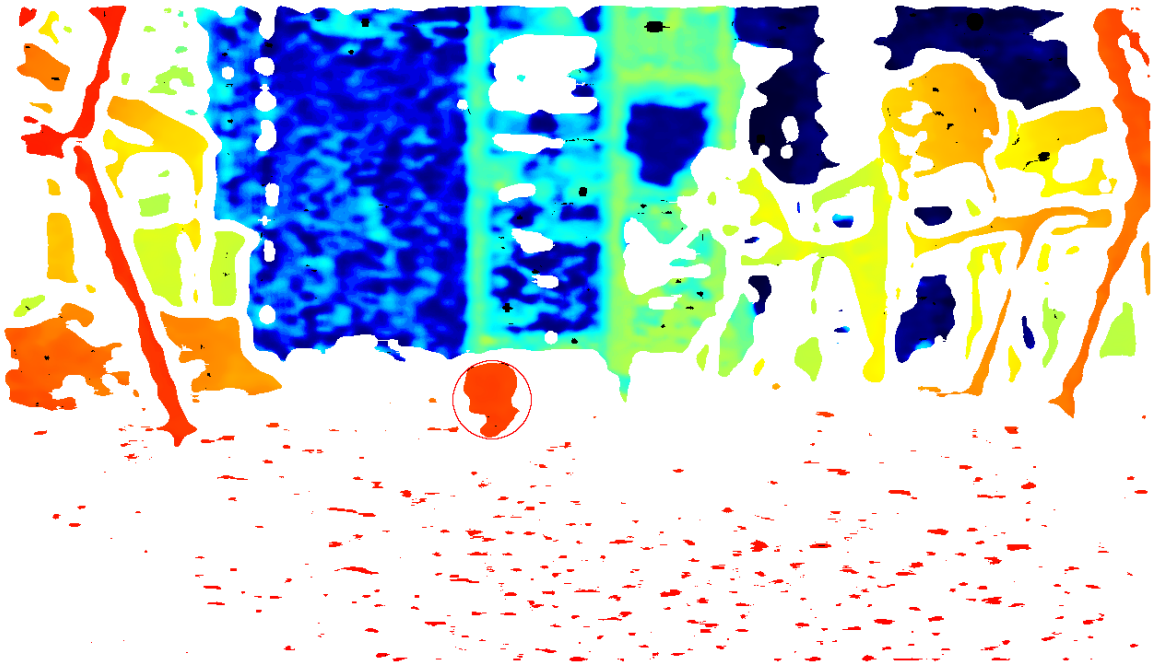


Bild 11: Balldetektion gefärbtes Tiefenbild

Wie erfasst man nun also einen Ball, den man nicht direkt als Ball erkennt?

Jedes der Polygone in der Liste hat einige Eigenschaften. Diese sind:

- Anzahl an Punkten
- Positionen der Punkte
- Abstand des Objektes zu der Kamera

Nachdem es nicht möglich ist, den Ball alleine aufgrund der Form zu bestimmen, war es notwendig, aus den Eigenschaften der Polygone Alternativen zu finden, um den Ball zu bestimmen.

Da der Durchmesser des Balles bekannt ist, ist ein Ausschlussverfahren aufgrund der Größe eines Polygons möglich. Dabei ist es möglich, jedem Polygon einen Kreis zuzuordnen, welches bei kleinst möglichen Radius alle Punkte beinhaltet. Dieser Kreis ist durch seinen Mittelpunkt und Radius bestimmt.

Durch den Abstand des Objektes ist es möglich, den tatsächlichen Durchmesser des Objektes zu bestimmen. Durch den tatsächlichen Durchmesser lässt sich ein großer Anteil der Objekte filtern.

Ein weiteres Ausschlusskriterium ist, dass die aufgespannte Fläche des Polygons eine bestimmte Fläche des kleinst möglichen Kreises auffüllt. Bei Tests ist dabei ein Wert von 60% festgelegt worden, der von dem Polygon ausgefüllt sein muss.

Einstellung der Tiefenbild-Kamera

Um die Bilder vernünftig verarbeiten zu können, wurden die Einstellung der Tiefenbild-Kamera angepasst. Dafür wird beim Start ein Profil geladen, welches für die Nutzung von Robotik-Systemen optimiert ist. Dadurch ist die Objekt- und Hinderniserkennung weniger aufwendig.

Die Einstellungen wurden von RealSense zur Verfügung gestellt und ändern die Einstellungen in Bezug auf Darstellung von den Objekten. Dabei verursacht die Kamera deutliche Abgrenzungen zwischen einzelnen Projekten und der Boden wird aufgrund der geringeren Reflektierung ausgeblendet. Da der Ball offen im Raum liegt, ist dieser klar als Objekt erkennbar.

Um die Einstellungen nutzbar zu machen, muss die eingesetzte Kamera im Advanced Mode[23] betrieben werden.

Verarbeitung der Tiefenbilder

Die eingegangenen Bilder werden im nächsten Schritt analysiert. Durch die vorangegangene Konfiguration der Kamera ist es dabei einfach möglich, jedes erkannte Tiefenbild in einzelne Polygone zu zerlegen.

Für jedes dieser Polygon wird der kleinstmögliche Kreis bestimmt, welcher das komplette Polygon beinhaltet. Durch die Methode in OpenCV wird dann ein Mittelpunkt und ein Radius in Pixel angegeben.

Mithilfe der Tiefenbild-Informationen lässt sich dann auch ein Abstand dieses Objektes ermitteln. Durch Kamera-Spezifikationen, welche sich aus der RealSense-Methoden erfassen lassen, lässt sich danach der Durchmesser dieses Polygons auf wenige Zentimeter genau ermitteln.

Der Durchmesser des vorhanden Balles wird danach verglichen. Sollte der Durchmesser zu klein oder zu groß sein, kann das behandelte Polygon kein Ball sein.

Im folgenden Schritt wird noch verglichen, ob das gegebene Polygon zu 60% des minimal möglichen Kreises ausgefüllt ist. Dadurch wird die Genauigkeit erhöht.

Verifizierung durch Farbbilder

Durch die eingebaute Farbbild-Kamera ist es in zukünftigen Schritten möglich, die ermittelte Position aus den Tiefenbilder zu verifizieren. Durch die unterschiedlichen Erfassungswinkel der Kameras ist es notwendig, die ermittelten Positionen anzupassen.

Aufbau der Software

Hauptprogramm

Das Hauptprogramm ist in C++14 geschrieben und läuft auf einen Computer, welcher sich mit dem Turtlebot bewegt.

