FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Speckle Tracking

ACTO Tracking durch optische Maussensoren

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Daniel Bernhardt, BSc
Matrikelnummer 0826223

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Mag. Dr. Hannes Kaufmann
Mitwirkung: Dipl.-Ing. Mag. Emanuel Vonach, Bakk.

Wien, 22. August 2016

Daniel Bernhardt	Hannes Kaufmann

Erklärung zur Verfassung der Arbeit

Daniel Bernhardt, BSc
Rosenhügelstraße 171, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. August 2016

Daniel Bernhardt

Danksagung

Ich möchte mich an dieser Stelle herzlich bei allen Personen bedanken, die an der Realisierung dieser Arbeit mitgewirkt haben.

Danken möchte ich in erster Linie Hannes Kaufmann, der diese Arbeit möglich gemacht und betreut hat, sowie Emanuel Vonach für die moralische Unterstützung, sein Know-How und seinen Ideenreichtum bei der Lösung technischer Probleme. Florian Käferböck gilt mein Dank für die Hilfe bei mathematischen Fragen zu den verwendeten Algorithmen.

Schlussendlich gebührt auch meiner Familie und meinen Freunden Dank für deren Unterstützung und wichtige Anregungen während des Schreibens dieser Arbeit.

Kurzfassung

Der Einsatz von Tangible User Interfaces (TUIs) erscheint, unter anderem aufgrund ihrer häufig sehr intuitiven Bedienung (z.B. Computermaus, Lenkrad etc.) und der spielerischen Komponente, sowohl im medizinischen Bereich, als auch in der Entertainmentsparte vielversprechend. TUIs können mit Aspekten mobiler Roboter ergänzt werden, um Feedback in Form einer selbständigen Positionsänderung zu ermöglichen. Mit diesen gesteuerten (engl. actuated) TUIs beschäftigt sich das ACTO-Projekt (kurz für *Actuated Tangible User Interface Objects* [3]). Das Resultat sind kleine, würfelförmige Roboter - genannt ACTOs. Jedes ACTO ist modular aufgebaut und kann, zusätzlich zum Bewegungsmodul, mit verschiedenen Input- und Output-Erweiterungen bestückt werden.

Um die Bewegungen der ACTOs zentral zu steuern und zu koordinieren, muss deren Position in Echtzeit bestimmt werden. Jede Art der Positionsbestimmung hat ihre eigenen Vor- und Nachteile. Bisher fand die Positionsbestimmung ausschließlich durch das optische Tracking eines Markers auf der Unterseite jedes ACTOs statt. Diese Form des Trackings liefert zwar recht präzise und absolute Positionswerte, zeigt allerdings Schwächen bei schlechter Beleuchtung und zu schnellen Bewegungen des ACTOs. Um diese Nachteile zu kompensieren, stellt diese Arbeit eine Erweiterung vor, die das Tracking durch zwei, an jedem ACTO befestigte, optische Maussensoren ergänzt. Dabei wird die Verschiebung entlang der X- und Y-Achse beider Sensoren ausgelesen. Aus diesen Werten wird mittels eines Algorithmus, die Translation und Rotation des ACTOs und damit dessen neue Position bestimmt.

Die Arbeit beschreibt die Design-Überlegungen und die Konstruktion dieses relativen Tracking-Systems. Dabei wird auf die Entwicklung der Hard- und Software eingegangen. Die mathematischen Hintergründe werden ebenfalls beleuchtet. Schlussendlich wird gezeigt, dass die Ergänzung des absoluten Trackings durch optische Maussensoren, im Bezug auf Präzision und Robustheit, sowie bei schnellen Bewegungen bestechende Vorteile bietet. Das System besteht aus günstigen, handelsüblichen Bauteilen und ist gut in das bestehende ACTO-System integrierbar. Damit erscheint der Einsatz des Systems in den verschiedensten Forschungsgebieten sinnvoll. Da sich während der Arbeit an den einzelnen Prototypen und dem Schreiben der Software außerdem ein großes Potential für weitere Modifikationen zeigte, mit denen das System an die verschiedensten Umgebungen und Herausforderung angepasst werden kann, wird abschließend auch auf diese Verbesserungen eingegangen.

Abstract

The use of Tangible User Interfaces (TUIs) for medical and entertainment purposes seems promising because of their mostly very intuitive handling and playful aspect (computer mouse, steering wheel, etc.). TUIs can be supplemented with elements of mobile robots to allow independent movement as feedback. The *Actuated Tangible User Interface Object*-project (ACTO [3]) is focused on these *actuated* TUIs, resulting in cubic robots called ACTOs. Every ACTO is modularly composed and, in addition to the movement unit, can be equipped with a variety of input and output extensions.

To control and coordinate the movement of the ACTOs, their position needs to be retrieved in real time. Every method of determining an ACTO's position has its own advantages and disadvantages. Until now, the position was obtained by the optical tracking of a marker at the ACTO's bottom. This tracking method provides precise, absolute tracking data but cannot cope with poor illumination or movement which is too fast. To compensate for these shortcomings, this thesis introduces an extension which supplements the existing tracking system by another tracking method based on two optical mouse sensors. In this process the translation along the X- and Y-axis is read from both sensors. Based on these four translation values, the ACTO's translation and rotation and therefore its new position is computed.

The thesis discusses the ideas leading to the design and construction of this relative tracking system and therefore elaborates on the development of the hard- and software, as well as the mathematical background. In the end it is shown that the supplement of the absolute tracking by the relative tracking based on two optical mouse sensors provides considerable advantages in terms of precision and robustness. The system is composed of low-priced, off-the-shelf components and is easily integrated into the existing ACTO system. Thus, the use of this system seems promising in various research areas. During the work on several prototypes and the coding of the software, great potential for more modifications and adaptions to various environments was revealed which is discussed in the last part of this thesis.

Inhaltsverzeichnis

1. Einleitung

Mobile Roboter gewinnen allgemein und speziell im medizinischen Bereich immer mehr an Bedeutung. Die Einsatzbereiche sind vielfältig und reichen vom Haushalt, über Rehabilitation bis zu Anwendungen in Virtual Reality. Autonom agierende Staubsaugerroboter fallen, ebenso wie elektrische Rollstühle oder sich selbst bewegende Schachfiguren, unter diesen weit gefassten und nicht eindeutig definierten Begriff. Hertzberg et al. verstehen im Buch *Mobile Roboter* unter einem Roboter

> „... eine frei programmierbare Maschine, die auf Basis von Umgebungssensordaten in geschlossener Regelung in Umgebungen agiert, die zur Zeit der Programmierung nicht genau bekannt und/oder dynamisch und/oder nicht vollständig erfassbar sind.“ [1]

Der Begriff *geschlossene Regelung* bedeutet in diesem Zusammenhang, dass der Roboter sein Verhalten aufgrund von Umgebungsdaten ständig anpasst und somit auf Umwelteinflüsse reagiert, um ein bestimmtes Ziel zu erreichen. Ein *mobiler* Roboter kann sich in seiner Umgebung mehr oder weniger frei bewegen.

Tangible User Interfaces (TUIs) sind vor allem in der Rehabilitation und Virtual Reality relevant. TUIs sind anfassbare Benutzerschnittstellen und erlauben eine Interaktion zwischen Mensch und Maschine (*Human Computer Interaction* - HCI). Dabei kann ein TUI sowohl Input akzeptieren als auch Output liefern.

Für eine lange Zeit schien sich die Human Computer Interaction auf die Benutzung von Maus und Tastatur zu beschränken, um mit Fenstern, Icons, Menüs und Zeigern (*Windows, Icons, Menus, Pointer* - WIMP) zu interagieren. Bei allen Anwendungen, unabhängig vom Anwendungsgebiet, wurden die gleichen Eingabegeräte verwendet. In den letzten zwei Jahrzehnten wurden viele, vom WIMP-Prinzip abweichende, Interaktionsmuster entwickelt. Diese neuen Interaktionsarten, wie zum Beispiel Zeige- (Wii Remote Controller) oder Multitouch-Gesten (Touchscreen), beziehen die reale Welt stärker mit ein als zu vor. Gleichzeitig werden, weitgehend unbemerkt, Mikroprozessoren in immer mehr Alltagsgegenstände integriert als zuvor. [2]

TUIs erweitern physische Objekte um Computeraspekte, indem sie sie mit der digitalen Datenwelt vernetzen. Durch die direkt greifbare Repräsentation von digitalen Daten können TUIs sowohl als Eingabe- als auch als Ausgabegerät fungieren und Feedback bereitstellen. Dabei beschränkt sich dieses Feedback nicht auf auditive und visuelle Reize, sondern wird um die taktile Wahrnehmung erweitert. Des Weiteren können Daten nicht nur zweidimensional auf einem Bildschirm dargestellt werden, sondern auch dreidimensional. Das Design und die Aspekte, die TUIs ausmachen, werden ständig weiterentwickelt. [2]

Das ACTO-Projekt des IMS forscht an modularen, selbstbewegten Tangible-User-Interfaces und verbindet damit die Konzepte des TUIs und die mobiler Roboter. [3] Das ACTO ist ein würfelförmiger, circa 5 Zentimeter großer Miniroboter und besteht grundsätzlich aus einer Antriebseinheit, einem Akku, dem Mikroprozessor und einem Erweiterungsmodul (siehe 3.2). Die Erweiterung kann flexibel ausgetauscht werden und

die verschiedensten Aufgaben erfüllen. So können zum Beispiel ein Thermometer oder eine digitale Anzeige als Output oder Buttons als Input integriert werden. Generell übernimmt das Erweiterungsmodul die Input- und Output-Funktionen zur Kommunikation mit Menschen. Die Antriebseinheit kann ebenfalls ausgetauscht werden und beinhaltet derzeit 2 Motoren mit Getriebe, um 2 Räder an der Seite des ACTOs anzutreiben.

1.1. Problemstellung

Für die Bewegung und Steuerung der ACTOs ist es notwendig deren aktuelle Position zu bestimmen. Hierfür gibt es verschiedene mit Vor- und Nachteilen behaftete Systeme. Derzeit kommt optisches Tracking mittels Kamera und Markern zum Einsatz. Dabei wird ein auf der Unterseite des ACTOs angebrachter Marker von unten durch die Plexiglasplatte des Tisches gefilmt. Da die Größe des Markers bekannt ist, kann so auf seine relative Positionierung zur Kamera geschlossen werden. So ist zwar ein verhältnismäßig genaues, absolutes Tracking möglich, es wird allerdings ein recht aufwendiges Setup mit einem Plexiglastisch und guter Beleuchtung benötigt. Zum einen steigern diese Hardwareanforderungen die Kosten des Systems. Außerdem ist die derzeitige Tracking-Lösung, durch die Notwendigkeit einer speziellen, durchsichtigen Tischplatte, weniger mobil. Die starke Abhängigkeit von guten Lichtverhältnissen macht das Setup zusätzlich komplexer. Gerade im medizinischen Bereich wäre eine einfachere Handhabung wünschenswert, um die Akzeptanz des Systems zu gewährleisten. Außerdem weist das optische Tracking Schwächen bei höheren Geschwindigkeiten auf. Da TUIs und die ACTOs konzeptbedingt häufig bewegt werden, erfordert ein robustes Tracking nur mit Kamera und Marker optimale Bedingungen.

Speckle Tracking stellt ein relatives Tracking basierend auf 2 optischen Maussensoren (ADNS 9500 [4]) bereit und wird in dieser Diplomarbeit als Ergänzung zum derzeit eingesetzten, optischen Tracking vorgeschlagen. Dafür werden 2 Maussensoren auf der Unterseite des ACTOs befestigt und messen die Bewegung über den Untergrund. Vereinfacht gesagt beleuchtet jeder Sensor den Untergrund mit kohärentem Licht und registriert das reflektierte Licht. Wenn die Oberfläche hinreichend rau ist, entstehen durch die unterschiedlich langen Laufzeiten des Lichts Interferenzen. Das Resultat ist ein körniges Muster in Graustufen, das die Oberfläche repräsentiert. Die Verschiebung dieses Musters, des *Speckle-Patterns*, wird analysiert und daraus die zurückgelegte Strecke entlang der x- oder y-Achse berechnet (für Details siehe 3.4). 2 Sensoren sind notwendig, da optische Maussensoren keine Rotation erkennen. Deswegen wird die Rotation und die zurückgelegte Strecke aus den Δx- und Δy-Werten von mindestens 2 Sensoren berechnet (siehe 3.5).

1.2. Ziele der Arbeit

Diese Arbeit soll zeigen ob optische Maussensoren für das Tracking von mobilen Robotern und insbesondere ACTOs geeignet sind. Das Ziel dieser Arbeit ist ein funktionstüchtiger Prototyp des Tracking-Systems, sowie dessen Einbindung in das ACTO-Framework. Zu diesem Zweck wird anhand der Datenblätter eine Platine entworfen, die über eine

ebenfalls selbst konstruierte Extension mit dem ACTO verbunden wird.

Das Tracking-System wird in mehreren Schritten entwickelt, um auftretende Fehler und Probleme möglichst schnell eingrenzen und beheben zu können.

- Zuerst wird überprüft ob die Kommunikation zwischen Arduino und ADNS 9500 über *Serial Peripheral Interface* (SPI) möglich ist. Dazu wird eine Platine als Halterung für den Sensor mit entsprechenden Anschlüssen für Kabel entworfen. So kann der Sensor an ein Breadboard und Arduino UNO (Mikrokontroller) angeschlossen werden und getestet werden.

- Anschließend wird eine weitere Platine konstruiert die am ACTO montiert werden kann. Auf dieser Platine finden bereits beide ADNS 9500-Sensoren Platz. Dieser Schritt dient zur Entwicklung der nötigen Software und zur Anpassung des Algorithmus. Zur Steuerung der Sensoren wird allerdings weiterhin ein Arduino eingesetzt.

- Danach wird das System am ACTO befestigt und angeschlossen. So kann die Tracking-Software ins ACTO-Framework integriert werden. Außerdem wird es notwendig die SPI-Implementierung anzupassen, da der im ACTO integrierte Funkchip ebenfalls über SPI kommuniziert. Um den neuen Anforderungen gerecht zu werden, wird nicht nur die Kontrollsoftware des ACTOs sondern auch die Software des Funkempfängers und das in Java geschriebene Framework am Androidsteuergerät adaptiert.

- Abschließend wird das Hardwarelayout auf Basis der gewonnenen Erfahrungen überarbeitet und für einen Einsatz mit dem ACTO optimiert.

Es wird eine Reihe von theoretischen und praktischen Tests durchgeführt. Darunter fallen synthetische Tests zur Genauigkeit des eingesetzten Algorithmus, genauso wie praxisorientierte Tests, die eine Aussage über die Präzision im Einsatz geben sollen. Eine wichtige Rolle kommt in diesem Zusammenhang der Kalibrierung der Sensoren zu. Außerdem soll die Bedeutung von Einflüssen wie der Akkuleistung oder der Bewegungsgeschwindigkeit abgeschätzt werden. Folgende Testreihen werden durchgeführt:

- Synthetische Tests: Der Algorithmus wird mit künstlichen Daten getestet, um systematische Fehler erkennen zu können.

- Konsistenztests: Mit diesen Tests soll geklärt werden, inwiefern sich Faktoren wie der Akkuladezustand und die Bewegungsgeschwindigkeit auf die Genauigkeit auswirken.

- Performancetests: Schlussendlich werden reproduzierbare Bewegungsmuster benutzt, um die Präzision in der Praxis quantifizieren zu können.

1.3. Grenzen dieser Arbeit

Das Tracking von TUIs ist ein sehr großes und aktuelles Forschungsgebiet. Viele verschiedene Fachgebiete wie zum Beispiel Hardware- und Softwaredesign, Algorithmik, Statistik oder Usability sind an der Entwicklung von praxistauglichen Tracking-Systemen beteiligt. Diese Arbeit soll abschätzen ob (und wenn ja mit welchen Einschränkungen) optische Maussensoren für das Tracking des ACTOs geeignet sind. Das Ergebnis kann als Ausgangspunkt für die Forschung an Tracking-Systemen für andere TUIs herangezogen werden.

Um den Rahmen dieser Arbeit nicht zu sprengen, können nicht alle möglichen Aspekte des Systems getestet werden. So wird zum Beispiel nicht genauer auf den Einfluss der Oberflächenbeschaffenheit, oder des Abstands der Linse zur Oberfläche eingegangen. Kapitel 8 *Weitere Verbesserungsmöglichkeiten* befasst sich unter anderem mit Testreihen, die zur weiteren Analyse des Tracking-Systems sinnvoll wären.

Um aus den Messdaten der zwei Sensoren Tracking-Daten zu erhalten, wurde der Algorithmus von Miyazaki et al. 2007 [5] benutzt. Es wurde zwar ein alternativer Algorithmus getestet der zu den selben Resultaten führte (siehe 3.5), im fertigen Prototyp fand dieser aber keine Anwendung. Weitere Algorithmen oder Heuristiken zur Verbesserung der Genauigkeit wurden in dieser Arbeit nicht behandelt.

Ebenso wurde aus Zeit- und Kostengründen auf den Vergleich verschiedener Hardwarevarianten verzichtet. Die Verwendung anderer Sensoren, oder eine anderen Anordnung der Sensoren, könnte zur Verbesserung der Genauigkeit führen.

Aus diesen Gründen soll diese Arbeit demonstrieren, dass es grundsätzlich möglich ist mit leicht verfügbaren und billigen Bauteilen, sowie relativ einfachen Mitteln für die Konstruktion, das Tracking von kleinen, mobilen Robotern zu verbessern. Allerdings stellt das Resultat dieser Arbeit keine universelle Lösung zum Tracking von TUIs dar.

1.4. Aufbau der Arbeit

Im Kapitel *Related Work* werden relevante Forschungsarbeiten - unter anderem die verschiedenen Einsatzgebiete von TUIs - vorgestellt. Anschließend wird grob umrissen, welche Sensoren für TUIs geeignet sind und welche Tracking-Systeme, auf der Basis von Maussensoren bereits existieren.

In den *Grundlagen* wird der technische Hintergrund von Arduino und SPI beleuchtet. Des Weiteren werden der Aufbau des ACTOs, die Funktionsweise von *Speckle Sense* und der eingesetzte Algorithmus erläutert.

Im *Designprozess* wird auf den Weg von der Auswahl des Sensors bis zum fertigen Prototyp, sowie die zugrunde liegenden Designüberlegungen eingegangen. Ebenso werden die benutzten Werkzeuge beschrieben.

Der Abschnitt *Konstruktion* ist den handwerklichen Tätigkeiten gewidmet, die für die Konstruktion der Prototypen und der Hilfsmittel für die Testläufe notwendig waren.

Software und Framework handelt von den notwendigen Anpassungen am Arduino-, ACTO- und Framework-Code. Ebenso wird eine Anleitung zur Kalibrierung gegeben.

Die *Evaluierung* beschreibt den Aufbau und Zweck der einzelnen Testdurchläufe im

Detail. Direkt anschließend werden die gewählten Parameter und die daraus resultierenden Ergebnisse angeführt und interpretiert.

Da eine hundertprozentige Testabdeckung nie möglich ist und aus verschiedenen Gründen nicht alle potentiellen Verbesserungen in das System implementiert werden können, wird im Kapitel *Weitere Verbesserungsmöglichkeiten* auf eine mögliche, weitere Vorgehensweise, im Rahmen von weiteren Arbeiten eingegangen.

Das *Fazit* erläutert anschließend anhand der Testresultate, ob (und wenn ja mit welchen Einschränkungen) ein Tracking-System auf Basis von 2 optischen Maussensoren für das Tracking des ACTOs geeignet ist.

Abschließend erfolgt ein kurzer *Ausblick* in die nahe und fernere Zukunft von Tracking-Systemen auf Basis von optischen Maussensoren.

2. Related Work

In diesem Abschnitt wird ein Überblick über verschiedenen Anwendungsgebiete von TUIs gegeben, damit sich ihre Einsatzgebiete und die dazugehörigen Tracking-Methoden besser verstehen lassen. Anschließend folgt ein Überblick über verschiedene Arten des Trackings von mobilen Robotern. Da der Begriff *mobiler Roboter* im Allgemeinen nicht exakt definiert ist, [1] werden in diesem Kapitel Tracking-Systeme beschrieben, für die die Anwendung mit dem ACTO-System grundsätzlich möglich scheint. Somit wird der Fokus auf Indoor-Tracking-Systeme gelegt, wie sie unter anderem bei Haushaltsrobotern zum Einsatz kommen. Obwohl für das Tracking optische Sensoren verwendet wurden, werden an dieser Stelle auch alternative Möglichkeiten für das Tracking vorgestellt. Ebenso wird erläutert, warum diese verworfen wurden.