Anforderungen

- Eingesetzter Computer:
  - USB 3.0
  - mobiler Betrieb ohne Stromanschluss
  - mindestens 2 Kerne mit je 2.4 GHz
- Eingesetzte Software:
  - Betriebssystem Ubuntu[24] 18.04 LTS oder vergleichbar
  - Installierte Build-Tools: cmake[25], build-essentials.
  - Optional: Installierte Entwicklungsumgebung CLion oder Eclipse
  - Installierte Pakete:
    - libSerial[26] für die serielle Kommunikation mit dem Roboter
    - RealSense2[08] für die Verbindung zu der Kamera
    - OpenCV[27] für die Bildverarbeitung
    - nlohmann json[28] für die Verarbeitung der Sharp-Laserdistanzsensoren

Installation

Um alle benötigten Pakete zu installieren, kann folgender Befehl in einem Terminal ausgeführt werden:

sudo apt-get install build-essential cmake cmake-qt-gui default-jdk doxygen g++ gcc git libserial-dev libusb-dev ninja-build nlohmann-json-dev

Zu den installierten Paketen muss das Framework OpenCV heruntergeladen und kompiliert werden[29]. RealSense2 muss über die bereitgestellte Anleitung[30] installiert werden.

Nachdem die Abhängigkeiten installiert worden sind, sollte sich das Projekt kompilieren lassen. Dafür wird eine Konsole gestartet, es wird in den Projektordner navigiert. Mit `cmake .` werden die benötigten Dateien zum Bauen des Projektes erzeugt. Mit `make` wird danach das Projekt gebaut.

Das Programm kann dann mit `./BaboCam` gestartet werden.

Durch die Strukturierung des Projektes ist es möglich, dieses mit unterschiedlichen Entwicklungsumgebungen zu programmieren. Die benutzte Umgebung CLion ist für Studenten kostenlos verfügbar, allerdings lässt sich das Projekt auch unter eclipse entwickeln.

Projektstrukturierung

Eingesetzte Threads

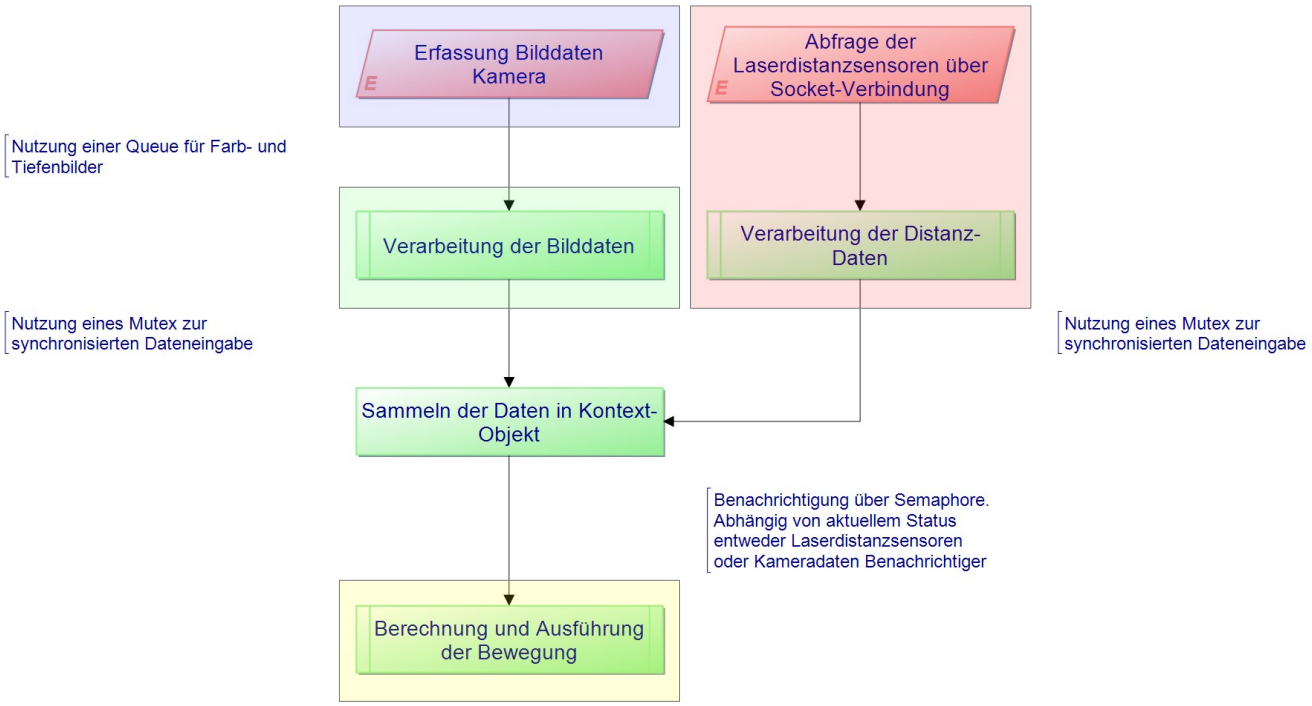


Bild 12: Threadübersicht

Das Projekt benutzt mehrere Threads, welche sich in folgende Bereiche aufteilen:

- Main-Thread: Initialisiert notwendige Objekte, startet Threads und wartet auf das Ende des Programms.
- Bilderfassung (blau): Empfangt von der Kamera Bilder und reiht diese in Queues ein, damit die Bilder dann im nächsten Schritt analysiert werden können.
- Bildverarbeitung (grün): Verarbeitet die Bilddaten, erfasst mögliche Positionen von Bällen und trägt diese Positionen in das zentrale Kontext-Objekt ein. Beim Zugriff auf die Variabeln im Kontext-Objekt wird über ein Mutex der Zugriff anderer Threads eingeschränkt.
- Laser-Distanzsensoren (rot): Öffnet eine Socket-Verbindung zum Beaglebone Black und empfängt die aktuellen Werte der Distanzsensoren. Die Berechnung möglicher Ballpositionen in näherer Distanz wird dann ebenfalls in das Kontext-Objekt eingetragen. Auch hier wird die Thread-Sicherheit über ein Mutex gesichert.
- Bewegen des Roboters (gelb): Dieser Thread wartet auf das Erscheinen neuer Daten über eine Semaphore. Je nach aktuellem Status des Roboters veranlasst entweder die Bildverarbeitung oder die Distanzsensoren den Fortschritt. Bei einem Timeout neuer Daten wird der Roboter automatisch gestoppt.

Die Erstellung dieser Threads geschieht beim Start des Programmes in der Klasse Babocam.cpp.

Datei BaboCam.cpp

In dieser Klasse liegt die Methode main, welche beim Start des Programmes aufgerufen wird. Diese initialisiert alle benötigten Ressourcen:

1. Syslog wird initialisiert. Dies ermöglicht das Loggen von Nachrichten in eine eigens angelegte Datei.
2. Die Klasse Context wird initialisiert. Diese beherbergt alle notwendigen Laufzeitdaten und Mechanismen, um Thread-Safety zu gewährleisten.
3. Die verschiedenen Threads für die Annahme von Bildern, Verarbeitung von Bildern, Annahme und Verarbeitung der Laser-Distanzsensoren und der Steuerung des Balles werden initialisiert und gestartet.

Durch das Ändern von Defines in der Klasse lassen sich einzelne Module deaktivieren. Dadurch ist es möglich, einzelne Sensoren ohne den Turtlebot zu testen.

Klasse Context

In dieser Klasse sind alle Laufzeitdaten erfasst. So werden in dieser Klasse Abstandssensoren, die Ballposition und Bewegungen erfasst.
Für die weitere Entwicklung ist angedacht, dass in dieser Klasse auch mögliche Hindernisse und weiterführende Ergebnisse der Pfadfindung gespeichert werden.

Da im Projekt mehrere Threads zum Einsatz kommen, sind die Daten mit einem Mutex abgesichert. Zudem gibt es eine Semaphore des Typs std::unique_variable (TODO), welche als Semaphore über neue Daten informiert.

Logfiles

Bei jeden Projekt ist es sinnvoll, Informationen in Dateien zu speichern. Diese Logfiles können dann auch nachdem das Programm geschlossen worden ist ausgelesen werden, um Vergleiche zu ziehen.
Für das Logging im Projekt wurde dafür der Unix-Standard syslog[31] benutzt, welcher das Loggen in unterschiedlichen Leveln ermöglicht. Dadurch lassen sich Debug-Informationen effektiv von Fehler-Nachrichten unterscheiden. Es stehen daher folgende Log-Level zur Verfügung:

- LOG_EMERG: Das System ist nicht nutzbar.
- LOG_ALERT: Ein Problem ist aufgetreten, welches unmittelbar behandelt werden muss.
- LOG_CRIT: Eine kritische Situation ist aufgetreten.
- LOG_ERR: Ein Fehler ist aufgetreten.
- LOG_WARNING: Ein ungewünschter Zustand ist aufgetreten, der sich unter Umständen zu einem Fehler entwickeln kann.
- LOG_INFO: Informationen
- LOG_DEBUG: Debug-Nachrichten.

Je nach Bedarf kann die Logdatei automatisch regelmäßig geleert werden, damit diese nicht viel Speicherplatz benötigt.

Mit dem Konsolen-Befehl

```
tail -F /var/log/babocam.log
```

kann die Logdatei in der Konsole betrachtet werden. Dabei werden neue Einträge automatisch mit angezeigt.

BuildTools

Als BuildTool für das C/C++-Projekt wurde auf cmake zurück gegriffen. Dieses BuildTool vereinfacht das Kompilieren von Projekten, weil Abhängigkeiten nicht manuell eingebunden werden müssen. Nach der Installation von Abhängigkeiten über die Paketverwaltung von Debian erkennt cmake diese automatisch und verwendet diese dann für den Linker.

Versionsverwaltung Git

Das Projekt ist über die Versionsverwaltung Git[32] organisiert. Durch die Versionierung ist es möglich, die Vergangenheit des Projektes nachzuverfolgen und dieses zentralisiert zu verwalten. So ist die Bachelorthesis auf der Webseite Github[01] zu finden.

Durch den Einsatz von Git und Github ist es nicht notwendig, das Projekt als komprimierte Datei an andere zu senden. Es besteht keine Gefahr, dass man eine veraltete Version verschickt, da alle Versionen auf Github verfügbar sind.

Durch Git ist es zudem möglich, eigene Branches des Projektes zu erstellen, um bestimmte Änderungen zu testen oder dauerhaft bestimmte Code-Teile zu ändern.

Erfassung der Bilddaten

Dies erfolgt in der Klasse CameraProcessing

Diese Klasse erstellt die Konfigurationen für die eingesetzte Kamera und startet die Bilderstreams. Dabei werden die empfangenen Bilder in Warteschlangen gelegt, welche vom Thread der Weiterverarbeitung geleert werden. Dabei gibt es zwei Warteschlangen, eine mit Tiefenbilder und eine mit Farbbildern.

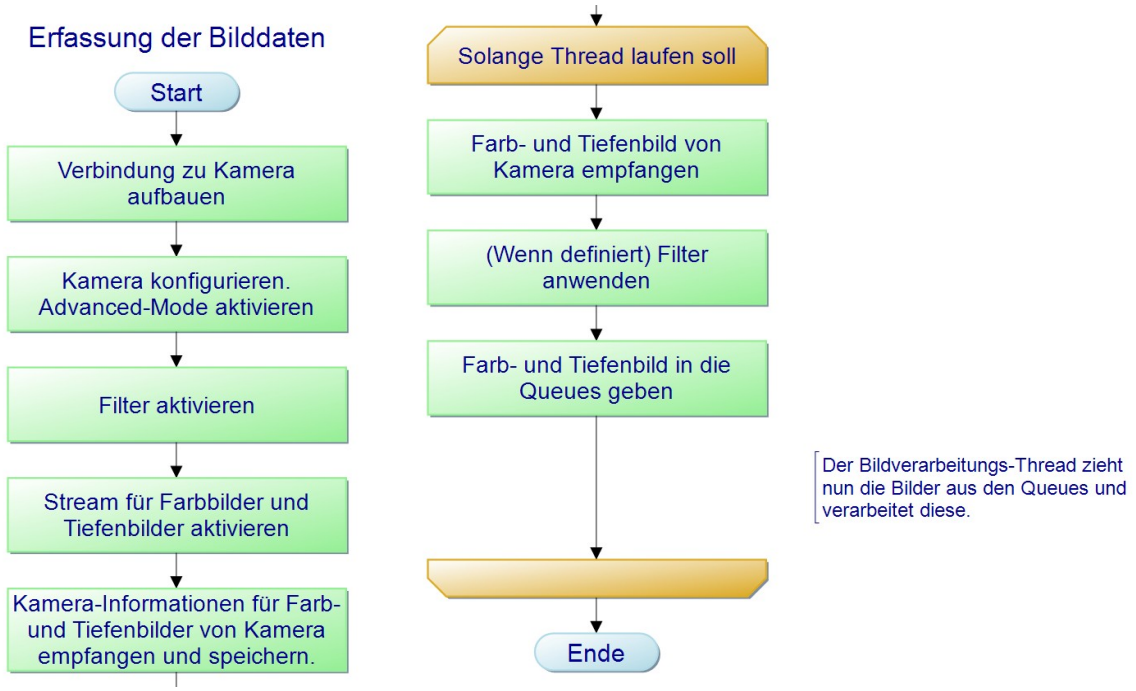


Bild 13: PAP: Erfassung der Bilddaten

Klasse BallFinder

In dieser Klasse werden empfangene Bilder von der Tiefenbild- und Farbbildkamera verarbeitet. Aus den Warteschlangen werden die entsprechenden Bilder gezogen. Danach wird das Tiefenbild in openCV importiert und in Polygone zerteilt. Mit diesen Polygonen fällt das weitere Analysieren deutlich leichter, weil jedes Objekt dank der Kameraeinstellungen eine homogene Farbe erhält. Mithilfe der Polygone lässt sich der maximale Durchmesser des Objektes und die Füllmenge bestimmen. Sobald beides den Kriterien eines Balles entspricht, wird diese Position als möglicher Ball weiter behandelt.
Solange es nur eine mögliche Position gibt, wird angenommen, dass an dieser der Ball liegt.