2.1. Tangible User Interfaces - TUIs

Die Anfänge der Tangible User Unterfaces gehen bis in die 1990er zurück. Fitzmaurice et al. [6] schlugen 1995 vor, die Windows-, Icons-, Menus-, Pointer-Elemente (WIMP) eines klassischen GUIs, teilweise durch eine reale Repräsentation mit physischen Artefakten zu ersetzen. Dieser relativ neue Ansatz für *Human-Computer-Interaction* (HCI) erwies sich als vielversprechend und sollte den Autoren nach, als *Proof of Concept*, beziehungsweise als Ausgangspunkt für weitere Forschungen in diese Richtung verstanden werden.

Zwei Jahre später 1997, formulierten Ishii und Ullmer die Herausforderungen in HCI wie folgt:

> „We live between two realms: our physical environment and cyberspace. Despite our dual citizenship, the absence of seamless couplings between these parallel existences leaves a great divide between the worlds of bits and atoms. At the present, we are torn between these parallel but disjoint spaces.“ [7]

Sie kamen zu dem Schluss, dass die Darstellung von Daten und Informationen, ausschließlich über einen zweidimensionalen Bildschirm und farbige Pixel, eine Einschränkung darstellt, da die Sinne der Menschen in der realen Welt zu wesentlich differenzierteren Wahrnehmungsarten fähig sind.

Im Laufe der Jahre wurde an vielen verschiedenen Interaktionskonzepten in den unterschiedlichsten Anwendungsgebieten geforscht. Laut einer Übersicht von Shaer und Hornecker aus dem Jahr 2010 [2] dominieren neben anderen Anwendungsdomänen die folgenden:

- Lernanwendungen (*TUIs for Learning*)

  Viele TUIs können als computerunterstützte Lernwerkzeuge angesehen werden. Das hat mehrere Gründe. Erstens wurde von den Lernforschern und Spielzeugherstellern schon immer versucht, Spielzeuge zu verbessern und mit mehr Funktionen auszustatten. Zweites sprechen physische Lernumgebungen alle Sinne an und unterstützen damit die Entwicklung des Kindes. Weiters richten sich Lernwerkzeuge,

die die Grundlagen eines bestimmten Gebietes vermitteln und so mit einer einfacheren Umgebung auskommen, meistens an Anfänger. Deswegen erübrigen sich oft Probleme die mit einer komplexeren Umgebung einhergehen.

- Problemlösen und Planen (*Problem Solving and Planning*)

  Drei Aspekte haben sich als effektiv herausgestellt, um das Problemlösen zu unterstützen: (1) epistemische Aktionen, (2) physische Einschränkungen, (3) greifbare Repräsentation eines Problems. Epistemische Aktionen zielen darauf ab eine Aufgabe in ihrer Natur zu verändern [8] (meist um sie zu vereinfachen). So können zum Beispiel Objekte rotiert werden um sie auf einer Oberfläche anzuordnen. Damit müssen die Objekte nicht mental *im Kopf* rotiert werden, bevor sie angeordnet werden können. Physische Einschränkungen können benutzt werden, um den Spielraum der Lösungswege und Lösungen einzuschränken. So können Regeln implementiert werden, ohne dass es notwendig ist, sie vor der Interaktion expliziert zu erlernen. Schlussendlich drängt sich die Nutzung von TUIs zur Manipulation geometrischer Formen nahezu auf, etwa bei der Stadtplanung oder in der Architektur.

- Informationsvisualisierung (*Information Visualization*)

  Durch das breite Angebot an Repräsentationsarten und die Möglichkeit zur Steuerung mit 2 Händen, haben die TUIs großes Potential die Interaktion mit Visualisierungen zu verbessern. Viele Anwendungen nutzen die haptische Interaktion, um Informationen in Visualisierungen zu erfahren und zu manipulieren. So können zum Beispiel Schnittebenen von Objekten in der Neurochirurgie oder Geophysik gesteuert werden.

- Greifbares Programmieren (*Tangible Programming*)

  Das Konzept TUIs für das Schreiben von Computerprogrammen zu benutzen, gibt es bereits seit mehreren Jahrzehnten. Radia Perlmans entwickelte ein *Slot Machine*-Interface, um Kindern zwischen 3 und 5 Jahren das Programmieren mit physischen Karten und Buttons zu ermöglichen. [9] *AlgoBlocks* unterstützt Kinder beim Programmieren-Lernen, indem es sich wie ein Videospiel verhält. Beide Systeme benutzen Programmbausteine der erziehungsorientierten Programmiersprache *LOGO*. Alan Blackwell beschreibt 2003 die Entwicklung einer funktionalen Programmiersprache, auf Basis des *Cognitiv Dimensions*-Frameworks, für die Anwendung in Klassenzimmern. [10].

- Entertainment, Spielen und Edutainment (*Entertainment, Play and Edutainment*)

  Die Grenzen zwischen Entertainment, TUI-verwandtem Spielzeug und Edutainment überlappen. Das wohl bekannteste Beispiel ist die Nintendo Wii, die mit ihrem kommerziellen Erfolg das Marktpotential demonstriert. Es gibt allerdings auch Beispiele die dem TUI-Begriff besser entsprechen. Viele moderne Edutainment-Spielzeuge wenden TUI-Prinzipien an. Zum Beispiel vertreibt *Neurosmith* verschiedenfarbige *MusicBlocks*, mit denen Kinder Musik kreieren können.

- Musik und Performance (*Music and Performance*)

  Musikanwendungen gehören zu den ältesten Anwendungsgebieten von TUIs. Musik-TUIs zielen entweder auf Anfänger, als intuitives und einfach zu benutzendes Spielzeug ab. Oder sie sind für Profis designt, die die Ausdrucksstärke, Leserlichkeit und Sichtbarkeit des Systems, wenn sie vor einem Publikum auftreten, zu schätzen wissen. Viele dieser TUIs werden von Profis, wie Komponisten elektronischer Musik entwickelt. Die NIME-Konferenz (*New Instruments for Musical Expression*) ist die wichtigste auf ihrem Gebiet.

- Soziale Kommunikation (*Social Communication*)

  Seit ihrer Frühzeit wurden TUIs zur Unterstützung von Kommunikation angewendet. Die greifbaren Objekte erschienen passend um Personen zu repräsentieren. Eine Reihe von Projekten konzentriert sich auf die *remote awareness within social networks*. Zum Beispiel übertragen physische Objekte abstrakte Nachrichten bei verteilten Arbeitsgruppen. [11] Ein weiteres Anwendungsgebiet ist *remote intimacy*. Dabei geht es grundsätzlich um eine mehr oder weniger intime Art der Kommunikation zwischen Personen mittels TUIs. Ein Beispiel sind 2 Weingläser die aufeinander reagieren. Das eine leuchtet wenn aus dem anderem getrunken wird. So kann das Ritual des Weintrinkens über größere Entfernungen simuliert werden. [12]

- Greifbare Erinnerungen und Notizen (*Tangible Reminders and Tags*)

  TUI-Objekte werden benutzt um Erinnerungen und Notizen zu visualisieren. So beschäftigen sich Holmquist et al. mit der Nutzung von physischen Objekten, um Lesezeichen für Webseiten zu speichern und abzurufen. [13] Mugellini et al. präsentieren 2007 [14] ein Framework, mit dem persönliche Objekte als TUIs, zum Beispiel mit Urlaubserinnerungen, verknüpft werden können.

2.2. Tracking-Methoden

Im Folgenden wird ein Überblick über die verschiedenen Arten des Trackings mobiler Roboter gegeben. Alternative Tracking-Methoden werden ebenso behandelt wie das Tracking mit optischen Maussensoren, auf welchem der Prototyp basiert.

2.2.1. Beschleunigungssensoren

Die Ausgangslage und Geschwindigkeit des Roboters wird initial in Relation zur Erdoberfläche gemessen. Danach auftretende Bewegungen werden in Form der Beschleunigung über die Massenträgheit gemessen. Integriert man die Beschleunigung über die Zeit erhält man die Geschwindigkeit. Integriert man die Geschwindigkeit über die Zeit erhält man den Weg.

Eine Drehbeschleunigung kann mit einem Gyroskop gemessen werden. Bei einem mechanischen Gyroskop rotiert eine Masse (Scheibe) um eine bestimmte Achse. Wird diese Achse durch einen externen Impuls ausgelenkt, erfolgt eine Auslenkung senkrecht zur angreifenden Kraft, um den Gesamtimpuls zu erhalten.

Bei Robotern wird häufig ein *Micro Electronical Mechanical System* (MEMS) eingesetzt. Dabei wird die Auslenkung einer, durch elektrostatische Prozesse zum Schwingen gebrachten, Siliziummasse gemessen, um die Beschleunigung zu bestimmen. [1] Der Begriff *MEMS* fasst ein sehr weites Spektrum. Markus Glück definiert es 2005 wie folgt:

> „Unter dem Oberbegriff der 'Mikroelektromechanischen Sensorsysteme' (MEMS) wird die Integration von Sensoren und Aktuatoren der Mechatronik, der Mikrosystemtechnik, der Mikrooptik, der Mikrofluidik und der Mikroelektronik zusammengefasst.“ [15]

Ein Ziel ist auch die Miniaturisierung der Systeme. Auf diese Art verkleinerte Beschleunigungssensoren finden in viele Gebieten, wie zum Beispiel bei der Auslösung von Airbags, oder der Messung von Vibrationen von Roboterarmen in der Fertigung Anwendung. [15] Mit Beschleunigungssensoren ist eine absolute Positionsbestimmung nicht möglich. Da sich kleine Messfehler verursacht durch die begrenzte Genauigkeit aufsummieren, erscheint ein Einsatz zum Tracking des ACTOs nicht zielführend. Optische Maussensoren messen zwar auch nur die relative Bewegung, allerdings kann das Speckle-Pattern (siehe Kapitel 3.4 und Abbildung 10) als Repräsentation des Untergrundes ausgelesen werden. Dadurch können mit einer entsprechenden Implementierung, theoretisch Rückschlüsse auf die absolute Position gezogen werden (siehe Kapitel 8).

2.2.2. Messung der Radumdrehung

Bei Robotern, die sich auf Rädern fortbewegen, kann die zurückgelegte Entfernung durch die Bewegung der Räder gemessen werden. Beziehungsweise werden die Räder so gesteuert, dass sich der Roboter unter optimalen Bedingungen genau auf dem geplanten Pfad bewegt. Besitzt zum Beispiel ein Roboter genau 2 Räder und wird von diesen gleichzeitig angetrieben, ergibt sich die zurückgelegte Strecke durch die gesteuerte Bewegung durch die Räder und wird streng genommen nicht unabhängig gemessen. Da der zurückgelegte Weg exakt durch die Bewegung der Räder definiert wird und keine weiteren Informationen zur Verfügung stehen, können Messfehler nicht kompensiert werden.

Denkbar wäre ein System, sowohl mit angetriebenen, als auch nicht angetriebenen Rädern, wobei letztere zusätzlich zur Messung herangezogen werden. Die Messung der nicht angetriebenen Räder entspricht eher dem klassischen Verständnis eines Tracking-Systems. Auch wenn mit zusätzlichen Rädern die Messewerte liefern, die Fehler zum Teil kompensiert werden können, bleibt ein grundsätzliches Problem bestehen. Da die Bewegung inkrementell gemessen wird, wirken sich kleine Fehler mit der Zeit immer gravierender aus. Wenn beispielsweise von 2 Rädern eines um eine Umdrehung „durchrutscht “, das andere aber nicht, weicht die geschätzte Ausrichtung des Roboters unter Umständen erheblich von der tatsächlichen ab. Dadurch nimmt der Fehler bei jeder Bewegung rapide in eine bestimmte Richtung zu.

Sekimori und Miyazaki errechneten unter anderem aus der Bewegung der Räder den zurückgelegten Pfad eines Roboters. Der Roboter bewegte sich dabei über eine Strecke von $2.5m$ und vollführte dabei vier 90° Manöver. Je nach Geschwindigkeit betrug die

Abweichung durch das Durchrutschen der Räder, (ohne komplexe Kurvenfahrten) am Ende zwischen $20cm$ und $30cm$. [16] Das Tracking des ACTOs wäre zwar theoretisch möglich, würde man die Bewegung der 2 Räder als Datenbasis nehmen. Allerdings wäre kein Tracking möglich, sobald das ACTO aufgehoben und/oder von Hand bewegt wird. Das und die zu erwartende Ungenauigkeit wären in Hinsicht auf die Ziele dieser Arbeit nicht akzeptabel.

2.2.3. Ultraschall- und Laserdistanzmessung

Wenn die Umgebung des Roboters bekannt ist, kann theoretisch die Position des Roboters durch eine Messung der Entfernung zur Umgebung bestimmt werden. Grundsätzlich ist eine Messung der Umgebung und eine Entfernungsbestimmung mit Ultraschall möglich. Ein Vorteil ist, dass Ultraschall (im Gegensatz zu Licht) auch von Glas vorhersehbar reflektiert wird. Für die praktische Anwendung ist Ultraschall im Allgemeinen aber zu ungenau ($1m$ gilt als gut). [17] Vielversprechender ist die Entfernungsmessung durch Lasersensoren. Meist werden mehrere Lasersensoren scheibenförmig in einer Ebene angeordnet. Diese Scheibe wird rotiert und die Entfernung zur Umgebung gradweise gemessen. Diese Entfernungsmessung kann unter guten Bedingungen sehr genau sein ($2mm$ Abweichung). [17] Da auf diese Weise aber immer nur in einer Ebene gemessen werden kann, kommt es zu Problemen bei der Erkennung von am Boden liegenden Gegenständen, Tischplatten oder dergleichen. Um dieser Einschränkung entgegen zu wirken, gibt es auch Lasersensorsysteme bei denen die Messebene rotiert wird. So kann zwar der gesamte Raum abgedeckt werden, allerdings ist dies mechanisch aufwändiger und die rotierenden Laserstrahlen sind in Augenhöhe, je nach Leistung, nicht unbedenklich. [17]

Denkbar wäre auch eine Messung in die umgekehrte Richtung - von der Umgebung in Richtung Roboter - um dessen Ausrichtung und Position zu bestimmen. Henrik Andreasson und Tom Duckett konstruierten 2003 einen Roboter, der mit Hilfe einer omnidirektionalen Kamera und einem Laser-Range-Finder, einige gewöhnliche Objekte in einer Büroumgebung erkennen konnte. [18] Jung et al. stellten 2009 eine verwandte Arbeit vor, in der von einem Roboter aus, mit einer Kamera und einem Laser-Range-Finder, die Position eines sich bewegenden Objekts bestimmt werden konnte. [19]

Mit Laserdistanzsensoren kann die Position eines Roboters, in einer bekannten Umgebung, grundsätzlich sehr präzise bestimmt werden. Gegen den Einsatz mit dem ACTO spricht die aufwändige mechanische Konstruktion, die für die Sensoren notwendig wäre. Außerdem müsste sich diese auf der Oberseite des ACTOs befinden und würde damit die Verwendung anderer Erweiterungen (siehe 3.2.3) unmöglich machen. Der erhöhte Strombedarf stellt eine zusätzliche Herausforderung dar.

2.2.4. Tracking mit optischen Maussensoren

Die Idee billige Maussensoren für das Tracking von Robotern einzusetzen ist nicht neu. Ein optischer Maussensor kann keine Rotation, sondern nur Bewegung entlang der x- und y-Achse messen. Meistens finden daher, um Kurven messen zu können, 2 Maussensoren Anwendung. [20][21] Andere Veröffentlichungen, wie Sekimori und Miyazaki 2005 [5] und

Sekimori und Miyazaki 2007 [16], welche weiter unten beschrieben werden, benutzen mehr als 2 Sensoren, oder untersuchen den Einfluss einer Erhöhung der Sensoranzahl.

Singh und Waldron [20] beschäftigen sich mit den mathematischen Grundlagen, um aus der zweidimensionalen Bewegung von 2 Sensoren die Drehung und abschließende Position zu konstruieren. Da diese Berechnungen für den verwendeten Mikrocontroller verhältnismäßig aufwendig sind, wurden entsprechende Optimierungen des Algorithmus vorgestellt. Anwendung fanden die Sensoren aber nicht in einem sich selbst bewegenden Roboter, sondern in einem mausähnlichen Zeigegerät (Abbildung 1) mit dem Namen DesktopBot. Dabei wurde als Gehäuse keine handelsübliche Maus verwendet, da der Platz im Inneren nicht ausreichte. Stattdessen wurde zuerst eine Schale aus Plastik in der erforderlichen Größe designt. Da diese aber zu grob gearbeitet und instabil war, wurde der finale Prototyp aus Aluminium gefertigt.

Abbildung 1: Desktop Bot aus Aluminium [20]

Schlussendlich stellte sich heraus, dass mit der Hilfe von 2 handelsüblichen Sensorchips, das Tracking in 2 Dimensionen kostengünstig zu bewerkstelligen war.

In Cooney 2004 [21] wurden ebenfalls 2 optische Sensoren benutzt, um Drehungen auch während der Fahrt - also Kurven - messen zu können. Der Roboter selbst (Abbildung 2) war mit einem halben Meter Länge allerdings wesentlich größer als ein ACTO. Das hatte den Vorteil, dass vorne und hinten jeweils eine Maus montiert werden konnte. Eine weitere Besonderheit stellten die 4 ungelenkten *Mecanum-Räder* dar. Diese erlauben eine Fahrt in alle Richtungen sowie Drehungen, ohne über gelenkte Räder verfügen zu müssen. [22]

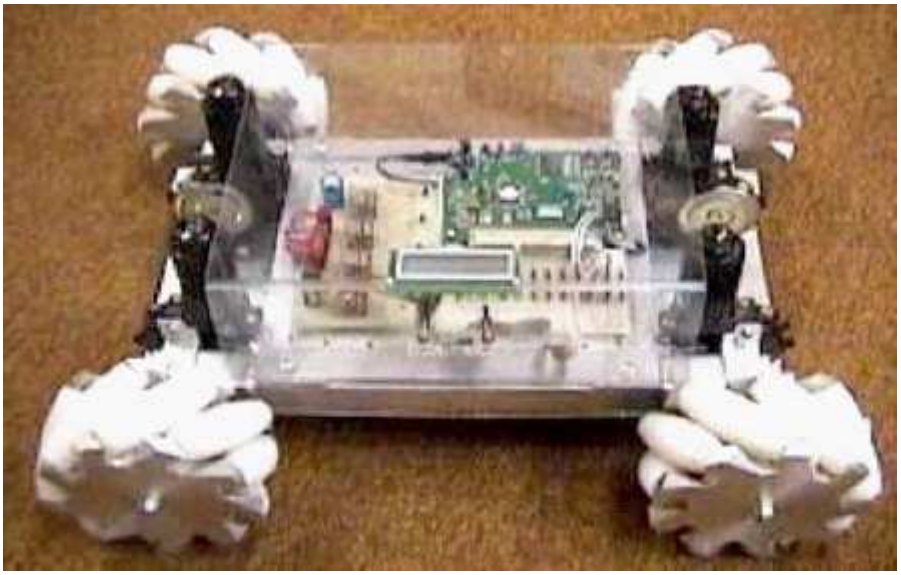


Abbildung 2: Roboter mit 2 Sensoren [21]

Da die beiden Mäuse so gut wie unverändert eingebaut wurden und damit auf den Sensorchip nicht direkt zugegriffen werden konnte, wurde ein eigener Treiber für das PS/2 Protokoll der Maus geschrieben. Aus den, von den beiden Mäusen gelieferten x- und y-Abstandswerten, konnte mit den trigonometrischen Sätzen die Rotation und die zurückgelegte Strecke und damit auch der Pfad bestimmt werden. Der Roboter musste anschließend einige Testfahrten auf einer Fläche von $160cm$ x $160cm$ absolvieren. Schlussendlich stellte sich heraus, dass der Roboter mit dieser Tracking-Methode sehr gut gesteuert werden konnte. Als Nachteil erwiesen sich die Mecanum-Räder und die Motoren, die teilweise durchrutschten, beziehungsweise sich nur ungenügend exakt ansteuern ließen. Da das ACTO als Interface auch manuell bewegt wird, stellen durchrutschende Räder allerdings keinen Nachteil dar.