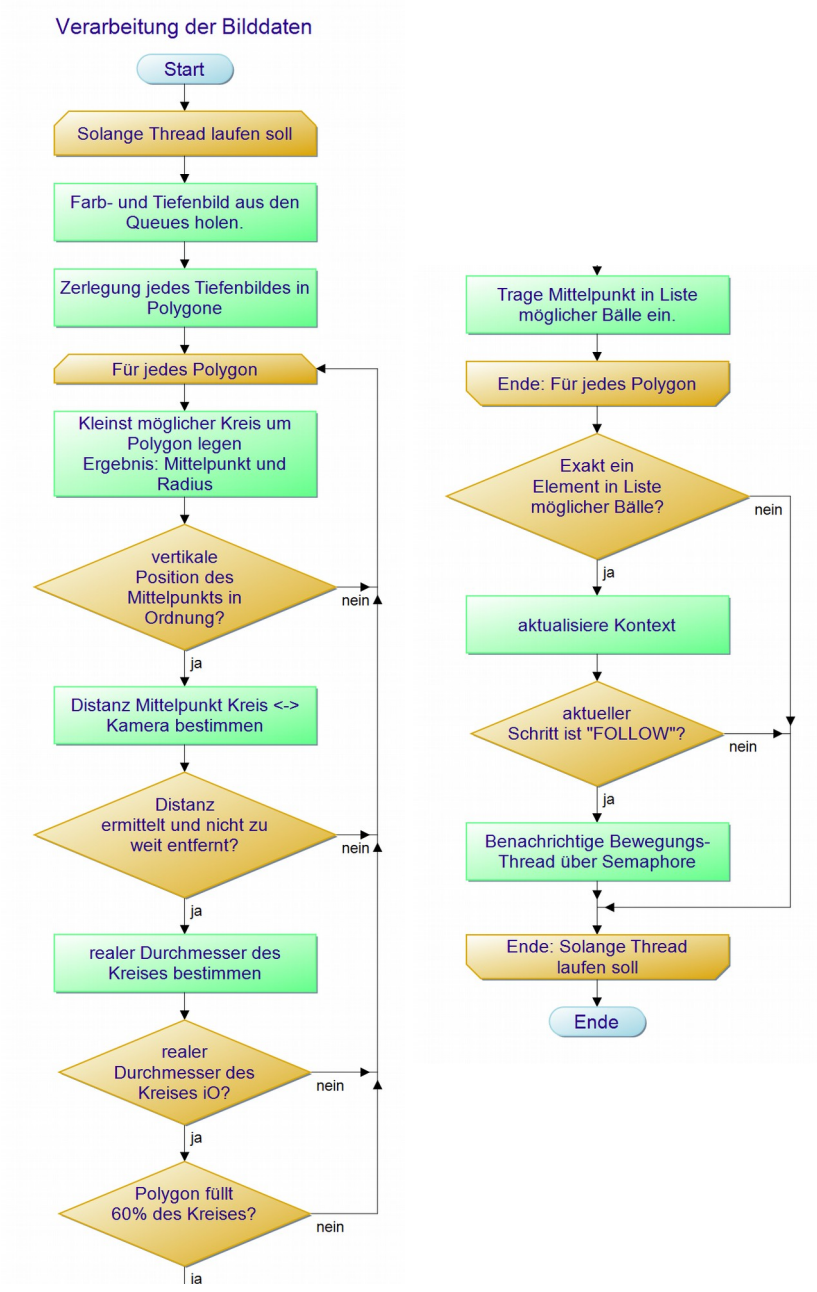
Im weiteren Verlauf wäre es dann noch möglich, die Ballposition aufgrund des Farbbilds zu verifizieren. So könnte man aus den Angaben des Tiefenbilds in unmittelbarer Umgebung des ermittelten Balles nach Charakteristiken des Balles suchen, wie zum Beispiel die vorhandenen schwarzen Fünfecke.

Verarbeitung der Bilddaten

Dies erfolgt in der Klasse BallFinder.

Nachdem die Bilder aus den Queues geholt wurden, wird das Tiefenbild in einzelne Polygone zerlegt. Jedes Polygon wird danach einzeln verarbeitet.

Wenn es exakt einen möglichen Ball gibt, wird der Kontext aktualisiert. Wenn der aktuelle Schritt der Statemachine „FOLLOW“ ist, wird die Semaphore ausgelöst.



Drawing 1: PAP: Verarbeitung der Bilddaten

Empfang und Verarbeitung der Laser-Distanzsensor-Daten

Dieser Thread baut die Socket-Verbindung zum Beaglebone Black auf und verarbeitet die empfangenen Daten.

Hierbei ist während der Ausarbeitung und der Präsentation aufgefallen, dass bei jeder Abfrage die Socket-Verbindung einmal komplett neu aufgebaut wird. Das ist unnötig und sollte bei Gelegenheit verbessert werden.