Bonarini et al. beschäftigten sich 2004 mit systematischen Messfehlern, die durch die Annahme von falschen Parametern der optischen Sensoren zustande kommen. [23] Für ein Setup mit 2 Maussensoren, wurden drei verschiedenen Fehlerquellen identifiziert:

- eine ungenaue Position und Ausrichtung der beiden Sensoren zum gesamten Tracking-System (dem Roboter);
- die Auflösung der Sensoren abhängig von der Oberflächenbeschaffenheit des Untergrunds;
- von Sensor zu Sensor unterschiedliche Auflösungen.

Weiters wurde von Bonarini et al. ein Kalibrierungsprozess entwickelt, um diese Parameter abzuschätzen und damit den systematischen Fehlern entgegen zu wirken. Auch für das ACTO-Tracking-System wurde die Kalibrierung der Sensoren berücksichtigt (siehe 6.3).

Auf Fehlerquellen durch die ungenaue Positionierung der Sensoren und mögliche Verbesserungen, wird später in Kapitel 8 eingegangen.

2005 ging die gleiche Gruppe in einer ähnlichen Arbeit weiter auf nicht-systematische Fehler ein. [24] Unter der Annahme, dass von den insgesamt 4 gemessenen Werten der zwei Sensoren, nur drei für das Tracking notwendig sind und damit eine Redundanz besteht, wurde ein Algorithmus zur Fehlererkennung implementiert. Wichtige Anforderungen sind der konstante Abstand und die konstante Ausrichtung der beiden Sensoren zu einander. Zu Testzwecken wurde ein reflektierender Streifen auf dem Pfad des Roboters angebracht, um einen nicht-systematischen Messfehler zu provozieren. Bei diesem Testsetup befanden sich, bedingt durch die Position am Roboter, nie beide Sensoren gleichzeitig auf dem reflektierenden Streifen. Deswegen lieferte zumindest immer ein Sensor zuverlässige Messwerte. Unter diesen Testumständen verbesserte der vorgestellte Korrekturalgorithmus die Genauigkeit des Trackings, auf einer Strecke von einem Meter (vor und zurück) erheblich. Beim Speckle-Tracking für das ACTO wird kein derartiger Algorithmus zu Fehlererkennung benutzt. Allerdings böte sich so eine Möglichkeit die Genauigkeit weiter zu verbessern.

Kanburoglu et al. benutzten 2007 ein maschinengestütztes Verfahren, um die Abweichung zwischen der gemessenen und der von der Maus tatsächlich zurückgelegten Strecke standardisiert festzustellen. [25] Als Sensor kam in der Maus ein Agilent ADNS 2150 Chip zur Anwendung. Um in den 50 Testdurchgängen immer die gleiche Strecke zurückzulegen, wurde eine computergesteuerte Maschine zum Funkenerodieren (eng. *edm*) zweckentfremdet. Die Maus wurde am Werkzeug eingespannt und 2x25 mal mit einer Beschleunigung von $5mm/s^2$ und einer Geschwindigkeit von $10mm/s$, über $50mm$ auf einer polierten Gusseisenoberfläche bewegt. Der durchschnittliche Fehler betrugt bei dieser Vorgehensweise $0.422mm$, bei einer Standardabweichung von $0.0651mm$.

Da mit diesem Testaufbau nur Strecken bis zu $50mm$ zurückgelegt werden konnten, wurde ein neues Testsetup geplant. Anstatt die Maus über eine feste Oberfläche zu bewegen, wurde sie fix vor einer servoangetriebenen, rotierenden Scheibe montiert (Abbildung 3). Um einen Vergleichswert zu erhalten, wurde auf der anderen Seite ein hochauflösender Optical-Encoder positioniert. Die rotierende Scheibe simulierte so für die Maus, eine potentiell unendlich lange, lineare Bewegung.

Die Tests wurden auf sechs verschiedenen Oberflächen und mit 4 verschiedenen Geschwindigkeiten vorwärts und rückwärts durchgeführt. Bei niedrigen Geschwindigkeiten waren die Abweichungen in mm/m bei allen Oberflächen nicht signifikant. Bei Geschwindigkeiten die sich der Spezifikationsgrenze des Sensors näherten, wurden die Messergebnisse zunehmend ungenauer und beim Überschreiten dieser unbrauchbar. Dies legte den Schluss nahe, dass die maximale Arbeitsgeschwindigkeit in der Praxis niedriger war als vom Hersteller angegeben. Eine Oberfläche mit möglichst starken Kontrasten auf kleinem Raum auf der gesamten Scheibe, führte zu den besten Ergebnissen, da so der Sensor eine ausreichende Zahl an eindeutigen Merkmalen, sogenannten Features auf der Oberfläche registrieren konnte. Die Ergebnisse unterschieden sich, je nach Rotationsrichtung der Scheibe stark. In Kapitel 7.3.1 wird mit einem einfacheren Setup ebenfalls die Genauigkeit der Sensoren über eine bestimmte Strecke geprüft.

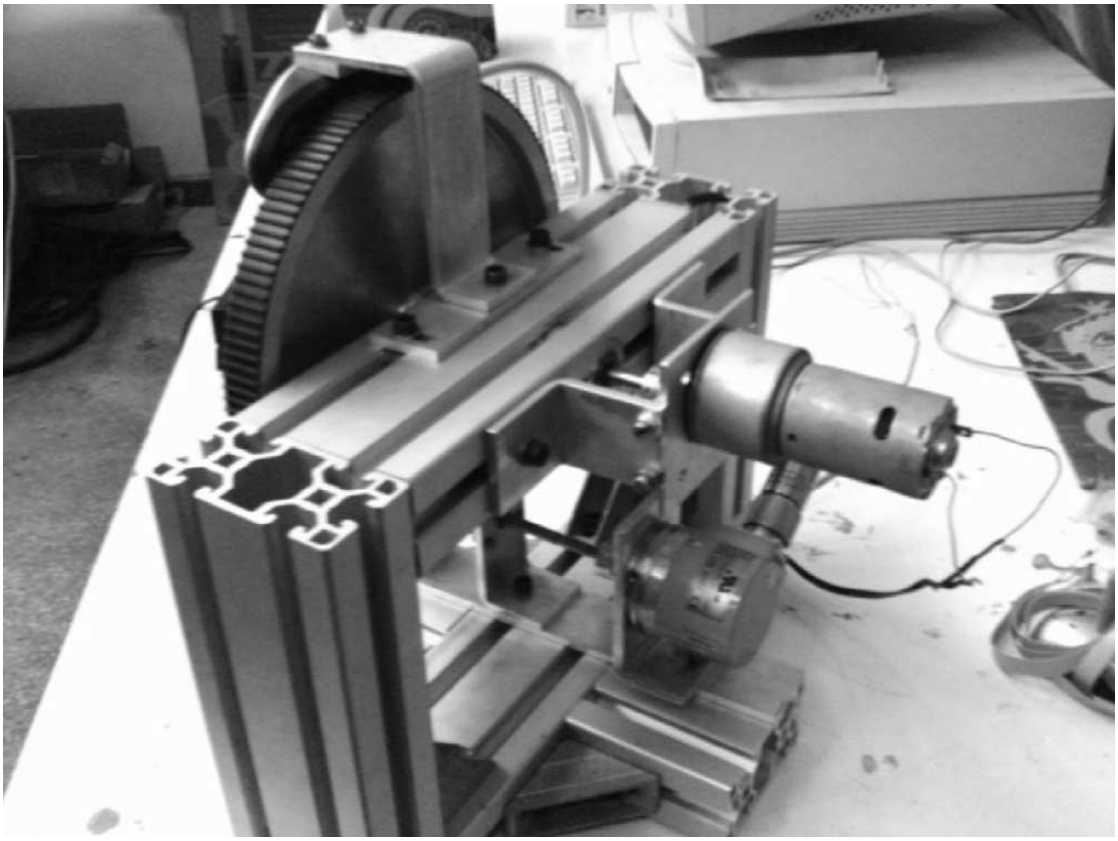


Abbildung 3: Testsetup mit Maus und servoangetriebener Scheibe [25]

Miyazaki und Sekimori beschäftigten sich 2005 ebenfalls mit dem Tracking durch 4 Maussensoren. [5] Um unrealistische und falsche Messergebnisse zu ignorieren, wurde ein spezieller mathematischer Filter (der Kalman-Filter) benutzt. Um ein robustes Tracking zu gewährleisten, wurde zusätzlich die globale Position eines Roboters mit einer Kamera bestimmt. Diese stellte allerdings die Orientierung der Roboter nicht fest. Im genannten Paper wurde auch der Einfluss der Abtastrate des Sensors auf die Genauigkeit des Trackings untersucht. Im Experiment sinkt die Genauigkeit bei einer Verringerung der Abtastrate, von $15Hz$ auf $5Hz$ nur unwesentlich. Die Abweichung steigt hierbei von $19.859mm$ auf $22.889mm$. Bei $3Hz$ liegt die Abweichung allerdings schon bei $39.927mm$ und steigt bei $1Hz$ weiter auf $197.858mm$. Dies lässt den Schluss zu, dass das Tracking auch bei niedrigen Abtastraten von $5Hz$ brauchbare Ergebnisse liefert. Unterhalb dieser Abtastrate sinkt die Präzision allerdings rasant.

Zwei Jahre später, im Jahr 2007, stellten Miyazaki und Sekimori ein ähnliches Paper, über die Verfeinerung des Trackings mit optischen Maussensoren vor. [16] Ein Problem beim relativen Tracking stellt der sich über die Zeit, beziehungsweise über die zurückgelegte Strecke, summierende Fehler dar. Dieser wird durch eine fehlerhafte Messung des Sensors verursacht. Um diese fehlerhaften Messungen zu kompensieren, können mehr

als 2 Sensoren verwendet werden. Im genannten Paper wurde die Zuverlässigkeit des Tracking-Systems mit 2 und 4 Sensoren verglichen. Um zuverlässige Messergebnisse der Maussensoren von unzuverlässigen zu trennen, wurde ein Algorithmus vorgestellt. Es wurde die Präzision des Trackings über die Radumdrehungen mit dem Tracking über 2 und über 4 Maussensoren verglichen. Um den absoluten Pfad zu messen, wurde eine auf der Decke über dem Testfeld installierte Kamera benutzt. Dabei wurde der Roboter mit 2 verschiedenen Geschwindigkeiten bewegt: a: $v = 300mm/s$ und $\omega = 1.82[rad/s]$ und b: $v = 500mm/s$ und $\omega = 3.03[rad/s]$. Geschwindigkeit a liegt unter der maximalen Arbeitsgeschwindigkeit des Sensors laut Datenblatt und b darüber. Dabei stellte sich heraus, dass die Bestimmung des Weges allein über die Radumdrehungen generell eher unzuverlässig war. Das Tracking mit 2 Sensoren stellte sich bei Geschwindigkeit a als zuverlässig heraus, versagte jedoch bei Geschwindigkeit b, die außerhalb der Spezifikationen des Sensors lag. Das Setup mit 4 Sensoren lieferte, aufgrund des benutzen Algorithmus zur Auswahl von zuverlässigen Messergebnissen, auch bei Geschwindigkeit b noch gute Ergebnisse. Das Testsetup wurde um eine weitere Kamera erweitert, die zusätzlich die absolute Position bestimmen konnte, nicht aber die Rotation. Dadurch ließ sich in Kombination mit Maussensoren, um die Rotation zu bestimmen, die Positionsbestimmung weiter verfeinern.

Für das in dieser Arbeit entwickelte Tracking-System für ACTOs wurde, wie in Kapitel 3.5 beschrieben, der in Miyazaki und Sekimori et al. 2005 [5] und Miyazaki und Sekimori et al. 2007 [16] vorgestellte Algorithmus benutzt.

Yi Dong-Hoon et al. untersuchten 2015, ob die Verwendung eines afokalen Sensorsystems mit einem Pinhole einen Vorteil gegenüber einem konventionellen, fokalen optischen Flow-Detection-System, wie zum Beispiel einer optischen Maus darstellt. [26] Zu diesem Zweck wurde die Genauigkeit der beiden Testsysteme, bei vertikalen Abständen zur Oberfläche zwischen $30mm$ und $50mm$, über eine Distanz von $80cm$ getestet. Für das vorgeschlagene afokale Sensorsystem ergab sich, für eine Höhe über der Oberfläche zwischen $30mm$ und $50mm$, ein durchschnittlicher Fehler von 0.1% pro $1mm$ Abweichung der Höhe. Bei dem Test mit dem fix fokussierten Sensorsystem wurde, für eine Höhe zwischen $30mm$ und $35mm$, ein durchschnittlicher Fehler von 14.7% pro $1mm$ Höhenabweichung gemessen. Allerdings stieg auch die Fehlerrate des afokalen Systems ab einer Höhe von $40mm$. Insgesamt wurde auf einer Teppichoberfläche bei dem afokalen System eine durchschnittliche Abweichung von 0.02% beobachtet, während der Fehler bei dem fix fokussierten System ohne Pinhole durchschnittlich 4.09% betrug. Die Ungenauigkeit des afokalen Systems stieg ab einer Höhe von $45mm$, weil eine ideale Größe des Pinholes von $0mm$ nicht möglich ist und die Beleuchtung mit der zunehmenden Entfernung abnahm. Beim ADNS 9500 [4] mit seiner Linse ADNS 6190 002 [27] handelt es sich um ein fokales System. Daher wurde der Threshold für die Lift-Detection auf das Maximum (laut Datenblatt) von $5mm$ gesetzt.

3. Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die für die Konstruktion eines Tracking-Systems auf Basis von Maussensoren und dem Arduino notwendig sind. Dies umschließt sowohl die grundsätzliche Funktionsweise und das Anwendungsgebiet des Arduinos, wie auch das Kommunikationsprotokoll SPI und die Maussensoren, die mit diesem angesteuert werden. Da das Tracking-System für eine Verwendung mit dem, auf dem Arduino basierenden, ACTO konzipiert ist, wird diesem ebenfalls ein eigener Abschnitt gewidmet. Anschließend werden die für das eigentliche Tracking erforderlichen, mathematischen Formeln vorgestellt. Diese sind notwendig, um aus den x- und y-Deltawerten der zwei Maussensoren absolute Positionen zu bestimmen.

3.1. Arduino

Arduino ist eine Open-Source-Prototypingplattform, die sowohl Hardware als auch Software beinhaltet. Arduino kann benutzt werden um verschiedene elektronische Bauteile zu steuern. [28]

Als Hardware stehen unterschiedliche Boards mit einem Mikrocontroller und analogen und digitalen Ein- und Ausgängen zur Verfügung. Die Boards unterscheiden sich zum Beispiel in den verwendeten Mikrocontrollern, der Anzahl der Ein- und Ausgänge, sowie dem Formfaktor und der verwendeten Eingangsspannung. Es gibt auch Arduino-Varianten mit einem zusätzlichen, stärkeren Mikroprozessor, oder sehr kleine, auf dem Arduino basierende Ableger mit integriertem Funkchip [29].

Um die Funktionalität der Maussensoren zu testen, wurde ein Arduino UNO benutzt. Das Arduino UNO besitzt einen ATmega328P Microcontroller mit $16Mhz$ und verwendet eine Arbeitsspannung von $5V$. Mit dem $32Kb$ großen Flash-Speicher des ATmega328P, steht genügend Speicherplatz für die ADNS 9500-Firmware zur Verfügung. Der SRAM- und EEPROM-Speicher würden alleine nicht ausreichen. Die Spannungsversorgung kann über den USB-Anschluss mit $5V$ sichergestellt werden. Alternativ kann eine externe Spannungsversorgung zwischen $7V$ und $12V$ angeschlossen werden. Diese kann zum Beispiel ein Netzteil oder eine Batterie sein. Da der ATmega328P nur mit einer Steckverbindung aufgesteckt ist, kann er bei einer Beschädigung einfach ausgetauscht werden.

Arduino stellt auch eine eigene integrierte Entwicklungsumgebung (IDE) zur Verfügung. Diese basiert auf der ebenfalls quelloffenen *Processing*-IDE [30]. Die IDE ist in Java geschrieben und plattformunabhängig. In der IDE wird ein sogenannter *Sketch* geschrieben und anschließend über USB auf das Arduino geladen. Ebenfalls via USB, können über den Serial-Monitor Daten vom Arduino gesendet und auf dem Computerbildschirm ausgegeben werden. Programmiert werden die *Sketches* in einer vereinfachten C-, beziehungsweise C++- Variante. In der Entwicklungsumgebung kommen deswegen bereits viele Libraries automatisch zur Anwendung. Für diese Arbeit wurde aus Kompatibilitätsgründen die Version 1.5.6-r2 der IDE verwendet.

3.2. Actuated Tangible User Interface Object - ACTO

Das ACTO- (Actuated Tangible User Interface Object) System [3] ist ein flexibles und modulares System, welches eine Interaktion zwischen Mensch und Computer ermöglicht. *Tangible User Interfaces* (TUIs) erlauben User-Input in verschiedenster Form. So können zum Beispiel Befehle über Buttons, Touch-Sensoren oder die Bewegung des TUIs an den Computer gesendet werden. Gleichzeitig kann das TUI aber auch Feedback verschiedenster Art geben. So können unter anderem LEDs oder Matrixanzeigen Informationen übermitteln. Die Modalitäten dieser Mensch-Maschine-Kommunikation können je nach technischer Ausgestaltung des TUIs sehr vielfältig sein.

Auf diesem Gebiet gab es bereits mehrere vielversprechende Projekte, die aber alle eher unflexibel und in der Benutzung aufwändig waren (siehe 2.1). Wenn ein Sensor, eine Eingabemodalität oder eine andere Funktionalität geändert werden sollte, war meist ein Redesign des TUIs notwendig. Für modular aufgebaute Systeme war es nicht möglich, einfach selbst Erweiterungen zu konzipieren. Mit dem ACTO wurde ein flexibles System geschaffen, bei dem verschiedene Komponenten unabhängig von einander ausgetauscht werden können. Durch den Open-Source-Ansatz ist es ohne großen Aufwand möglich, selbst Erweiterungen zu kreieren.