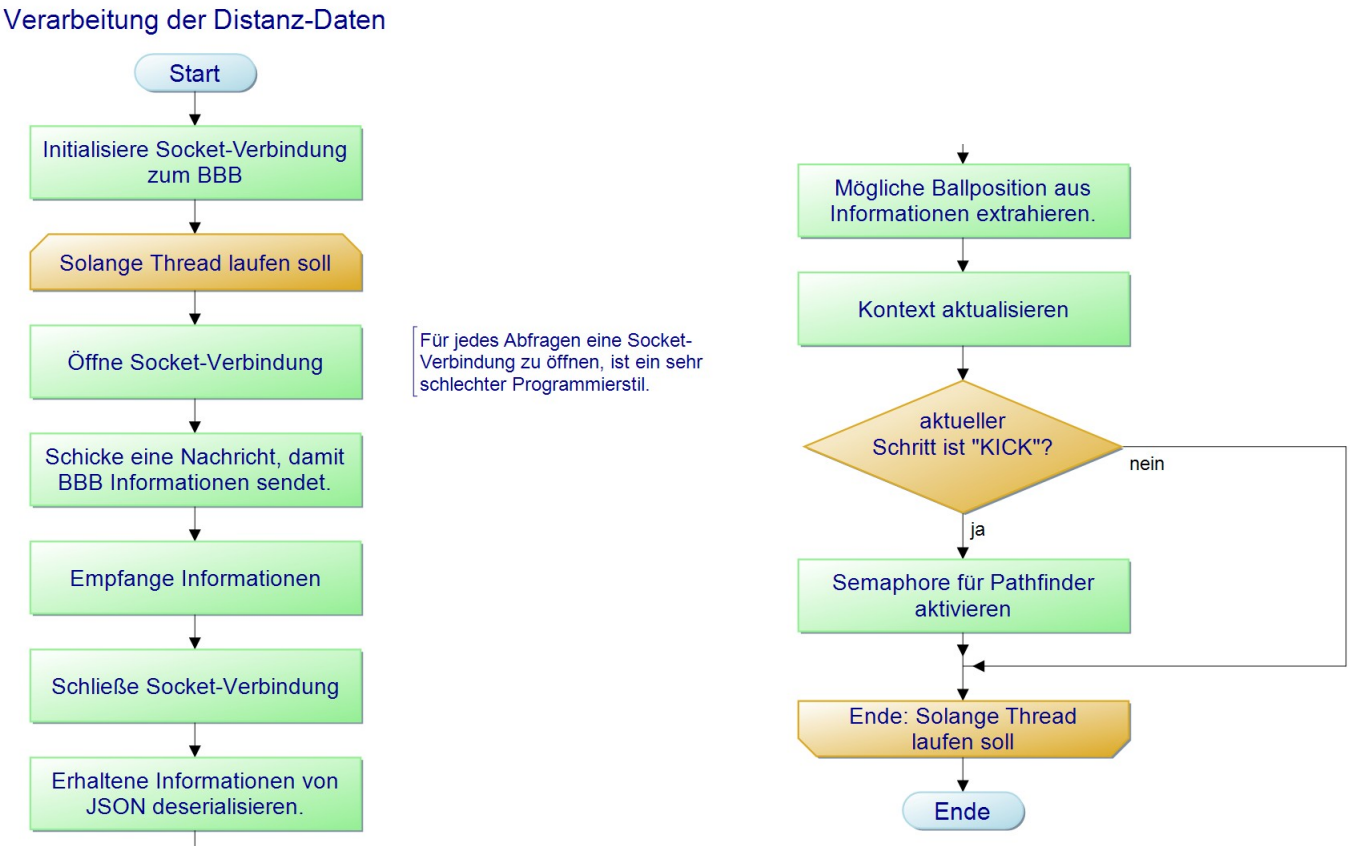


Bild 14: PAP: Verarbeitung Laser-Distanzsensoren

Klasse PathFinder

Nachdem der BallFinder den Kontext aktualisiert hat und dies durch das Setzen einer Semaphore an den PathFinder gemeldet hat, versucht der PathFinder einen Weg zu dem Ball zu finden. Dabei werden aktuell keine Hindernisse erkannt und somit umfahren, weswegen die vorhandene Logik sehr simpel ist.

Innerhalb der Methode ist eine StateMachine verbaut, welche je nach aktuellem Status den Roboter navigiert. Während der Roboter keinen Ball gefunden hat oder nach einem Stoßen des Balles wartet, bleibt der Roboter stehen. Wenn der Roboter den Ball erfasst hat, fährt er auf ihn zu. Wenn der Winkel zum Ball zu groß wird, wird der Winkel korrigiert. Sobald der Ball innerhalb der Erfassung der Sharp-Laserdistanzsensoren ist, stoßt der Roboter den Ball durch eine schnelle Vorwärtsbewegung. Nach einem Stoßen des Balles verharrt der Roboter einige Sekunden in seiner Position, bevor er erneut zum Ball fährt.

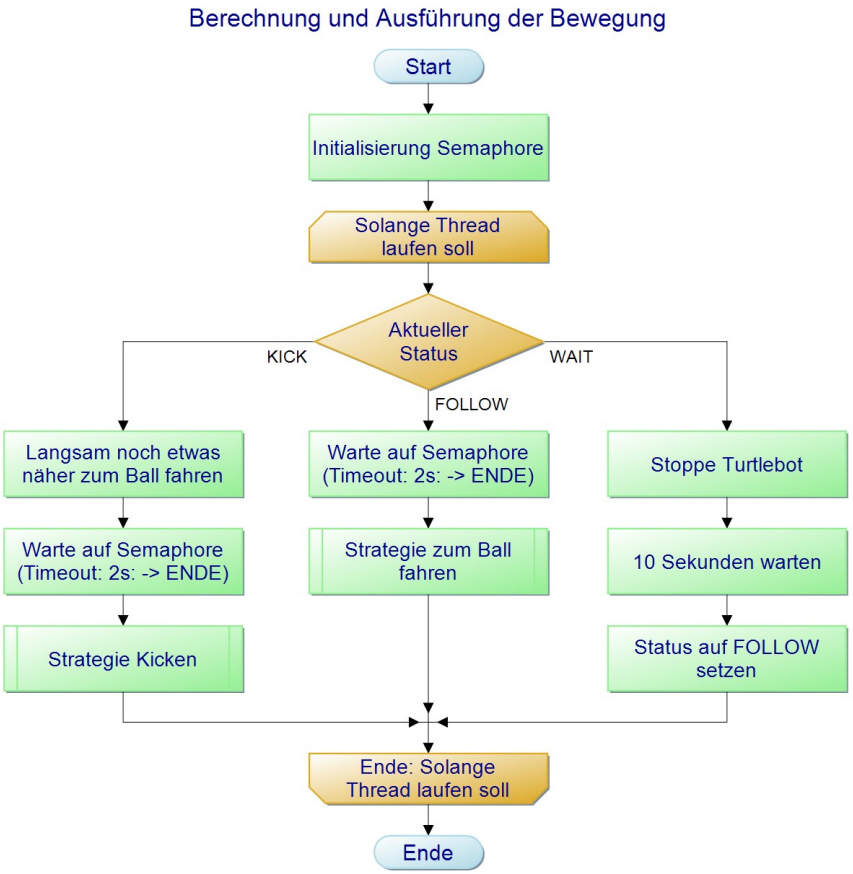


Bild 15: PAP: Klasse Pathfinder

Strategien

In einer Strategie werden die Daten aus dem Kontext verarbeitet und dadurch eine Bewegung des Turtlebots erwirkt.

Dafür gibt es die abstrakte Klasse AbstractStrategy, welche über die Methode step Möglichkeiten bietet, regelmäßig das Verhalten zu ändern. Die aktuelle Auswahl der passenden Strategie erfolgt aktuell über die StateMachine aus der Klasse PathFinder

Strategie „Zum Ball fahren“

Der Turtlebot fährt in dieser Strategie zu einem Ball. Die Position des Balles wird dabei aus den Daten der Tiefenbild-Kamera von der Klasse BallFinder ermittelt und in den aktuellen Kontext gegossen.

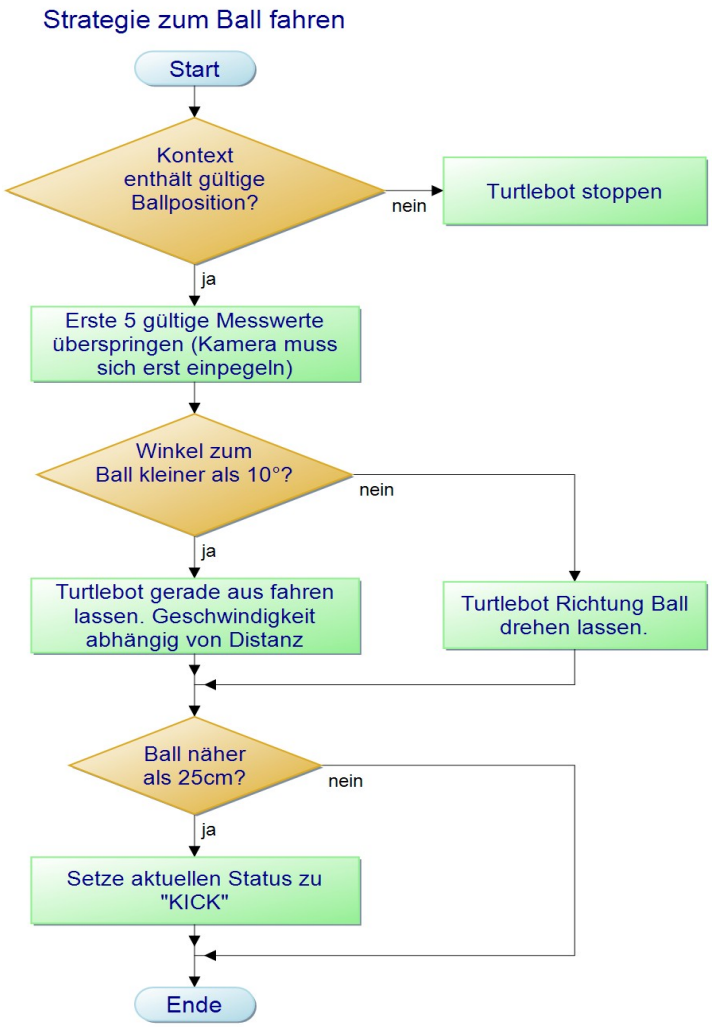


Bild 16: PAP: Strategie zum Ball fahren

Strategie „Kicken“

Die Strategie „Kicken“ kickt den Ball. Dafür werden aus den Laserdistanz-Sensoren die Werte validiert und bei Erfassen eines Balles wird dieser durch eine feste Abfolge von Geschwindigkeiten und Zeiten gekickt. Die Zeiten und Geschwindigkeiten wurden experimentell ermittelt.

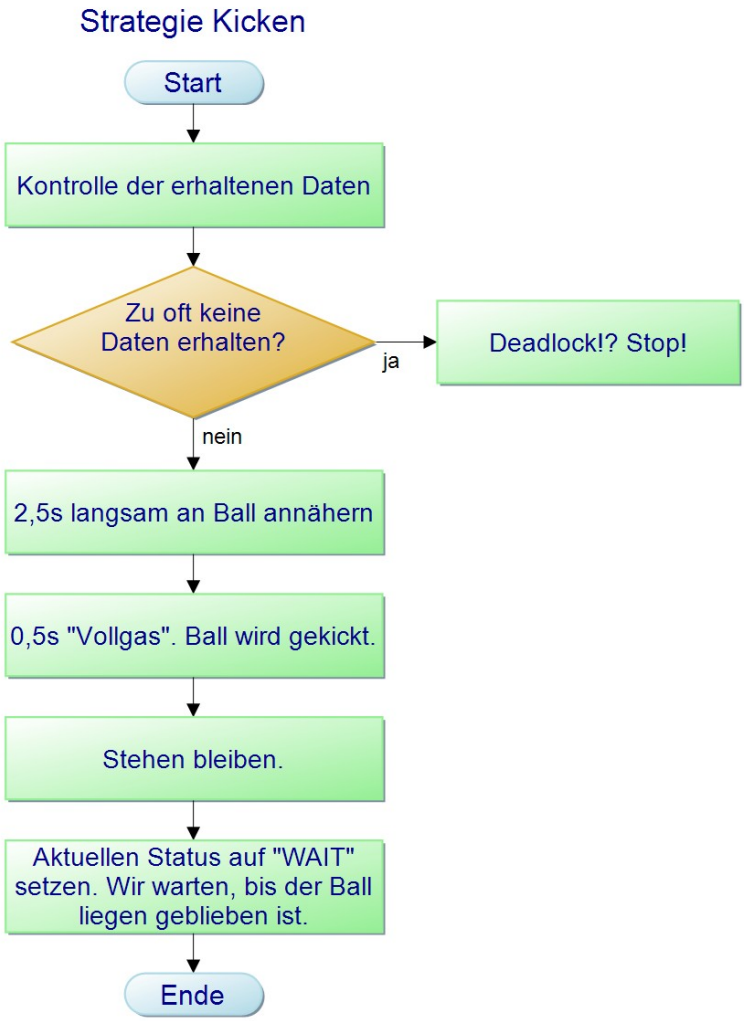


Bild 17: PAP: Strategie Kicken

Kobuki Treiber

Bei Recherchen zu der Ansteuerung des Turtlebots ist aufgefallen, dass alle verfügbaren Treiber auf ROS basieren. Leider wurde kein Turtlebot-Treiber gefunden, welcher ohne weiteres kompilierbar ist. Es sind vermehrt Probleme aufgetreten, darum wurde entschieden, den vorhanden Kobuki-Treiber anzupassen.
Für die Anpassung wurde die Bibliothek libSerial benutzt, welche die serielle Kommunikation über USB handhabt. Dabei wurden alle vorhandenen Verweise auf ROS-Pakete entfernt und unbekannte Klassen durch eigene oder bereits vorhandene ersetzt.
Dabei wurde die Logik zum Empfangen der Pakete angepasst, da die Umsetzung über libSerial in diesem Fall weniger Code benötigt.

Während der Arbeiten ist aufgefallen, dass regelmäßig ungültige Pakete empfangen werden. Dabei fangen anscheinend mitten in einer Nachricht neue Pakete an. Dieses Problem lässt sich darauf zurückführen, dass bei einem vollen seriellen Buffer dieser überschrieben wird. Dadurch wird dann der Buffer kompromittiert. Als Lösung dieses Problems wird bei fehlerhaften Paketen gewartet, bis der eindeutig identifizierbare Anfang einer Nachricht erkannt wird. Die verlorenen Pakete werden aktuell ignoriert, da diese zum erfolgreichen Betrieb nicht zwingend notwendig sind.
Durch die Überarbeitung des Treibers wurden die Callbacks bei empfangenen Pakete verändert. Während man zuvor für Nachrichten mehrere Callbacks registrieren kann, ist es nun nur noch möglich, einen einzelnen Callback zu registrieren.
Diese Entscheidung wurde getroffen, weil die empfangenen Daten in dieser Art und Weise nicht von mehreren Empfängern verarbeitet werden müssen.