3.2.1. Basismodul

Als Steuereinheit aller Erweiterungen und Module, kommt eine vom Arduino Pro Mini inspirierte Plattform zum Einsatz (Abbildung 4). Um Platz zu sparen, wurde auf einen Reset-Button und USB-Chip verzichtet. Der Mikrocontroller wurde durch einen leistungsstärkeren Atmega328P ersetzt. Weiters wurde ein Nordic nRF24L01+ RF Chip [31], gemeinsam mit dem notwendigen $3.3V$ Spannungswandler und einer Antenne integriert. Der Funkchip kommuniziert direkt über den SPI-Bus mit dem Mikrocontroller. Insgesamt verfügt das Basismodul über 19 I/O Pins von denen 4 PWM-fähig sind und 8 als analoge Input-Pins benutzt werden können. Die Platine ist $36x36x6mm$ groß und wiegt $10g$. [3]

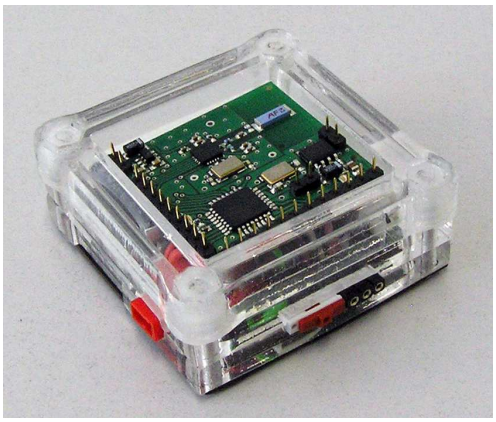


Abbildung 4: ACTO-Basismodul

3.2.2. Motor- und Tracking-Modul

Um das ACTO mit dem Computer bewegen zu können, wurde eine Motoreinheit (Abbildung 5) entwickelt. Diese orientiert sich am modularen Design und besteht aus 2 Motoren, 2 separaten Getrieben und 2 Rädern. Damit sind gerade Fahrten und Kurvenfahrten, sowie die Rotation am Stand möglich. Um sich seitwärts zu bewegen, muss sich das ACTO allerdings drehen, nach vorne bewegen und wieder drehen. Außerdem wurden die notwendigen elektronischen Bauteile, wie ein Spannungswandler und ein Stecker für die Stromversorgung, in das Modul integriert. Auf der Unterseite kann ein Marker für das Tracking durch eine transparente Platte angebracht werden. [3]

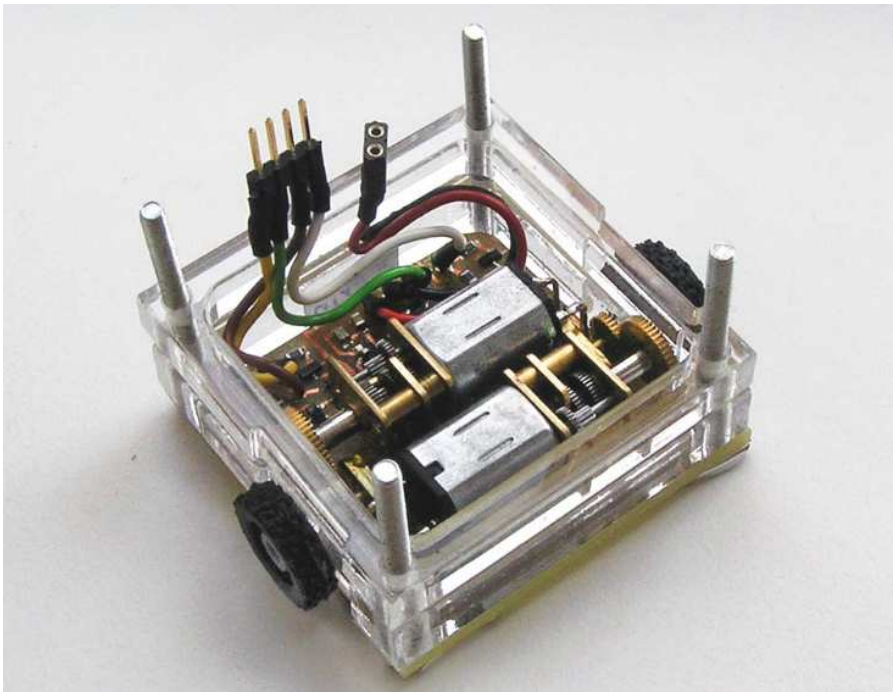
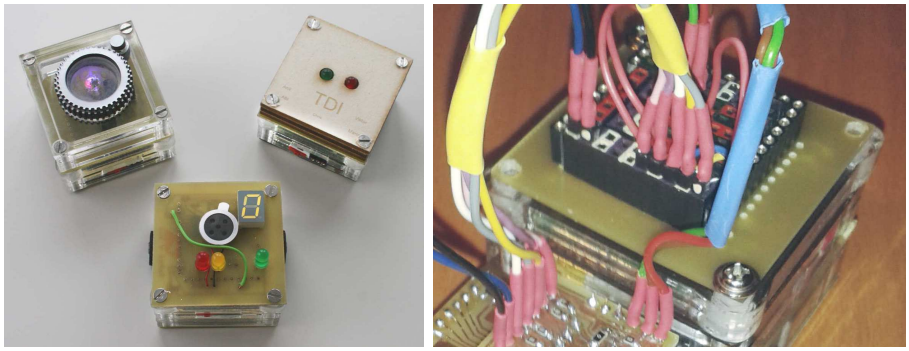


Abbildung 5: ACTO-Motoreinheit [3]

3.2.3. Erweiterungsmodule

Durch den 27-Pin-Sockel des Basismoduls und ein bereitgestelltes *Fritzing*-Template (siehe A.1), sollte es ähnlich einfach wie für das Arduino möglich sein, selbst verschiedene Erweiterungen (Abbildung 6a) zu designen. Das Erweiterungsmodul kann dabei alle Komponenten enthalten, die auch von einem Arduino gesteuert werden können. So können zum Beispiel Status-LEDs, Matrixanzeigen oder Buttons Anwendung finden. In dieser Arbeit wurde, zu Testzwecken für die Verkabelung, eine Breadboard-Extension benutzt (Abbildung 6b). Bei dieser befindet sich auf der Oberfläche ein in der Größe reduziertes Breadboard. Zusätzlich werden die verfügbaren Pins des ACTOs zu Steckerleisten durchgeleitet.



(a) (b)

Abbildung 6: (a) ACTOs mit verschiedenen Erweiterungen [3], (b) ACTO mit Breadboard-Erweiterung

3.2.4. Setup

Im bestehenden ACTO-Framework ist bereits ein optisches Tracking mittels Markern auf der Unterseite der ACTOs implementiert. Dabei werden die ACTOs von einer Kamera von unten durch eine Plexiglasscheibe getrackt (Abbildung 7). Das kann zum Beispiel durch die Frontkamera eines Android-Gerätes geschehen. Das Android-Gerät auf dem auch das Framework läuft, kommuniziert mit den einzelnen ACTOs und steuert diese. Die Kommunikation erfolgt dabei über ein, an den USB-Port angeschlossenes, Arduino mit einer Funkerweiterung. [31]

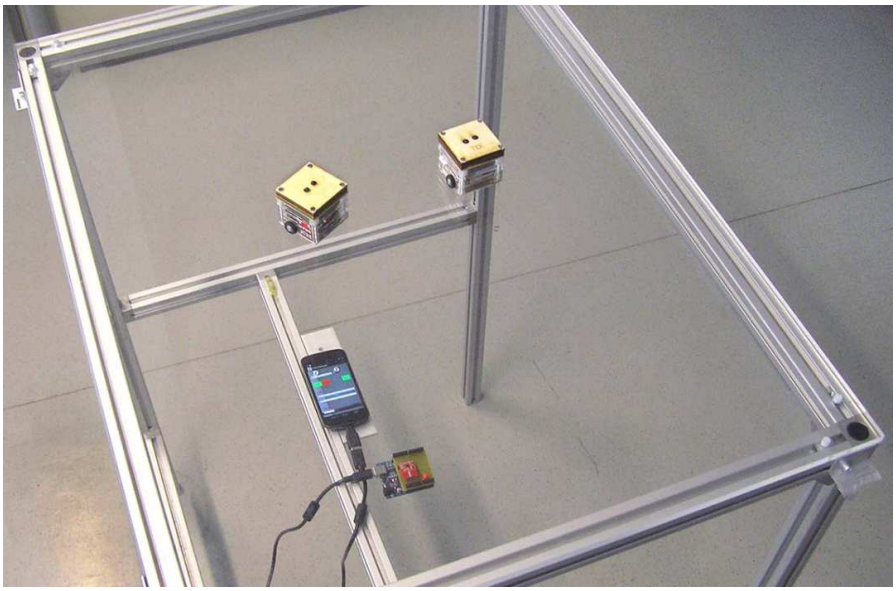


Abbildung 7: ACTO-Setup [3]

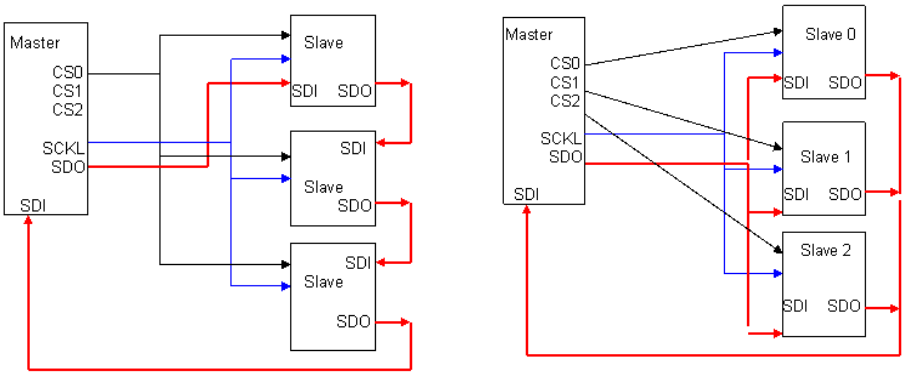
3.3. Serial Peripheral Interface - SPI

Serial Peripheral Interface (SPI) [32] wurde von Motorola entwickelt und dient vor allem zur Kommunikation von Hostprozessoren mit Peripheriebausteinen. Das Bus-System wurde aber nur „locker“ spezifiziert, weswegen wichtige Details für die Datenübertragung meist in den Datenblättern der Komponenten zu finden sind. So sind zum Beispiel die Betriebsspannung und Taktfrequenz nicht spezifiziert und unterscheiden sich je nach verwendeten Bauteilen. Generell stehen für die Kommunikation 2 Steuer- und 2 Datenleitungen zur Verfügung. Je nach Anwendung gibt es verschiedene Bezeichnungen. Gängige Bezeichnungen sind in Tabelle 1 angeführt:

Abkürzung	Bezeichnung
SCLK	Serial-Clock
SS	Slave-Select
MISO	Master-Input, Slave-Output
MOSI	Master-Output, Slave-Input

Tabelle 1: SPI-Pins

Die einzelnen Komponenten können sich in 2 Modi befinden - Master oder Slave. Master sendet über SCLK ein Clock-Signal und wählt über SS die richtige Slave-Komponente. Für das Slave-Device sind SCLK und SS Inputleitungen. Über MOSI werden Daten vom Master zum Slave übertragen, während über MISO Daten vom Slave zum Master gesendet werden. Für MISO und MOSI existieren auch die Beizeichnungen SDI (*Serial Data Input*) und SDO (*Serial Data Output*). Mit dieser Bezeichnung müsste aber Master-SDO mit Slave-SDI verbunden werden, anstatt MOSI mit MOSI. (Für die MISO-Verbindung gilt das gleiche.) Für SS ist auch CS (*Chip Select*) gebräuchlich. Im Allgemeinen gibt es, wie in Abbildung 8 dargestellt, 2 sinnvolle Arten mehrere SPI-Bausteine mit einem Master zu verbinden.



(a) (b)

Abbildung 8: SPI-Setup-Varianten: (a) kaskadierend, (b) sternförmig [32]

Im Setup kann es damit immer nur ein Master- und beliebig viele Slave-Devices geben. Bei einer sternförmigen Struktur muss für jedes Slave-Device eine Chip-Select-Leitung (CS/SS) vorhanden sein. Bei einem kaskadierenden Setup erscheinen die einzelnen Bausteine dem Master als ein Element, weswegen auch nur eine Chip-Select-Leitung existiert. Darüber hinaus gibt es Mischformen der beiden Setups und die Möglichkeit zwei Mikrocontroller zu verwenden.

Es gibt von Motorola keine offiziellen Spezifikationen. Deswegen finden sich die Details über den Takt und die gültigen Taktflanken in den Datenblättern der SPI-Bausteine. Unabhängig von Motorola haben sich allerdings 4 verschiedene SPI-Betriebsarten etabliert. Diese legen fest ob bei SCLK auf High oder Low und bei steigender oder fallender Taktflanke auf MOSI/MISO Daten übertragen werden. Die inoffiziellen Spezifikationen der SPI-Betriebsarten können Tabelle 2 und der dazugehörenden Erklärung zu CPHA und CPOL entnommen werden. Befehle werden, genau wie Datenwerte, über die seriellen Leitungen in Schieberegister geladen und von dort gelesen.

```
CPHA (Clock Phase) 0: Phase des Clocksignals null
CPHA (Clock Phase) 1: Phase des Clocksignals eins
CPOL (Clock Polarity) 0: Datenübernahme bei steigender Taktflanke
CPOL (Clock Polarity) 1: Datenübernahme bei fallender Taktflanke
```

SPI-Modus	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Tabelle 2: SPI-Modi

3.4. Speckle Sense

Um eine Bewegung messen zu können, erstellt ein auf dem Speckle-Prinzip basierender Sensor, wie beispielsweise ein optischer Maussensor, von einer beleuchteten Oberfläche bis zu 11750 mal pro Sekunde [4] ein Abbild. Aus dem Unterschied zwischen den Bildern wird die Verschiebung berechnet. Genauer gesagt formt sich ein Speckle-Pattern (Abbildung 10), wenn kohärentes Licht von einer nicht glatten Oberflächen diffus reflektiert wird. Jeder Pixel im optischen Sensor detektiert Beiträge von mehreren reflektierten Wellen. Die gemessene Intensität an jedem Pixel ist ein Resultat der unterschiedlich langen zurückgelegten Wege des Lichts und der dabei entstehenden Interferenz (Abbildung 9). Die Phasen der Wellen sind statistisch unabhängig (und zufällig), solange die Oberfläche hinreichend rau ist. Das so entstehende Bild zeichnet sich durch körnige, graue Strukturen mit hohem Kontrast aus. [33]

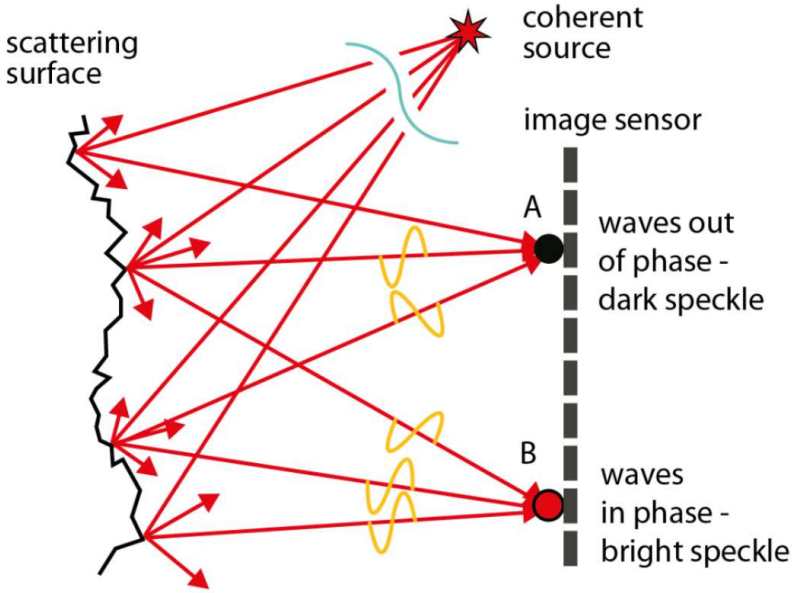


Abbildung 9: Speckle-Prinzip [33]

Es ist zwar nicht möglich Einfluss auf das damit entstehende Speckle-Pattern zu nehmen, allerdings lassen sich seine Features analysieren. So kann eine Verschiebung des Patterns in x- oder y-Richtung gemessen werden. Wird der Sensor in eine bestimmte Richtung bewegt, verschiebt sich das gemessene Pattern in die entgegengesetzte Richtung. [33]

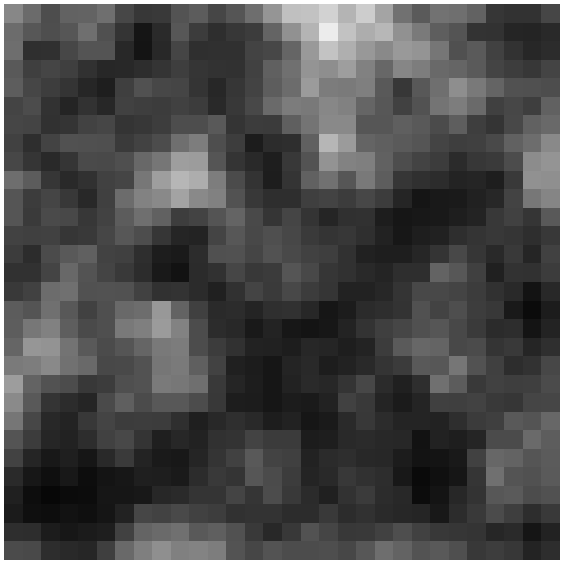


Abbildung 10: Speckle-Pattern

3.5. Mathematischer Hintergrund

Das System welches in dieser Arbeit entwickelt wird, genannt Speckle-Tracking-System, benutzt 2 optische Sensoren, deren Position und Ausrichtung zueinander bekannt sind. Anhand der Messwerte der Sensoren, wenn sie verschoben werden, wird mit Hilfe des in diesem Kapitel beschriebenen Algorithmus aus „Precise Dead-Reckoning for Mobile Robots Using Multiple Optical Mouse Sensors“ von Sekimori et.al. 2007 [16] die neue Position und Ausrichtung des Systems bestimmt.

Abbildung 11 zeigt den linken (O_L) und rechten (O_R) Sensor. Beide Sensoren befinden sich dabei in einem definierten Abstand (d_L und d_R) und einem bestimmten Winkel (ϕ_L und ϕ_R) zum Ursprung des ACTOS (O).

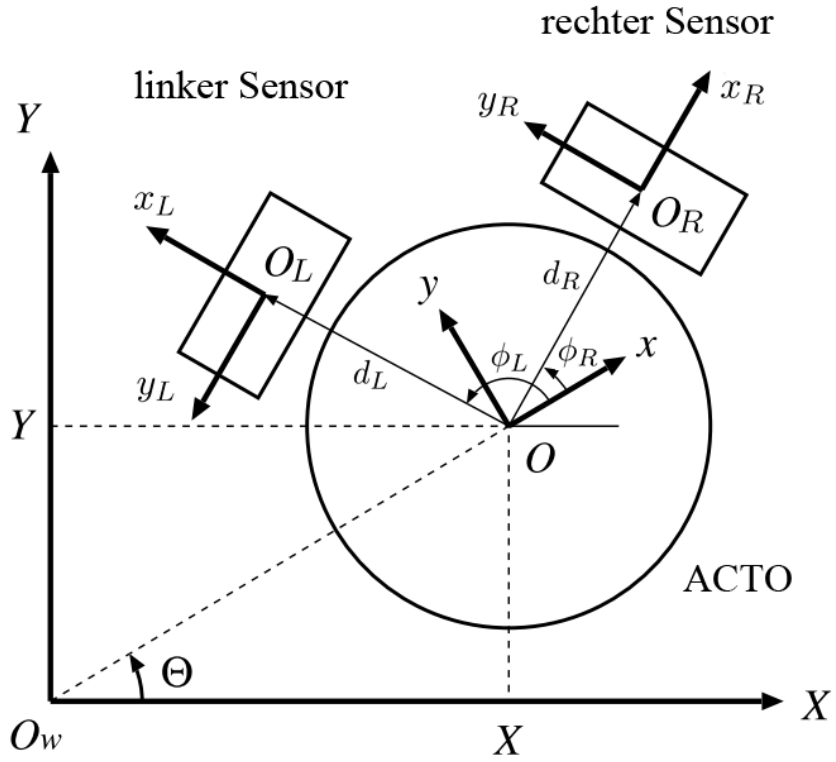


Abbildung 11: Schema des Tracking-Systems anhand von Sekimori et al. 2007 [16]

Wird das ACTO verschoben, registrieren das die Sensoren. Jeder Sensor misst dabei die Bewegung entlang seiner x- und y-Achse (x_L, y_L, x_R, y_R).

Diese Verschiebung wird regelmäßig (in etwa alle $25ms$ - siehe Kapitel 7.1.1), in der Form von 4 Δ-Werten ($\Delta x_L, \Delta y_L, \Delta x_R, \Delta y_R$), ausgelesen. Mit Hilfe der Δ-Werte kann die Verschiebung und Rotation des Gesamtsystems $u = \begin{bmatrix}\Delta x, & \Delta y, & \Delta\theta\end{bmatrix}^T$ errechnet werden. Mit dieser Verschiebung/Rotation und der Position/Rotation des Ausgangspunkts, wird anschließend die neue Position und Rotation errechnet.

$$u = \begin{bmatrix} \Delta x, & \Delta y, & \Delta\theta \end{bmatrix}^T$$

Ziel ist es dabei u (die Verschiebung und Rotation des Systems) zu errechnen. Damit können abschließend die neue Position und Ausrichtung bestimmt werden.

$$A = \begin{bmatrix} 1 & 0 & -d_R \sin\phi_R \\ 0 & 1 & d_R \sin\phi_R \\ 1 & 0 & -d_L \cos\phi_L \\ 0 & 1 & d_L \cos\phi_L \end{bmatrix}$$

Matrix A muss dabei nur einmal berechnet werden, solange sich die Ausrichtung und Entfernung der Sensoren zueinander nicht verändert.

$$a = \begin{bmatrix} \Delta x_R \cos\phi_R - \Delta y_R \sin\phi_R \\ \Delta x_R \sin\phi_R + \Delta y_R \cos\phi_R \\ \Delta x_L \cos\phi_L - \Delta y_L \sin\phi_L \\ \Delta x_L \sin\phi_L + \Delta y_L \cos\phi_L \end{bmatrix}$$

Da sich der Winkel der Sensoren zueinander nicht verändern sollte, ist es auch bei Matrix a möglich die trigonometrischen Berechnungen ($\cos\phi_R$...) vorweg zu nehmen.

$$u = A^- a$$

A^- steht für das Pseudoinverse der Matrix A und wird mit Matrix a multipliziert um u zu erhalten. Wo oben erwähnt bleiben Matrix A und damit auch A^- während des Trackings unverändert.