Der Treiber schickt regelmäßig Nachrichten zum Turtlebot und der Turtlebot schickt regelmäßig Pakete zurück. Diese können Antworten auf gesendete Pakete sein. Eine Übersicht aller Pakete findet sich hier.

Klasse SharpSensors

Die Daten der Laserdistanz-Sensoren werden über ein Python-Skript als Socket zur Verfügung gestellt. Sobald ein Client sich zu diesem Server verbindet und ein beliebigen Text sendet, wird im JSON-Format die Spannungen der Sensoren zurückgesendet.

In der Klasse SharpSensors wird in regelmäßigen Zeitabständen ein Socket initialisiert und per USB-Verbindung zum Beaglebone Black abgefragt.
Durch Übertragungsfehler kommt es dabei vor, dass die übertragene Zeichenkette ungültige Zeichen am Ende enthält. Um diese zu entfernen werden überschüssige Zeichen hinter dem JSON-Block abgeschnitten.

Durch die Bibliothek JSON von nlohmann wird dann die empfangene Nachricht deserialisiert, die einzelnen Werte verarbeitet und im Kontext zur Verfügung gestellt.

Abstandssensoren

Die Abstandssensoren von Sharp geben den ermittelten Abstand mit einer analogen Spannung zwischen 0 und 3,2V an. Diese Spannung muss über einen AD-Wandler umgewandelt werden, damit das Hauptprogramm auf diesen zugreifen kann.

Hardware

Für diesen Zweck wurde ein BeagleBone Black eingesetzt. Dieser Einplatinencomputer hat mehrere AD-Wandler, welche zwischen 0 und 1,8 V die Spannung ermitteln können.

Da der Spannungsbereich der AD-Wandler nicht dem der Sensoren entspricht, musste dafür ein Spannungsteiler[33] eingesetzt werden. Dieser Spannungsteiler skaliert die Spannung des Sensors auf den Messbereich und schützt den ADC-Eingang. Die Spannungsteiler befinden sich auf der Erweiterungsplatine des BeagleBone Blacks.

Software

Als Software kommt ein simples Python[34]-Skript zum Einsatz. Dieses nutzt die Python-Erweiterung Adafruit_BBIO, welche als Treiber für die Hardware-Funktionen von dem Einplatinencomputer fungiert.

Das Skript besteht aus einem einfachen Websocket. Dieser Socketserver wartet auf eingehende Anfragen und antwortet mit den gemessenen Spannungen. Die Spannungen werden per JSON serialisiert an den Anfragenden gesendet.

Betrieb

Um die Software zu starten, ist es notwendig, sich auf den Beaglebone Black zu verbinden. Nach dem Start wird dazu eine SSH-Verbindung[35] zum Einplatinencomputer aufgebaut:

```
ssh root@192.168.7.2
```

Das benötigte Passwort lautet „icm“. Daraufhin wird das Python-Skript gestartet:

```
python /home/debian/dev/sharp/BaboCamSharpSensors.py
```

Programmierhilfen

Beim Arbeiten mit der RealSense2-Bibliothek, dem Kobuki-Treiber und dem OpenCV-Framework sind einige Probleme und Fragen aufgetreten, in die einige Zeit investiert werden musste.

Der folgende Abschnitt soll helfen, auftretende Fragen zu beantworten.

Programmieren mit generischen Klassen

Einige Klassen in den RealSense2-Treibern und den Kobuki-Treiber nutzen generische Schablonen[36]. Diese Schablonen geben an, dass eine Klasse eine zugehörige Klasse hat, die vom Programmierer erst dann deklariert wird, wenn er diese Klasse benutzt. So hat für gewöhnlich eine Liste eine bestimmte Klasse, die aber bei der Implementierung der Liste selbst nicht relevant ist.

Umwandeln von Bildern in das OpenCV-Framework

Die Bilder, die ausgelesen werden, werden von RealSense2 in eine Queue gelegt. Diese Bilder werden dann in das OpenCV-Framework transferiert. Diese Transformation ist leider nicht einfach so möglich, sondern hat einiges an Recherche erfordert.

```
depth_frame depth_frame = depth_queue.wait_for_frame().as<rs2::depth_frame>();
const int w = depth_frame.get_width();
const int h = depth_frame.get_height();
frame colorized_depth = color_map.process(depth_frame);
Mat mat = Mat(Size(w, h), CV_8UC3, (void*)colorized_depth.get_data(), Mat::AUTO_STEP);
```

In diesem Beispiel wird zuerst das Tiefenbild aus der Queue geholt. Durch eine Umformung in ein Tiefenbild (Typ „rs2::depth_frame“) ist im weiteren Verlauf die Möglichkeit vorhanden, auf Informationen wie die Distanz eines Punktes zurück zu greifen.

Im Folgenden wird die Höhe und Breite des Bildes ermittelt und das Bild eingefärbt.

Als letzter Schritt wird das Bild in den Typ von OpenCV umgewandelt. Dabei muss die Größe und die Daten angegeben werden. Der Datentyp ist im Fall des Tiefenbildes CV_8UC3, kann sich allerdings je nach Verarbeitung und je nach Kamera ändern.

Fehler bei Verbindung zu der Tiefenbild-Kamera

Dieses Problem kann auftreten, wenn die Kamera in einem USB2.0-Slot gesteckt wurde. Zudem sollte der USB-Hub über die PowerBank versorgt werden, da der Gesamtstrom von Kamera, Turtlebot und BeagleBone Black für einige USB-Treiber zu groß ist.

Sollten trotzdem Probleme auftreten, ist eine kurzfristige Trennung der USB-Verbindung und ein Neustart des Programmes eine gute Lösung.

Keine Verbindung zum BeagleboneBlack

Hier sollte man über die Konsole schauen, ob der BeagleboneBlack sein Netzwerk korrekt aufgebaut hat. Unter ungünstigen Umständen kann hierbei der Verbindungsaufbau Probleme bereiten. Ein Trennen und wieder Anstecken der USB-Verbindung mit Neustart des Gerätes behebt das Problem.