Wenn u und der Ausgangspunkt bekannt sind, kann schlussendlich die nächste Position errechnet werden:

$$\begin{bmatrix} X_t \\ Y_t \\ \Theta_t \end{bmatrix} = \begin{bmatrix} X_{t-1} + \Delta x \cos\Theta_{t-1} - \Delta y \sin\Theta_{t-1} \\ Y_{t-1} + \Delta x \sin\Theta_{t-1} + \Delta y \cos\Theta_{t-1} \\ \Theta_{t-1} + \Delta\theta \end{bmatrix}$$

Alternativ kann u, wenn man eine parallele Ausrichtung der Sensoren voraussetzt wie sie in Abbildung 12 dargestellt ist, auch einfacher berechnet werden. Das kann beispielsweise hilfreich sein um unnötige Berechnungen auf dem Mikrocontroller zu vermeiden. Zusätzlich wird der Code lesbarer und verständlicher. Die Anordnung der Sensoren in Abbildung 12 entspricht der Anordnung der Sensoren im Prototypen. Um den Code möglichst allgemein zu halten, wurde jedoch trotzdem der Algorithmus von Sekimori et al. 2007 [16] (siehe oben) verwendet.

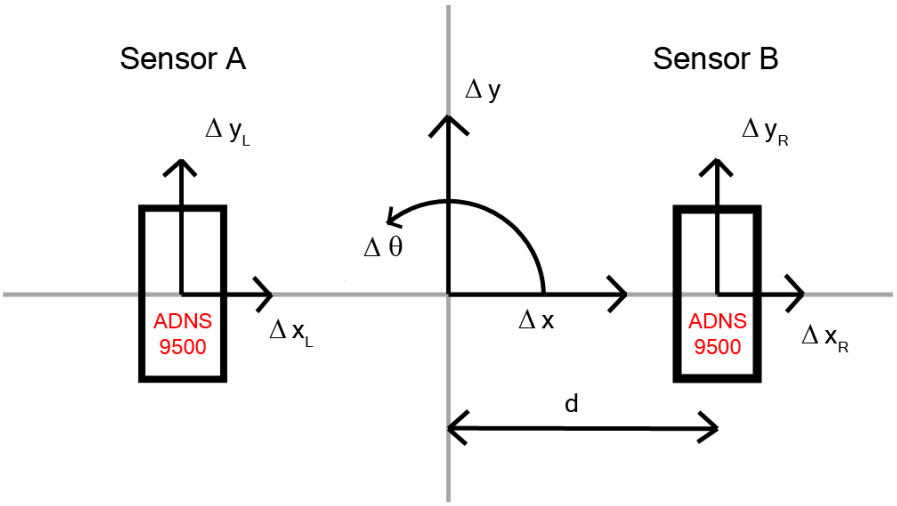


Abbildung 12: Anordnung der Sensoren beim Tracking für das ACTO-Projekt

$$u = \begin{bmatrix} \Delta x, & \Delta y, & \Delta\theta \end{bmatrix}^T$$

$$\Delta x = \Delta x_L = \Delta x_R$$
$$\Delta y = \frac{\Delta y_L + \Delta y_R}{2}$$
$$\Delta\theta = \frac{\Delta y_L - \Delta y_R}{2d}$$

Sind die Sensoren parallel entlang der x-Achse ausgerichtet, müssen die gemessenen Δx_L und Δx_R Werte gleich sein und der Verschiebung entlang der x-Achse des Systems (Δx) entsprechen. Die Verschiebung entlang der y-Achse des Systems (Δy) kann aus dem Durchschnitt der Verschiebung der einzelnen Sensoren entlang ihrer y-Achse (Δy_L und Δy_R) berechnet werden. Der Winkel $\Delta\theta$ wird aus der Differenz der beiden Messungen entlang der y-Achse (Δy_L und Δy_R), im Verhältnis zum Abstand der Sensoren ($2d$), errechnet.

4. Designprozess

In diesem Kapitel werden die Anforderungen an die Konstruktion und der Designprozess, von den ersten Funktionstests bis zum finalen Prototyp, beschrieben. Es werden jeweils die Überlegungen ausgeführt die hinter bestimmten Designentscheidungen und Vorgehensweisen stehen. Die im Zuge dieser Evolution auftretenden Probleme, sowie deren Identifizierung und Lösung werden gleichermaßen beleuchtet. Dieses Kapitel beschäftigt sich mit den Designentscheidungen und dem Designprozess als solchem. Die Konstruktion und die Beschreibung der konkreten Hardware befindet sich im Kapitel 5. Auf die benutzten Werkzeuge und Verfahren wird in Anhang A eingegangen.

4.1. Anforderungen

Um ein praxistaugliches Tracking-System für das ACTO zu entwickeln, sind einige Anforderungen zu berücksichtigen, welche hier angeführt werden:

4.1.1. Platzsparend

Der Motor und das Batteriepaket des ACTOs befinden sich direkt in der Hauptsektion des Gehäuses. Dabei liegen die beiden Motoren auf der Unterseite und der Akku befindet sich direkt darüber. Die eigentliche Platine liegt lose auf dem Akku, wird aber seitlich und von oben durch das Gehäuse fixiert. Um die einwandfreie Funktion des Tracking-Systems zu gewährleisten, müssen die 2 Sensoren möglichst weit von einander entfernt, auf der Unterseite des ACTOs platziert werden. Da das ACTO mit dem Gehäuse direkt auf dem Untergrund aufsitzt und nur ein wenig durch die beiden Räder gehoben wird, müssen die beiden Sensorchips in die Bodenplatte integriert werden. Ursprünglich wurde für die praktische Umsetzung eine Aussparung in der Bodenplatte vorgesehen. Diese Vorgehensweise erwies sich allerdings, aufgrund des mangelnden Platzes in der Motoreinheit, als nicht zielführend. Stattdessen werden die Sensoren auf der Außenseite des Gehäuses befestigt. Wegen der geringen Größe des ACTOs müssen die beiden Sensoren außerdem, obwohl ein großer Abstand wünschenswert wäre, sehr nahe beisammen liegen.

4.1.2. Robustes Tracking

Damit das Tracking-System zuverlässig arbeitet, müssen sich die beiden Sensoren in einem definierten, sowie gleichbleibenden Abstand und Winkel zueinander befinden. Weicht die tatsächliche Position der Sensoren zueinander von der theoretischen ab, wirkt sich das negativ auf das Tracking aus. Umso näher die Sensorchips beieinander liegen, umso gravierender wirken sich auch nur kleine Abweichungen aus. Da im ACTO-Gehäuse nur begrenzt Platz vorhanden ist, müssen die Chips einerseits nahe zusammen liegen und andererseits sehr präzise zueinander ausgerichtet sein. Um kleine Abweichungen zu kompensieren, könnte die reale Position der Sensoren zueinander im Nachhinein aus einer definierten Bewegung des Systems und den dazugehörigen Tracking-Daten berechnet werden. Siehe dazu Kapitel 8.3.

4.1.3. Widerstandsfähig

Da das Tracking-System für das ACTO entwickelt wird, welches als TUI durchaus moderaten äußeren Belastungen ausgesetzt wird, ist ein möglichst guter Schutz gegen mechanische Einwirkungen notwendig. Dieser kann einerseits erreicht werden, indem sich möglichst viele Bauteile im Gehäuse selbst befinden. Andererseits ist auch ein zusätzlicher mechanischer Schutz für Bauteile außerhalb des Gehäuses denkbar.

4.1.4. Günstig

Das gesamte Tracking-System sollte trotz der Anforderungen an die Präzision möglichst günstig in der Herstellung bleiben. Weiters sollte die Konstruktion mit einfachen Mitteln zu bewerkstelligen sein. Daher wird so weit wie möglich auf günstige und einfach zu beschaffende Bauteile zurückgegriffen.

4.2. Wahl des Sensors

Als Sensor wurde aus verschiedenen Gründen der ADNS 9500 [4] von *Pixart Imaging Inc.* gewählt. Zu einen ist er in Gaming-Mäusen verbreitet und damit einfach und günstig zu beschaffen. Außerdem sind Code-Beispiele für die SPI-Integration und die Firmware verfügbar. Grundsätzlich würden sich auch spezialisiertere Sensoren eignen, wie zum Beispiel der kleinere ADNS 3530 [34] oder der ADNS 8020 [35], welcher auch auf Glas funktioniert. Allerdings wurde dem ADNS 9500 aufgrund der einfacheren Handhabung der Vorzug gegeben.

Ursprünglich wurde der ADNS 9500 als Gaming-Sensor konzipiert. So wird der Chip unter anderem in der *SteelSeries Sensei RAW Rubberized* Laser-Gaming-Maus verbaut. [36] Des Weiteren ist die benötigte Laserlichtquelle bereits integriert. Das ermöglicht grundsätzlich eine platzsparende Konstruktion. Der ADNS 9500 kann mit einer Spannung von entweder $3V$ oder $5V$ betrieben werden. Je nach gewünschter Spannung ändert sich die Auswahl der anderen benötigten Komponenten, wie der Widerstände und Kondensatoren. Abbildung 13 zeigt den Schaltplan laut Datenblatt für den Betrieb mit $5V$ Spannung, der als Basis für die Prototypen verwendet wurde. Die Framerate beträgt bis zu 11750 Bilder pro Sekunde, während die Auflösung bis $5737.5cpi$ wählbar ist. Für die Anwendung als Tracking-System sollten diese Werte in jedem Fall ausreichen. Zusätzlich können die 900 Pixel (30x30) des aktuellen Frames ausgelesen werden. Da danach ein Hardware-Reset durchgeführt und die Firmware neu geladen werden muss, eignet sich diese Funktion allerdings nur bedingt für den laufenden Betrieb.

Weiters wird für die einwandfreie Funktion zusätzlich eine Linse (ADNS-6190-002) [27] benötigt.

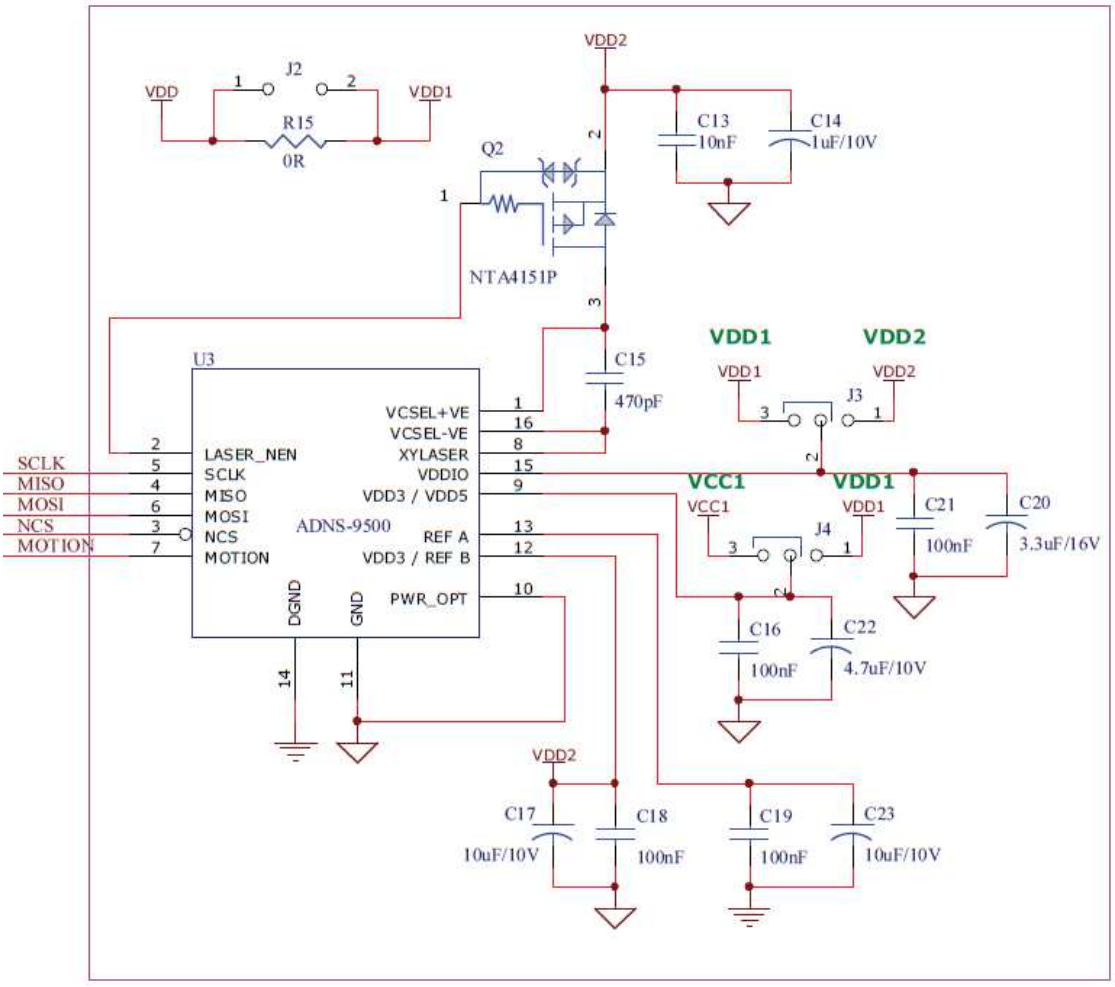


Abbildung 13: Schaltplan für ADNS 9500 ($5V$-Modus) laut Datenblatt [4]

4.3. Funktionstests

Um zu klären, ob sich der gewählte Sensor grundsätzlich für das Tracking eignet und welche Bauteile für einen Tracking-System-Prototyp notwendig sind, wurden Funktionstests durchgeführt. Des Weiteren sollte die SPI-Anbindung getestet und damit sichergestellt werden, dass die für das Tracking notwendigen Daten auslesbar sind. Um das modifizierte Design des ACTOs als Fehlerquelle auszuschließen, wurde für diesen Test stattdessen ein Arduino UNO als Mikrocontroller verwendet. Bei späteren Prototypen wurde das Arduino UNO durch ein ACTO ersetzt. Abgesehen vom eigentlichen Sensorchip ADNS 9500 werden laut Datenblatt [4] und wie in Abbildung 13 dargestellt zusätzliche Bauteile benötigt und in Tabelle 3 angeführt.

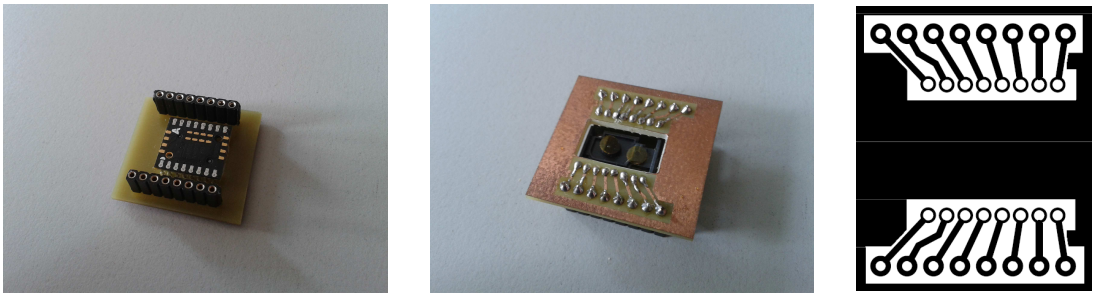
Anzahl	Typ	Bezeichnung
1	Kondensator	non-polarized 470pF
1	Kondensator	non-polarized 10mf
4	Kondensator	non-polarized 100nf
1	Kondensator	polarized 1uF/10V
1	Kondensator	polarized 3.3uF/16V
1	Kondensator	polarized 4.7uF/10V
2	Kondensator	polarized 10uF/10V
1	P-Channel-Mosfet	

Tabelle 3: Laut Schaltplan (Abbildung 13) benötigte Bauteile

Die Kondensatoren werden benötigt um Spannungsschwankungen auszugleichen. Der Mosfet steuert und schützt den Laser. Für die 6 nicht-polaren Kondensatoren wurde Keramik-Multilayer-Kondensatoren benutzt. Tantal-Kondensatoren fanden als polare Kondensatoren Verwendung.

Um Material zu sparen, wurde hier auf eine vollständige Implementierung auf einer Leiterplatte verzichtet. Stattdessen wurde ein Breadboard benutzt, um gegebenenfalls die Bauteile austauschen und auf Designfehler reagieren zu können. Somit wurden für eine Durchsteckmontage geeignete Kondensatoren verwendet. Diese wurden einfach auf das Breadboard aufgesteckt.

Der Pin Abstand des ADNS 9500 stimmt mit keinem gängigen Breadboardformat überein, weshalb der Chip nicht direkt auf das Breadboard gesteckt werden kann. Deswegen wurde zu erst versucht, direkt an die Pins des Chips Kabel anzulöten und diese dann mit dem Breadboard zu verbinden. Aufgrund des geringen Abstandes der Pins kam ein direktes Anlöten von Kabeln nicht in Frage. Daher wurde für die 16 Pins des ADNS 9500 eine Leiterplatte als Adapter designt. Mittels Durchsteckmontage war es möglich den Chip auf der Platine zu verlöten und die Pins mit Buchsensteckern zu verbinden. Da sich Laser und Sensor auf der Unterseite befinden, musste in der Mitte der Platine eine Aussparung geschaffen werden. Abbildung 14 zeigt den Adapter mit dem ADNS 9500 von oben (a) und von unten (b) sowie die Ätz-Vorlage (c).



(a) (b) (c)

Abbildung 14: ADNS 9500-Adapter: (a) Oberseite, (b) Unterseite, (c) Ätzvorlage (c)

Da kein passender Mosfet für das Breadboard verfügbar war, wurde ähnlich wie für den ADNS 9500, eine eigene kleine Platine mit einem aufgelöteten SMD-Mosfet verwendet. Durch Löcher in der Platine wurden Drahtstifte geführt und angelötet, die schließlich in das Breadboard gesteckt werden konnten.

Anschließend wurden die Bauteile auf dem Breadboard gruppiert und das Arduino UNO sowie der Adapter angeschlossen (Abbildung 15). Dabei wurden das Arduino und der ADNS 9500 auf einer passenden Holzplatte fixiert, welche mit einem Loch für den Laser und den Sensor versehen wurde.

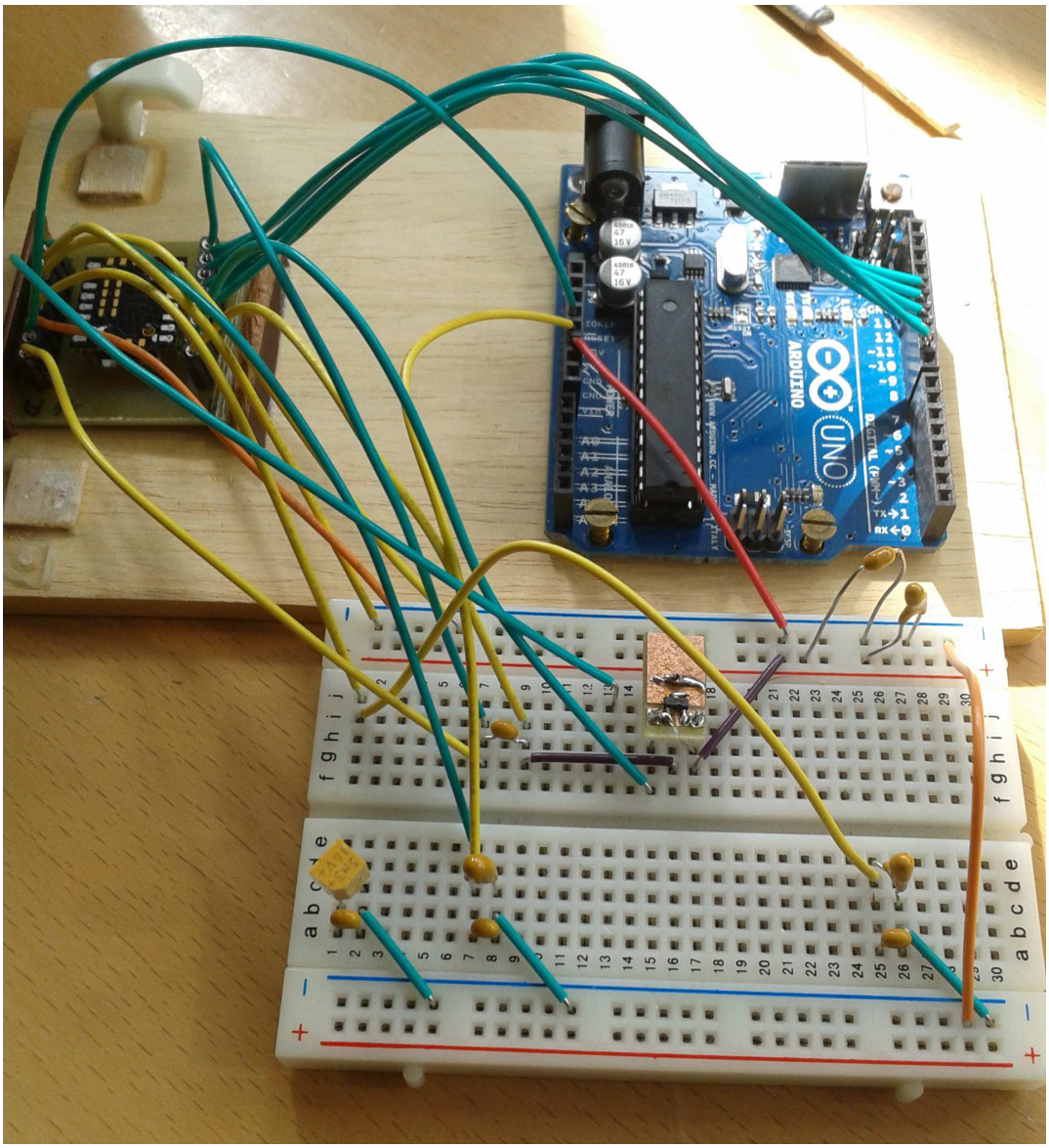


Abbildung 15: ADNS 9500-Breadboard-Prototyp

Während der Power-Up-Sequenz des Chips wird zuerst die Versorgungsspannung angelegt und der SPI-Port zurückgesetzt. Anschließend werden mehrere Register geschrieben und gelesen. Am Schluss wird die Firmware geladen und der Laser aktiviert. Die genauen Spezifikationen zum Power-Up, den Schreib- und Lesevorgängen und dem SPI-Modus finden sich im Datenblatt [4] und in Kapitel 6.2.1. Nachdem die SPI-Kommunikation hergestellt und der Chip gestartet wurden, ist es möglich die 4 Register (für die x- und y-Achse, jeweils ein High- und ein Low-Byte) für die Bewegung auszulesen.

Um aus den Registerwerten eine Distanz in mm zu erhalten, ist eine kurze Umrechnung notwendig:

$$x = \frac{Register\Delta x}{DotsperInch} * \frac{mm}{Inch}$$

Auf die Implementierung des Auslesens und der Umrechnung wird in Kapitel 6.2.1 genauer eingegangen.

Diese einfachen Funktionstests zeigten die grundsätzliche Eignung des ADNS 9500 für ein Tracking-System. Die Schaltung war funktionsfähig und sowohl die SPI-Kommunikation als auch die Umrechnung der Registerwerte waren möglich. Auf Basis dieser vielversprechenden Ergebnisse wurde ein zweiter Prototyp geplant.

4.4. Zwei Sensoren und mehrere SPI-Devices

Um den zurückgelegten Weg des Tracking-Systems berechnen zu können, sind mindestens 2 Maussensoren notwendig. Zusätzlich muss die Position und Ausrichtung dieser Sensoren zueinander eindeutig definiert und bekannt sein. In diesem Schritt sollte ein erstes funktionsfähiges Tracking-System konstruiert werden. Ziel war es, eine Konstruktion zu entwerfen, die einerseits gut getestet werden kann, andererseits aber bereits für den Einsatz mit einem ACTO geeignet ist. Die äußere Form wurde hauptsächlich durch diese Überlegungen bestimmt.

Zumindest die Linsen [27] und Sensoren [4] müssen sich zwangsläufig unten im Roboter, knapp über dem Untergrund befinden. Eine weitere Anforderung ist, die beiden Sensoren möglichst weit von einander entfernt anzuordnen, um die Auswirkungen geringer möglicher Ungenauigkeiten beim Einbau zu minimieren. Leider nimmt die Motoreinheit des Bewegungsmoduls derzeit die gesamte Bodenfläche des ACTOs ein, was die direkte Integration in diese erschwert. Zusätzlich müssten die beiden Sensorlinsen, bedingt durch den begrenzten Platz und die Außenwand des ACTOs, sehr nahe beisammen liegen. Unter diesen Gesichtspunkten wurde eine Unterbringung des Systems im ACTO-Gehäuse verworfen. Stattdessen wurde eine Konstruktion designt, bei der sich beide Sensoren und die notwendigen elektronischen Bauteile ausschließlich außerhalb des ACTOs befinden und somit innen keinen zusätzlichen Platz einnehmen.

Das System besteht aus 2 miteinander verbundenen, jeweils einseitigen PCB-Leiterplatten, wie in Abbildung 17a und 17b ersichtlich. Abbildung 16 zeigt den entsprechenden Schaltplan.

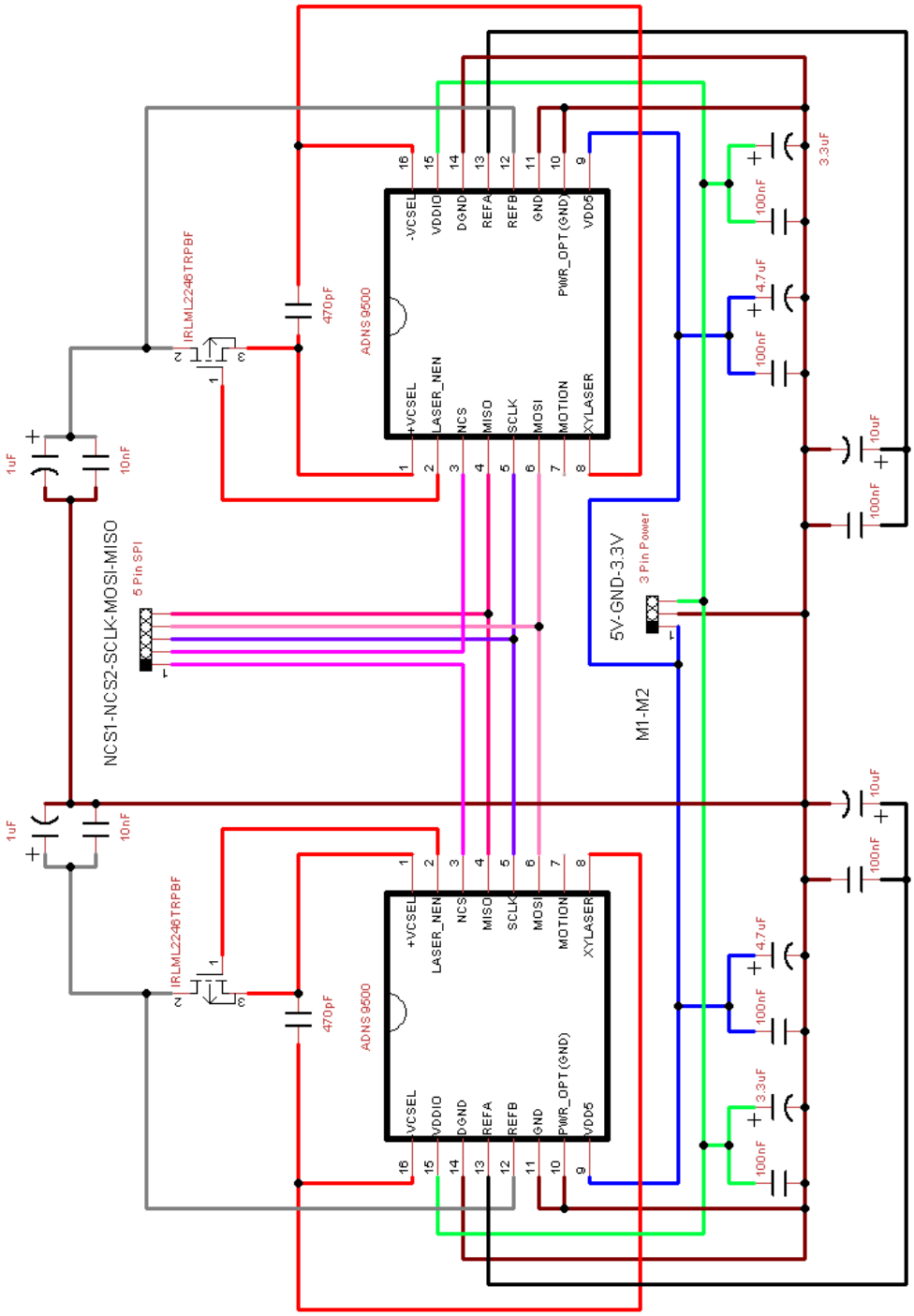
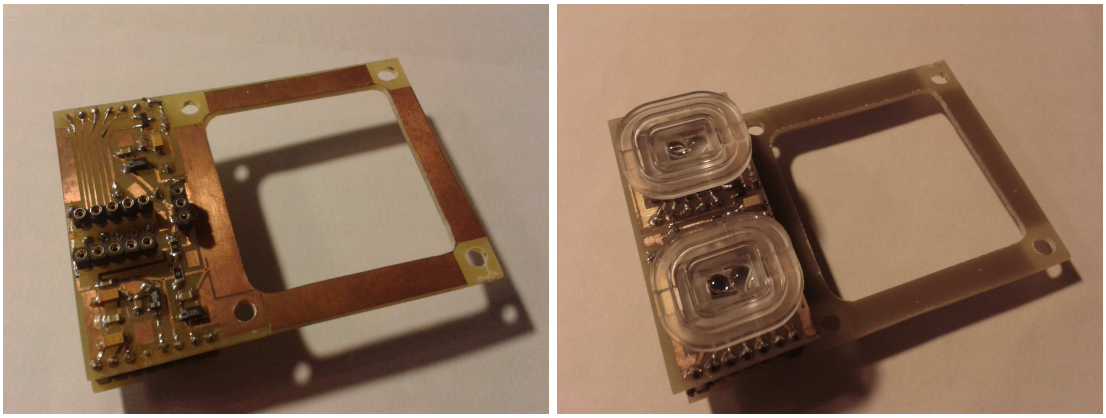


Abbildung 16: Schaltplan des ersten Prototyps mit zwei SPI Devices

Auf der unteren Ebene sind die beiden Sensoren und wenige andere Bauteile verbaut. Auf der oberen Ebene befindet sich der größte Teil der restlichen elektronischen Komponenten. Für diesen Prototyp wurden Bauteile im SMD-Formfaktor verwendet, da diese einfacher auf einer Leiterplatte anzubringen und kleiner sind. Die obere Leiterplatte ist außerdem wesentlich größer, sodass auf einer Seite eine Halterung mit den entsprechenden Bohrlöchern für eine Integration mit dem ACTO entsteht.



(a) (b)

Abbildung 17: ADNS 9500-Prototyp 2: (a) Oberseite, (b) Unterseite

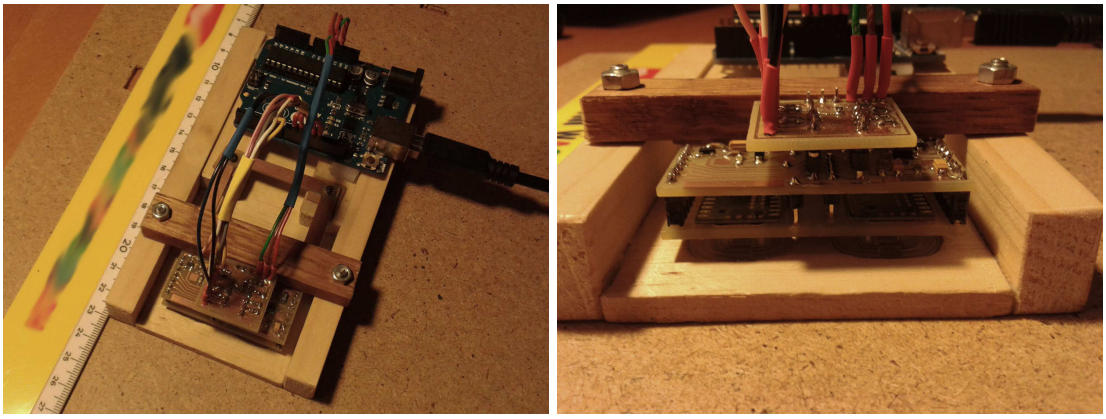
Als Anschlüsse wurden die Stromversorgung mit Ground, $3.3V$ und $5.5V$, sowie 5 SPI-Leitungen (SCLK, MISO, MOSI und 2 mal Chip-Select) und 2 Motion-Pins vorbereitet. Die Abbildungen 17a und 17b zeigen eine bereits etwas fortgeschrittenere Variante, in welcher die Stecker von SCLK, MISO und MOSI zur einfacheren Fehlersuche (siehe 4.5) nicht zusammengelegt sind. Beim ersten Prototypen standen in der Realität somit nur 2 Steckerleisten mit einem zusätzlichen Pin, statt drei Steckerleisten zur Verfügung.

Um beide Sensoren über den SPI-Bus steuern zu können, sind 2 Chip-Select-Leitungen und damit 2 Variablen in der SPI-Implementierung notwendig. Deswegen mussten mehrere Methoden im Standard-Arduino-SPI überschrieben werden (siehe Kapitel 6.2.1).

Im Zuge der Inbetriebnahme stellte sich heraus, dass das Arduino die SPI-Pins mit $5V$ Spannung versorgt, was aber weit außerhalb der Spezifikationen der Sensoren liegt. Zur Fehlereingrenzung wurde die $3.3V$ Leitung des Sensors getrennt, die unter anderem auch als Schutzschaltung für eine zu hohe Eingangsspannung fungiert. Dies führte zu irreparablen Defekten durch Überspannung an beiden Sensoren. Des Weiteren stellte sich das Design mit nur einer SPI-Steckerreihe für Testzwecke als zu unflexibel heraus. Obwohl in dieser Projektphase bereits die Integration mit dem ACTO geplant war, musste das Design wie im folgende Kapitel dargelegt, ein weiteres Mal überarbeitet werden.

4.5. Spannungsteiler und modularer Aufbau

Um auftretende Probleme besser eingrenzen und die Sensoren einzeln testen zu können, wurden die SPI-Pins zu 2 separaten Steckerleisten geführt. Zusätzlich musste die SPI-Spannung reduziert werden. Um die Schaltung nicht unnötig komplex zu machen, wurde statt eines Levelshifters nur ein Spannungsteiler mit Widerständen konstruiert. Auf das Anheben der Ausgangsspannung des MISO-Pins wurde vorerst verzichtet, da keine Schäden am Arduino durch Unterspannung zu erwarten waren und gleichzeitig die Spannung hoch genug erschien, um vom Arduino UNO als Taktflanke erkannt zu werden. Abbildung 18a zeigt den überarbeiteten Aufbau. Auf Abbildung 18b ist der Spannungsteiler gut zu erkennen.



(a) (b)

Abbildung 18: (a) Prototyp und Arduino UNO, (b) Spannungsteiler

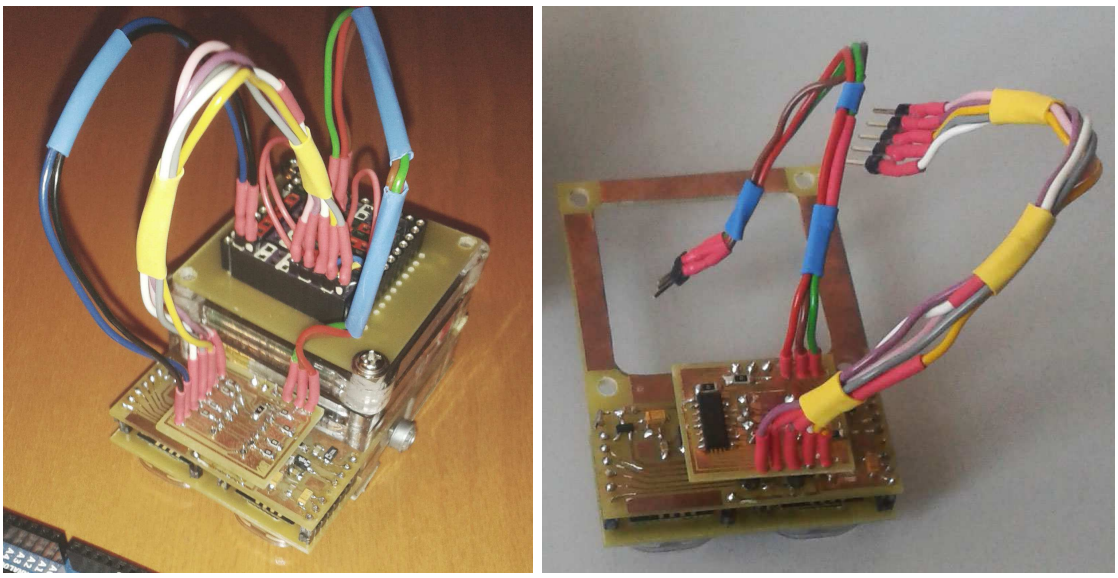
Für den Spannungsteiler wurden zu erst jeweils 4 Stück $15kOhm$ und $10kOhm$ Widerstände für die SCLK-, MOSI- und 2 NCS-Leitungen benutzt. Die Widerstände wurden auf einer Leiterplatte angelötet, welche mittels Steckerleisten einfach auf den neuen und flexibleren Prototyp aufgesteckt werden konnte. Auf der Spannungsteilerplatine wurden die SPI-Pins der beiden Sensoren zusammengeführt und die Spannung reduziert. Anschließend wurden Kabel zum Arduino geführt. Da die Taktflanken im Oszilloskop verzerrt und undeutlich erschienen, wurden die Widerstände auf $1.5kOhm$ beziehungsweise $1kOhm$ reduziert. Mit dieser Konstruktion konnten schlussendlich die Register beider Sensoren ausgelesen und geschrieben werden. Dies war der erste lauffähige Prototyp.

4.6. Verwendung des ACTOs und der Levelshifter

Der nächste logische Schritt war die Integration mit dem ACTO-System (Abbildung 19a). Der Formfaktor des Tracking-Systems war bereits seit dem ersten Prototyp für die Montage an einem ACTO ausgelegt. Daher wurde die Halterung einfach zwischen zwei Plexiglasringen des ACTOs fixiert. Damit befanden sich die Linsen ca. $1.5mm$ über dem

Boden, was innerhalb der Toleranz laut Datenblatt liegt. [4] Um Platz zu sparen, sind die Anschlüsse am ACTO nicht als Steckerbuchsenleisten sondern als Pins ausgeführt. Zuerst wurde die Breadboard-Extension (siehe 3.2.3) benutzt, um das Tracking-System anzuschließen.

Der Anschluss an das ACTO erwies sich als wesentlich aufwändiger, als an ein Arduino UNO. Durch die Menge an Metallbauteilen über dem Funkchip, kam es sowohl zu Funkübertragungsfehlern, als auch zu SPI-Kommunikationsfehlern mit den Sensoren. Deswegen wurde die Breadboard-Extension durch eine eigene Extension, mit möglichst wenig leitenden Bauteilen und Steckverbindungen, ersetzt. Direkt über dem Funkchip wurde eine Aussparung in der Leiterplatte vorgesehen, um den Funk möglichst wenig zu stören. Außerdem wurde der Spannungsteileraufsatz doch durch einen Aufsatz mit 2 Levelshiftern ersetzt, um unregelmäßig auftretenden Problemen in der SPI-Kommunikation zwischen ACTO und Sensoren entgegenzuwirken (erkennbar in Abbildung 19b). Schlussendlich musste das SPI ein weiteres Mal angepasst werden, da der Funkchip und die Sensoren unterschiedliche SPI-Modi verwenden (siehe Tabelle 2 in Abschnitt 3.3). Deswegen wurden die Methoden dahingehend angepasst, dass vor jedem Schreib- oder Lesevorgang eine Änderung des SPI-Modus stattfindet.