Anhang

Literaturhinweise

- Rosebrock, A.: „Ball Tracking with OpenCV“, in: pyimagesearch (14.09.2015) unter: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/ (abgerufen am 18.05.2019)
- Lievendag, N.: „Sensor shootout: Realsense D415 vs. SR300 vs. Orbbec Astra S“, in 3dscanexpert unter: https://3dscanexpert.com/3d-scan-sensor-shootout-realsense-d415-vs-sr300-vs-orbbec-astra-s/ (abgerufen am 22.05.2019)
- BDTi: „Evaluating Intel‘s RealSense SDK 2.0 for 3D Computer Vision“, in bdti unter https://www.bdti.com/MyBDTI/pubs/Evaluating-Intel-RealSense-SDK-2.0.pdf (abgerufen am 06.04.2019)
- Agam, G.: „Introduction to programming with OpenCV“, in Illinois Institute of Technology unter http://www.academia.edu/download/30301467/introduction_to_programming_with_opencv.pdf (abgerufen am 15.04.2019)

Linksammlung

Index	Link	Letzter Zugriff	Beschreibung
01	https://github.com/Joo200/BaboCam	31.05.2019	Github-Verzeichnis des Projektes
02	https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/	30.05.2019	Blog-Beitrag Ball-Tracking mit OpenCV
03	https://www.robocupgermanopen.de/en	30.05.2019	Webseite RoboCup
04	http://www.robocuphumanoid.org/wp-content/uploads/RCHL-2019-Rules-final.pdf	30.05.2019	Webkampf-Regeln RoboCup
05	https://spl.robocup.org/open-source/	31.05.2019	OpenSource-Projekte von RoboCup
06	https://www.funkschau.de/telekommunikation/artikel/162962/	31.05.2019	Beitrag zu eingesetzten Sensoren in autonomen Fahrzeugen
07	https://www.etz.de/files/05_06_jamal.pdf	30.05.2019	Sensorfusion im autonomen Fahren
08	https://github.com/IntelRealSense/librealsense/	01.06.2019	Github-Seite RealSense
09	https://github.com/IntelRealSense/librealsense/releases/tag/v2.22.0	01.06.2019	Release Notes RealSense v2.22.0
10	https://www.sick.com/de/de/mess-und-detektionsloesungen/2d-lidar-sensoren/tim5xx/tim551-2050001/p/p343045	30.05.2019	Produktseite zu Sensor Sick TIM-551
11	http://wiki.ros.org/sick_tim	30.05.2019	ROS Treiber für Sick TIM-551
12	https://de.wikipedia.org/wiki/TOF-Kamera	31.05.2019	Wikipedia-Artikel TOF-Kamera
13	https://www.asus.com/de/3D-Sensor/Xtion_PRO/	31.05.2019	Produktseite Nachfolge-Artikel Asus Xtion
14	https://en.wikipedia.org/wiki/OpenNI	31.05.2019	Wikipedia-Artikel OpenNI
15	https://ark.intel.com/content/www/de/de/ark/products/92329/intel-realsense-camera-sr300.html	31.05.2019	Produktseite Intel SR300
16	https://www.intelrealsense.com/depth-camera-d435i/	01.06.2019	Produktseite Intel D435i
17	http://www.sharp-world.com/products/device/lineup/selection/opto/haca/diagram.html	30.05.2019	Sharp Produktseite Laserdistanz-Sensoren
18	https://www.microsonic.de/de/service/ultraschallsensoren/prinzip.htm	30.05.2019	Erklärung Funktionsweise Ultraschall-Sensoren
19	https://de.wikipedia.org/wiki/Induktiver_N%C3%A4herungsschalter	30.05.2019	Wikipedia-Artikel zu induktive Näherungsschalter

20	https://de.wikipedia.org/wiki/Kapazitiver_N%C3%A4herungsschalter	30.05.2019	Wikipedia-Artikel zu kapazitiven Näherungsschalter
21	https://de.wikipedia.org/wiki/FTDI	31.05.2019	Wikipedia-Artikel zu FTDI-Chips
22	http://yujinrobot.github.io/kobuki/enAppendixProtocolSpecification.html	02.06.2019	Protokoll-Spezifikation Turtlebot
23	https://github.com/IntelRealSense/librealsense/blob/master/doc/rs400/rs400_advanced_mode.md	02.06.2019	Erklärung Advanced Mode
24	https://www.ubuntu.com/download/desktop	02.06.2019	Webseite von Ubuntu
25	https://cmake.org/cmake-tutorial/	01.06.2019	Getting Started cmake
26	https://libserial.readthedocs.io/en/latest/	02.06.2019	ReadTheDocs zu LibSerial
27	https://opencv.org/	02.06.2019	Webseite von OpenCV
28	https://github.com/nlohmann/json	01.06.2019	Github-Seite von nlohmann json
29	https://www.learnopencv.com/install-opencv3-on-ubuntu/	01.06.2019	Installationsanleitung OpenCV
30	https://github.com/IntelRealSense/librealsense/blob/master/doc/distribution_linux.md	02.06.2019	Installationsanleitung LibRealSense
31	https://stackify.com/syslog-101/	02.06.2019	Getting Started Syslog
32	https://git-scm.com/book/en/v1/Getting-Started	01.06.2019	Getting Started Git
33	https://de.wikipedia.org/wiki/Spannungsteiler#Einfacher_Spannungsteiler_mit_zwei_ohmschen_Widerst%C3%A4nden	02.06.2019	Erklärung Spannungsteiler auf Wikipedia
34	https://www.python.org/about/gettingstarted/	01.06.2019	Getting Started Python
35	https://de.wikipedia.org/wiki/Secure_Shell	02.06.2019	Wikipedia-Artikel SSH
36	https://de.wikipedia.org/wiki/Generischer_Typ	01.06.2019	Wikipedia-Artikel Generischer Typ

Abkürzungen

Abkürzung	Bezeichnung	Beschreibung
BBB	BeagleBone Black	Einplatinencomputer
Lidar	Light detection and ranging	Methode zur optischen Abstands- und Geschwindigkeitsmessung
ADC	Analog Digital Converter	Ein Wandler, der eine analoge Spannung in einen digitalen Wert wandelt.
AD	Analog Digital	
JSON	JavaScript Object Notation	Kompaktes Dateiformat zum Datenaustausch zwischen Anwendungen
USB	Universal Serial Bus	Serielles Bussystem zur Verbindung von externen Geräten mit einem Computer

Begriffserklärungen

Callback: Die Methode, die nach einer bestimmten Aktion aufgerufen wird. Dies tritt zum Beispiel ein, wenn ein bestimmtes Paket empfangen wird.

Segway: Ein einachsiges Gefährt, welches sich selbst balanciert. Ein Passagier kann durch seine Körperneigung die Richtung und Geschwindigkeit vorgeben.

OpenCV: Eine Sammlung von Klassen und Methoden, um Bilder und Videos zu verarbeiten.

Framework: Eine Rahmenstruktur, welche als Programmierschnittstelle dient.

Ubuntu: Betriebssystem, Unix-Architektur, Linux-Distribution

Bildverzeichnis