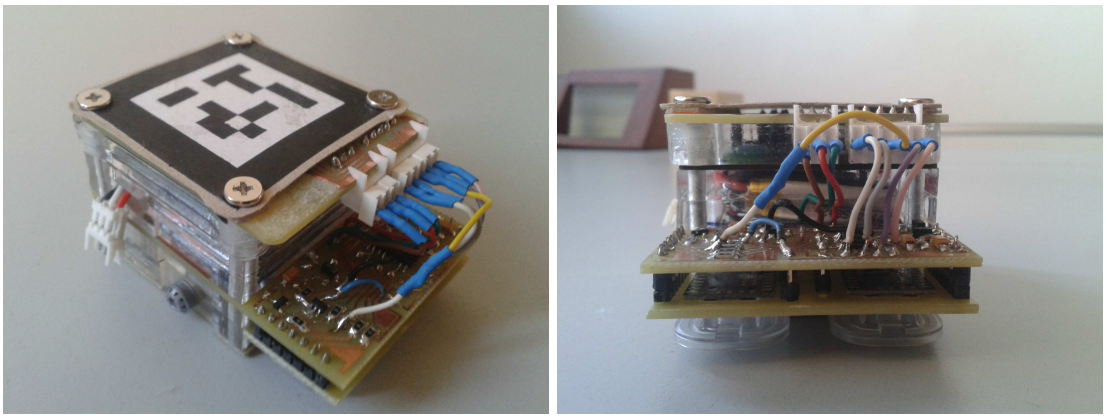
(a) (b)

Abbildung 19: (a) ACTO-Integration mit Spannungsteiler, (b) Tracking-Erweiterung mit Levelshifter

Mit diesen Anpassungen wurde sowohl die SPI-Kommunikation zwischen den einzelnen Bauteilen, als auch die Funkverbindungen zwischen ACTO und Basis möglich. Sporadisch auftretende Fehler in der Funkkommunikation blieben allerdings bestehen.

4.7. Finaler Prototyp

Nachdem die Funktionsfähigkeit der Schaltung mit dem ACTO demonstriert worden war, musste das Tracking-System für einen produktiven Einsatz in erster Linie verkleinert werden (Abbildung 20a und 20b). Im Zuge des notwendigen Redesigns wurde darauf geachtet, einerseits möglichst platzsparend zu arbeiten und andererseits auf der Oberseite des ACTOs eine glatte Fläche für die Montage eines Markers zu erhalten. Das alte System mit dem Marker auf der Unterseite konnte nicht weiterverwendet werden, da die ADNS 9500-Sensoren nicht auf Glas, beziehungsweise Plexiglas, arbeiten. Der Marker ist notwendig, da sonst der sich summierende Fehler des relativen Trackings nicht ausgeglichen werden kann.



(a) (b)

Abbildung 20: Finaler Prototyp auf ACTO: (a) von oben mit Marker, (b) von vorne

Für den finalen Prototyp wurde der MOSI-Levelshifter auf der Hauptplatine integriert, jedoch der MISO-Levelshifter weggelassen und eine komplett neue und flache Steckerplatine entworfen. Abbildung 21 zeigt den vollständigen Schaltplan. Die entsprechenden Ätzvorlagen befinden sich im Anhang B und geben einen Eindruck wie die verschiedenen Bauteile angeordnet wurden. In der Praxis wurde, wie erwähnt, der in der Vorlage vorgesehen Levelshifter von $3.3V$ auf $5V$ für die MISO-Leitung nicht verwendet. Stattdessen wurde die MISO-Leitung direkt verbunden. In einem finalen Fertigungsschritt der Platine wurden unbeabsichtigt die $3.3V$ und die $5V$ Anschlüsse für die Levelshifter vertauscht. Zur Lösung dieses Problems wurden nachträglich kurze Kabel direkt als Überbrückung angelötet.

Bei der eingesetzten Funktechnologie können Übertragungsfehler naturgemäß nie vollständig ausgeschlossen werden. Um diese so weit wie möglich zu reduzieren, wurde über dem Funkchip die Steckerplatine ausgeschnitten, wodurch Fehler nur noch sehr selten auftreten und damit kein Problem darstellen.

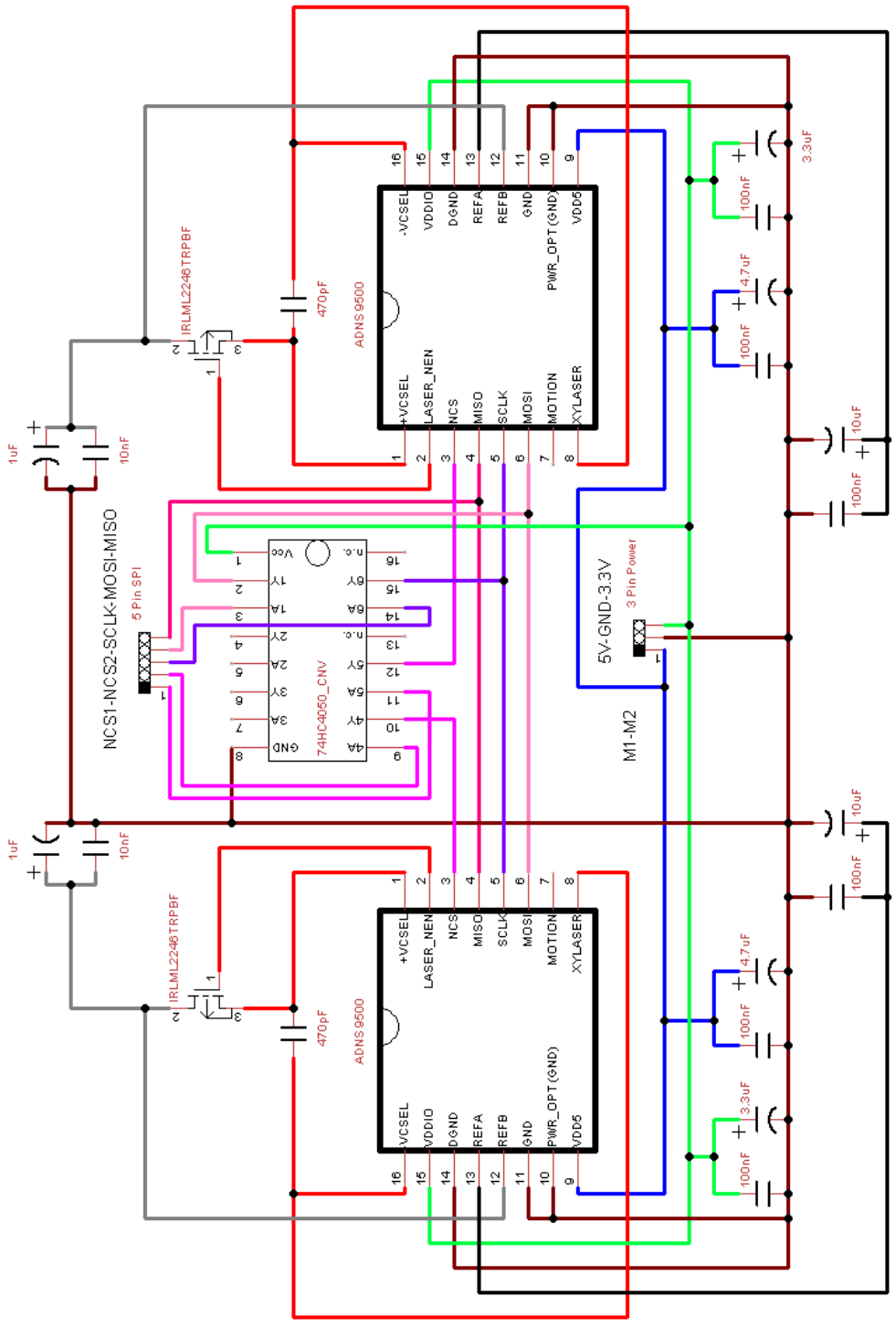


Abbildung 21: Schaltplan des finalen Prototyps

5. Konstruktion

Es wurden im Verlauf der Arbeit mehrere Prototypen entworfen und gebaut. Das Tracking-System durchlief so eine Evolution, die am Ende zu einem funktionierenden ACTO-Modul führte. Das Kapitel Konstruktion beschäftigt sich mit den handwerklichen Aspekten der Herstellung der Prototypen und Hilfsmitteln für die Tests. Da beim relativen Tracking mit Maussensoren, für das zusätzliche absolute Tracking, ein Marker auf dem ACTO von oben gefilmt wird, muss mittig über der Oberfläche eine Kamera positioniert werden. Der Entwurf für die dafür notwendige Kamerahalterung wird hier ebenfalls beschrieben.

5.1. Konstruktion der Prototypen

In diesem Abschnitt wird die handwerkliche Seite der Konstruktion des Tracking-Systems behandelt. Der Designprozess und die Überlegungen dahinter finden sich in Kapitel 4 Designprozess.

5.1.1. Herstellung der Platinen

Leiterplatten stellen die notwendige physische Fixierung und elektronische Verbindung der Bauelemente dar. Eine Platine besteht aus einem nicht leitenden Trägermaterial und einer darauf liegenden Kupferschicht. Auf dieser Kupferschicht befindet sich ein Schutzlack. Die Herstellung der Leiterbahnen auf der Platine kann auf verschiedene Arten erfolgen. [37] An dieser Stelle wird die für das Projekt benutzte Methode beschrieben.

Zuerst muss die Schaltung mit allen Bauteilen und den dazugehörigen Leiterbahnen entworfen werden. Dazu stehen verschiedene Programme wie *Eagle* [38] oder *Fritzing* [39] zur Auswahl. Theoretisch reicht für einfache Layouts auch ein normales Zeichenprogramm wie *CorelDraw* [40]. Für den Entwurf des Tracking-Systems wurde wegen der intuitiven Bedienung *Fritzing* (siehe A.1) benutzt. Für jede Ebene (beziehungsweise für die Spannungsteiler oder Levelshifter), wurde eine eigene Platine entworfen. Dabei wurde hauptsächlich in der Leiterplattenansicht gearbeitet. Grundsätzlich wurden alle Bauteile aus den Vorlagen in *Fritzing* im SMD-Format eingefügt. Nur für die beiden optischen Sensoren wurde eine eigene Vorlage mit 16 separat platzierten Kupferringen, für die Durchsteckmontage der Pins, angelegt. Beim Entwurf muss darauf geachtet werden, dass sich Leiterbahnen nicht überschneiden und bei den vorgesehenen Stellen genug Platz zum Löten bleibt.

Das so erstellte Design kann als Schablone in PDF-Form exportiert werden. Die Leiterbahnen und alle anderen Stellen, an denen sich Kupfer befindet, werden schwarz eingezeichnet. Die Stellen an denen das Kupfer entfernt werden muss bleiben weiß. Abbildung 22 zeigt beispielsweise die Ätzvorlage für den Spannungsteiler, der in Kapitel 4.5 beschrieben wird. Die Ätzvorlagen für den finalen Prototyp befinden sich im Anhang B.

Diese Schablone wird auf eine durchsichtige Folie gedruckt. Nachdem die Folie auf dem Platinenrohling fixiert wurde, bestrahlt man die Platine einige Minuten lang mit UV-Licht. Dadurch wird der Schutzlack an den, nicht durch die schwarze Farbe geschützten, Stellen geschädigt.

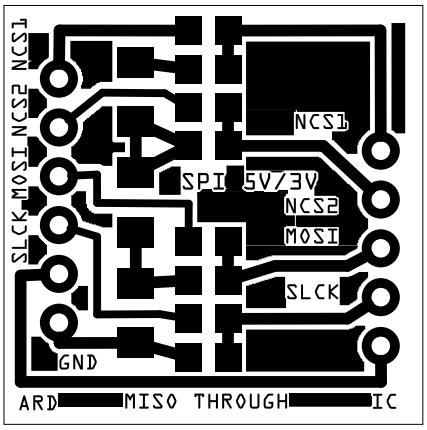


Abbildung 22: Platine mit Spannungsteilern

Im nächsten Schritt wird die Platine für ca. 30 Sekunden in eine Natriumhydroxidlösung gelegt. An den vorher exponierten Stellen löst sich der Schutzlack auf und legt das darunter liegende Kupfer frei. Abbildung 23 zeigt die Bestrahlung mit UV-Licht und den anschließenden Ätzvorgang.

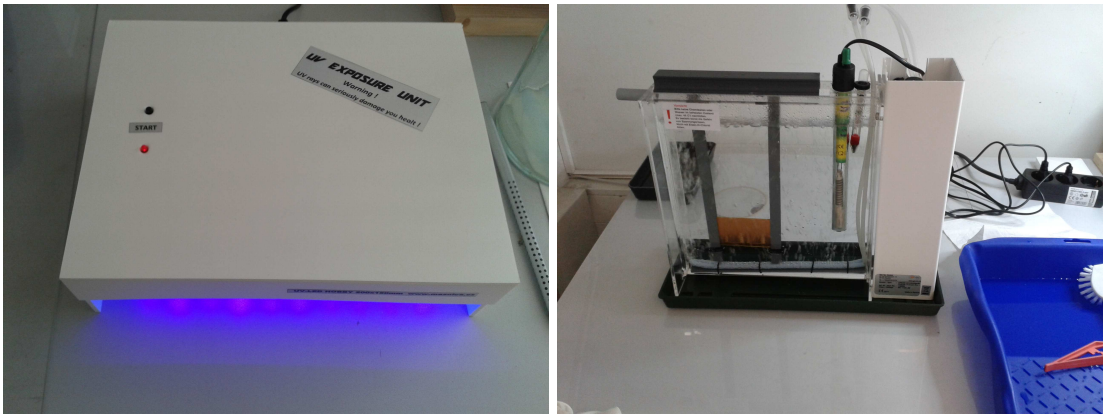


Abbildung 23: Bestrahlung mit UV-Licht und Ätzen mit Natriumpersulfat bei 40°C

Danach wird das Kupfer an den, nicht mehr durch den Lack geschützten, Flächen bei 40°C in Natriumpersulfat aufgelöst. Übrig bleiben die Leiterbahnen und Lötpads. Anschließend kann der restliche Schutzlack über den Leiterbahnen wiederum mit Natriumhydroxid entfernt werden. Das ist allerdings nicht zwingend notwendig, da er sich beim Löten von selbst verflüchtigt.

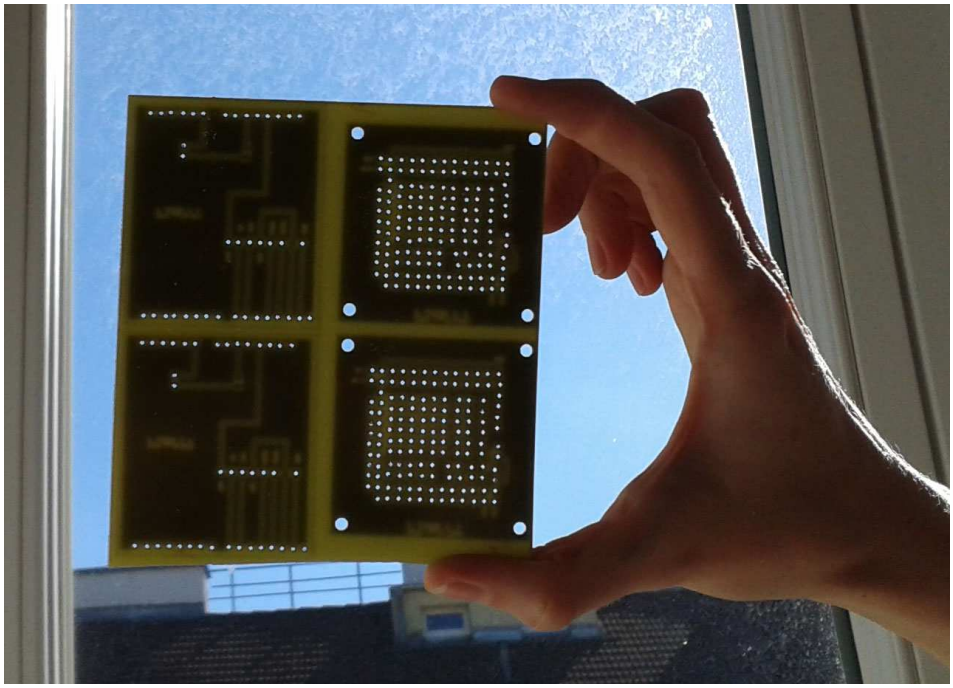


Abbildung 24: Mehrere geätzte Leiterplatten gemeinsam auf einer Europlatine vor dem Zersägen

Die Größe der unbearbeiteten Leiterplatten ist standardisiert. Daher ist es empfehlenswert, mehrere Platinen in einem Schritt zu ätzen (Abbildung 24) und diese anschließend auf die richtige Größe zuzuschneiden. Dies kann zum Beispiel mit einer handelsüblichen Metall- oder Bandsäge erfolgen. Da manche Bauteile, wie der Sensor oder die Steckerleisten, mittels Durchsteckmontage angebracht werden, müssen die entsprechenden Löcher gebohrt werden. Eine Standbohrmaschine erleichtert diese Arbeit erheblich. Des Weiteren werden in der Platine mehrere Ausnehmungen benötigt. Einerseits muss der größte Teil der oberen Platine für die Montage am ACTO entfernt werden. Andererseits müssen auch Aussparungen für die Linse der Sensoren vorgesehen werden. Die entsprechenden Teile wurden zuerst ausgebohrt. Für die anschließende Feinarbeit sind Schlüsselfeilen sehr hilfreich. Damit erhält man eine fertige Platine und die Bauteile können aufgelötet werden.

5.1.2. Auflöten der Bauteile und Kabel

Die überwiegende Zahl der passiven Bauteile sind Kondensatoren und wurden im SMD-Formfaktor 0805 montiert. Dieses Format ist einerseits platzsparend und andererseits für ein problemloses manuelles Löten groß genug. Um Kreuzungen von zwei Leiterbahnen herzustellen, wurden Nullwiderstände im 1206-Format benutzt. Die Sensoren selber wurden, nachdem die Pins etwas gekürzt worden waren, mittels Durchsteckmontage angelötet. Die selbe Montageart kam bei den Steckerleisten zum Einsatz. Die restlichen

Bauteile wie Levelshifter oder Mosfets wurden in SMD-Bauart montiert.

Um die beiden Ebenen zu verbinden, wurden Pinreihen mit Abstandhaltern sowie auch einzelne Pins verwendet (siehe Abbildung 18b in Kapitel 4.5). Diese stellen eine belastbare Verbindung zwischen den Ebenen her und dienen gleichzeitig als Leiterbahnen.

Um die Zahl der Steckverbindungen zu reduzieren, wurden auch Kabel zum Arduino oder ACTO (meist mit Durchsteckmontage) direkt an die Platine des Tracking-Systems gelötet. Vereinzelt wurden zu Testzwecken, oder um größere Entfernungen auf der Platine zu überwinden, auch zwei Lötpads auf der selben Platine mit einem Kabel verbunden. Wenn Kabel an Pins gelötet werden mussten (zum Beispiel an Steckern), wurden die Lötstellen mit passenden Schrumpfschläuchen geschützt.

5.1.3. ACTO Integration

Das Gehäuse des ACTOs ist aus ca. $5mm$ breiten und ebenso hohen quadratischen Plexiglasrahmen mit ca. $50mm$ Kantenlänge aufgebaut, die von 4 senkrechten Schrauben zusammengehalten werden. Beim Entwurf der Platinen wurde von Anfang an die Montage am ACTO mit eingeplant. Dabei wurde von der oberen Platine ein großer Teil ausgeschnitten, um einen viereckigen Rahmen zu erhalten, in den an den Ecken Löcher für die oben erwähnten Schrauben gebohrt wurden. Dieser Rahmen entspricht von der Größe her genau den Plexiglasringen. Daher wird der Rahmen einfach zwischen 2 Plexiglasringe gesteckt und mit den Schrauben mitfixiert.

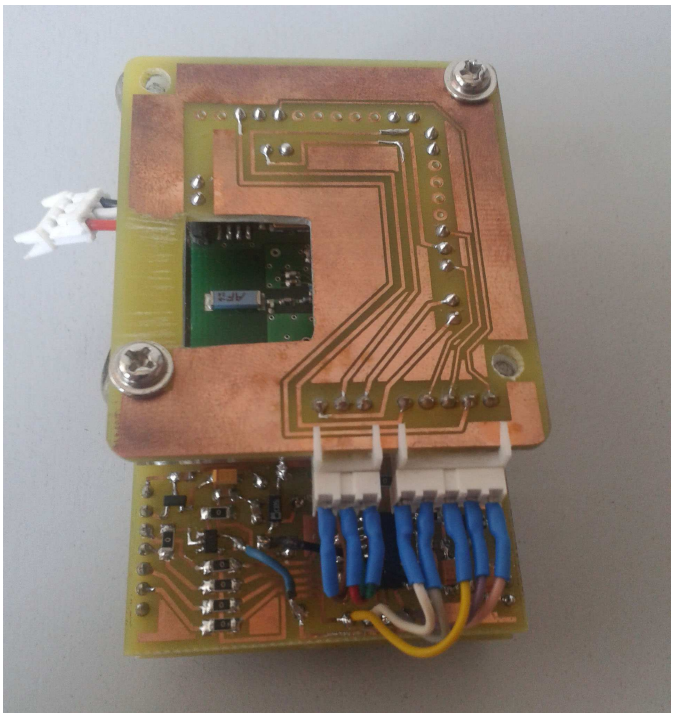


Abbildung 25: Finale ACTO-Erweiterung für das Speckle Tracking (ohne Marker)

Um die Kabelverbindung zum ACTO zu realisieren, wurde eine flache Extension entworfen (siehe 4.7). Auf diese Extension wurden auf der Unterseite Buchsen für eine Steckverbindung mit den Pins des ACTOs montiert. Am Rand der Platine wurden 2 Steckerbuchsen für die Verbindung zum Tracking-System angelötet. Diese Extension wird, wie in Abbildung 25 gezeigt, auf das ACTO aufgesteckt und mit kurzen Schrauben fixiert.

5.2. Konstruktion der technischen Hilfsmittel

Im Laufe der Arbeit wurden verschiedene, selbst gebaute, technische Hilfsmittel, wie eine Kamerahalterung oder eine Führungskonstruktion für Performancetests benutzt.

5.2.1. Kamerahalterung

Bisher wurden die ACTOs mit der Frontkamera eines Android-Gerätes und einem Marker auf der ACTO-Unterseite durch eine Plexiglasscheibe getrackt. Da der ADNS 9500 nicht auf einer durchsichtigen Oberfläche arbeitet, sich aber auf der Unterseite des ACTOs befinden muss, wurde der Marker stattdessen auf der Oberseite befestigt. Um die Kamera (die Kamera auf der Rückseite eines ASUS Transformer TF201 Android-Tablets) ruhig über der Mitte der Testoberfläche zu halten, wurde ein normales Kamerastativ in Kombination mit einer selbst entworfenen Halterung für das Asus Transformer benutzt (Abbildung 26a und 26b).

(a) (b)

Abbildung 26: (a) Asus Transformer TF201 Android-Tablet, (b) Halterung auf Stativ

Da keine entsprechende Halterung im Handel erhältlich war, wurde diese selbst gebaut. Zu diesem Zweck wurden sieben Einzelteile in *123D Design* (siehe A.3) modelliert (Abbildung 27) und in einem *Ultimaker 2* 3D-Drucker (siehe A.4) ausgedruckt. Anschließend wurden die Einzelteile mit Superkleber verklebt.

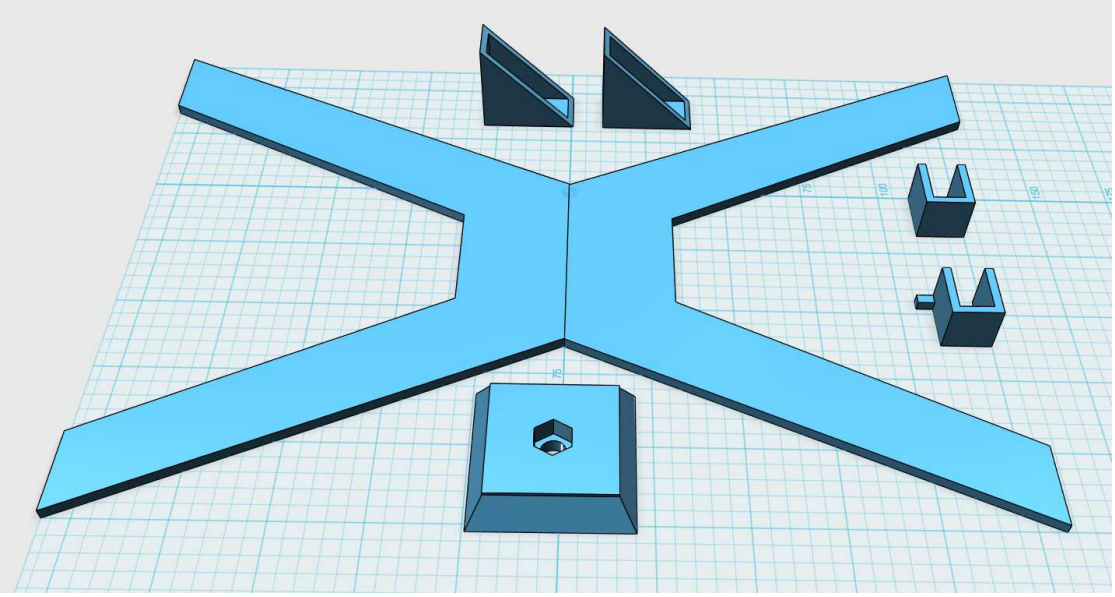


Abbildung 27: Bauteile der Kamerahalterung in *123D Design* [41]

5.2.2. Führungskonstruktion für die Evaluierung

Für die Evaluierung (siehe Kapitel 7) war es notwendig das Tracking-System reproduzierbar entlang eines fixen Pfades zu bewegen. So wurde es möglich den vom System gemessenen Weg mit dem tatsächlich zurückgelegten Weg zu vergleichen und daraus die Abweichung zu berechnen und auszuwerten. Für eine gerade Linie wurde das Tracking-System einfach entlang einer geraden Metallschiene verschoben. Für einen kreisförmigen Pfad erschien das allerdings nicht zielführend. Durch die quadratische Form des ACTOs und der runden Form der Führungsschiene wäre es nicht möglich gewesen des Tracking-System mit gleichbleibender Geschwindigkeit zu bewegen.

Aus diesem Grund wurde statt einer Führungsschiene eine Vorrichtung benutzt, bei der das Tracking-System mit einem Arm auf gleichbleibendem Abstand zum Mittelpunkt der Kreisbewegung gehalten wird. Die Konstruktion besteht aus einer handelsüblichen Türangel, die am Mittelpunkt der Kreisbahn befestigt wird. An dieser Angel wird eine Alustange befestigt, an deren Ende sich die Halterung für das ACTO befindet. Die einzelnen Bauteile wurden entweder mit Leim oder Konstruktionsklebstoff verbunden. Mit dieser Konstruktion kann sicher gestellt werden, dass das Tracking-System exakt der vorgegebenen Kreisbahn folgt. Die Abbildungen 28 und 29 zeigen den Arm für Kurventests und die Schiene für gerade Bewegungen.

Abbildung 28: Führungskonstruktion für Kurventests

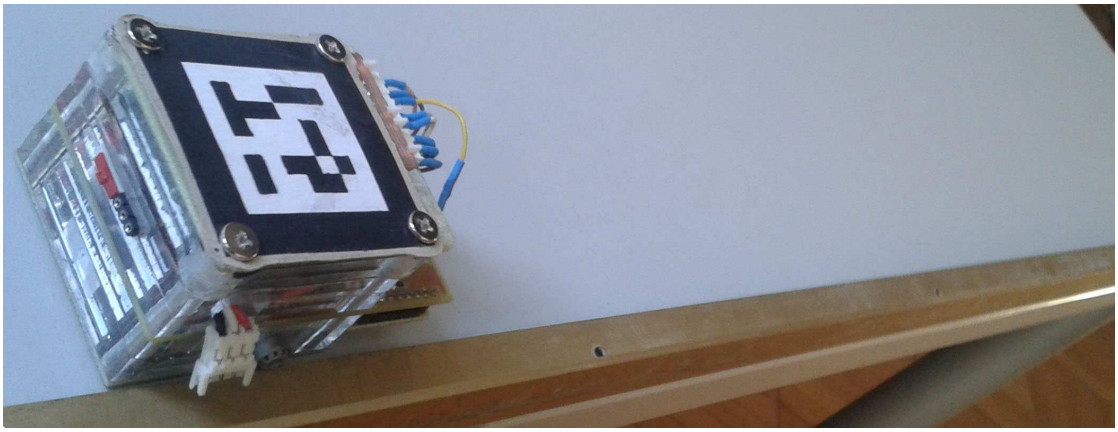


Abbildung 29: Führung entlang einer Schiene

6. Software und Framework

Um zusätzlich zum absoluten Tracking mittels Marker eine zweite Art der Positionsbestimmung implementieren zu können, musste die Software des ACTOs, des Arduino-Empfangsteils und des Android-Geräts erweitert werden. Dieses Kapitel beschreibt sowohl das Framework vor der Speckle-Tracking-Integration, wie auch die notwendigen Veränderungen daran. Die Adaptierungen werden anhand des Weges der Tracking-Daten, von den Sensoren bis auf den Bildschirm erläutert. Anschließend wird genauer auf die, für das Tracking-System notwendige, Kalibrierung eingegangen.

6.1. Ausgangslage

Ursprünglich wurden die ACTOs von einem Android-Gerät (Smartphone oder Tablet), wie in Abbildung 30 dargestellt, gesteuert und koordiniert. Dabei bewegen sich die ACTOs auf einer Plexiglasoberfläche und werden von unten von einer Kamera gefilmt. Die Kamera kann sich dabei sogar im Android-Steuergerät selbst befinden. Es kann also die integrierte Kamera des Smartphones oder Tablets benutzt werden. Diese Kamera verfolgt die auf der Unterseite der ACTOs befestigten Marker. Somit weiß das Steuergereät welches ACTO sich wo befindet.

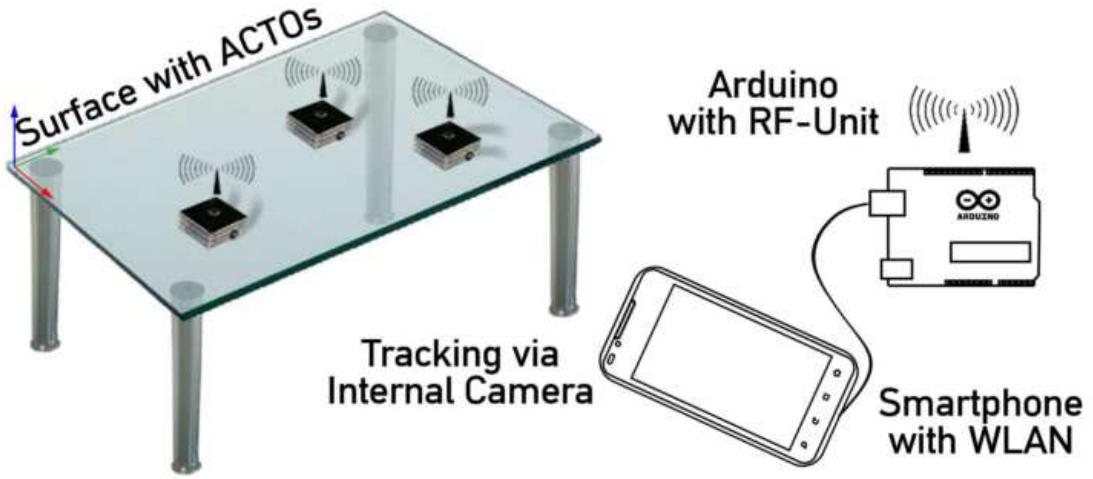


Abbildung 30: ACTO-Schema [3]

Abbildung 31 zeigt wie das ACTO-OS auf dem ACTO, über mehrere Schritte, mit dem ACTO-Server-App auf dem Adroid-Steuergerät kommuniziert. Dafür besitzt das ACTO einen integrierten NRF24L01+ Funkchip [31]. Dieser wird über SPI angesprochen (siehe 3.3). Das Android-Steuergerät hat allerdings üblicherweise keinen kompatiblen Funkchip integriert. Deswegen wird ein Arduino UNO mit einem daran angeschlossenen NRF24L01+ Funkchip als Server-Arduino benutzt. Das Server-Arduino kommuniziert einerseits mit dem ACTO-Server-App auf dem Adroid-Steuergerät über USB und ande-

rerseits mit seinem Funkchip und damit mit dem ACTO-OS auf dem ACTO. Alle Daten werden über diese Kaskade in Byte-Form gesendet. Damit der Empfänger weiß, welche Art von Daten vorliegt, definiert jeweils das erste Byte des Pakets, wie die Daten interpretiert werden müssen. Daraus wird auch die zu erwartende Länge des Datenpakets abgelesen. Input und Output erfolgen über das GUI des ACTO-Server-Apps.

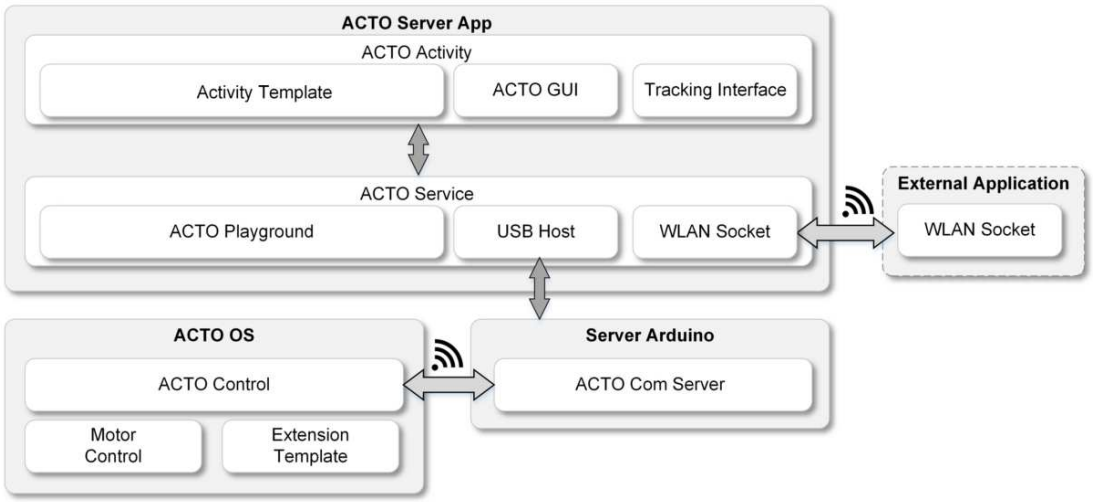


Abbildung 31: ACTO-Framework [3]

Diese Kommunikation wird im ursprünglichen Framework nur dazu benutzt, Steuersignale an die Motoreinheit des ACTOs, oder Output-Daten von ACTO-Extensions (siehe 3.2.3) an das Steuergerät zu senden. Tracking-Daten werden in der Implementierung bisher ausschließlich von der Kamera des Steuergeräts erfasst.

6.2. Veränderungen am Framework

Um diese Verarbeitung der Tracking-Daten zu ermöglichen, musste das Framework an mehreren Stellen adaptiert werden. Die Veränderungen werden im Folgendem in der Reihenfolge abgehandelt, die dem Weg der Rohdaten bis zu Bildschirmausgabe am ehesten entspricht.

6.2.1. ACTO-Sketch

Der Arduino-Sketch, also der in der Arduino-IDE geschriebene Code, musste für das ACTO erweitert werden. Dabei wurde die gesamte Steuerungssoftware für die Sensoren und ein Teil der Tracking-Logik implementiert.

Das SPI musste ebenfalls angepasst werden um mit mehreren Geräten zurecht zu kommen. Konkret wurden in einer selbst geschriebenen `SPI.begin()`-Methode 2 statt einer Chip-Select-Leitung vorgesehen. Beide Chip-Select-Leitungen müssen bei der SPI-

Initialisierung in der `SPI.begin()`-Methode als Output definiert und auf „High“ gesetzt werden.

Es wurden eigene Methoden für die Read- und Write-Vorgänge geschrieben. In diesen muss zu erst die Chip-Select-Leitung des anzusteuernden Sensors auf „Low“ und die andere Chip-Select-Leitung auf „High“ gesetzt werden. Anschließend wird mit dem Parameter der `SPI.transfere()`-Methode der anzusprechende Register definiert und angegeben ob es sich um einen Read- oder Write-Vorgang handelt. Dann wird die `SPI.transfere()`-Methode ein zweites Mal aufgerufen. Diesmal werden, im Falle eines Write-Vorganges, die zu übertragenden Daten als Parameter übergeben. Im Falle eines Read-Vorganges, werden die gelesenen Daten von der Methode zurückgegeben. Da der Funkchip des ACTOs einen anderen SPI-Modus (siehe 3.3) also der ADNS 9500 benutzt, muss am Beginn und Ende jedes Read- oder Write-Vorganges der Modus mit der `SPI.setDataMode()`-Methode geändert werden.

Bei jedem Start des Systems muss die Firmware der ADNS 9500-Sensoren vom Arduino auf die Sensoren übertragen werden. Dafür werden beim Start, wie im Datenblatt des ADNS 9500 [4] beschrieben, bestimmte Werte in verschiedene Register geschrieben. Anschließend kann die Firmware Byte für Byte übertragen werden. Als Zielregisters wird der `SROM_Load_Burst`-Register angegeben. Da die Firmware als Variable zu groß für den SRAM des Arduinos ist, wird sie in dessen Flash-Speicher gespeichert.

Für die Verarbeitung der Tracking-Daten ruft das ACTO zuerst über den SPI-Bus (siehe 3.3) die Δx- und Δy-Werte von den Sensoren ab. Diese Δ-Werte repräsentieren die Verschiebung der Sensoren seitdem sie das letzte Mal ausgelesen wurden. Das geschieht circa 40 mal pro Sekunde, für den linken und rechten Sensor und für die x- und y-Achse. Jeder Δ-Wert besteht dabei aus einem High- und einen Low-Byte das jeweils in einem eigenen Register untergebracht ist. Insgesamt müssen also 8 Register ausgelesen werden um die, für das Tracking notwendigen, Werte $(\Delta x_L, \Delta y_L, \Delta x_R, \Delta y_R)$ zu erhalten.

Der folgende Arduino-Code zeigt wie Low- und High-Byte für die x-Achse des linken Sensors (`bdxlL, bdxhL`) mit der Methode `adns_read_reg` ausgelesen und anschließend gemeinsam als Integer (`ixL`) gespeichert werden. Aus diesem Integer-Wert wird anschließend die Distanz in mm (`mmxL`) berechnet.

```
// read high and low byte for x-axis from left Sensor
byte bdxlL = adns_read_reg_L(REG_Delta_X_L);
byte bdxhL = adns_read_reg_L(REG_Delta_X_H);

// perform shift operation and write to int variable
int ixL = bdxlL | bdxhL << 8;

// compute mm value from dots/inch format and write to double variable
double mmxL = double(ixL)/5040*25.4;
```

Das ACTO berechnet (nachdem die Kalibrierung eingeflossen ist - siehe 6.3) aus den 4 Δ-Werten $(\Delta x_L, \Delta y_L, \Delta x_R, \Delta y_R)$, welche die Verschiebung der Sensoren aus-

drücken, einen zweidimensionalen Vektor ($\Delta x/\Delta y$) und den Rotationswert $\Delta\theta$. Diese drei Werte repräsentieren die Verschiebung des Systems von der letzten Position. Der Systemursprung ist dabei der Mittelpunkt zwischen den beiden Sensoren. Der Algorithmus hierfür wurde wie folgt umgesetzt und bereits in Kapitel 3.5 genauer erläutert.

Die Variablen `mmxL`, `mmyL`, etc. sind die Δ-Werte der Sensoren in mm. Die Ergebnisse der trigonometrischen Berechnungen (`SpeddTrig.cos()` und `SpeddTrig.sin()`) werden in das Array a geschrieben. Die trigonometrischen Berechnungen müssen theoretisch nicht jedes Mal ausgeführt werden, sondern können als Konstanten (je nach Ausrichtung der Sensoren, wie in Kapitel 3.5 beschrieben, beispielsweise mit 0 oder 1) ersetzt werden. Anschließend wird die Matrix `pseudoInverseA[]` mit der Matrix `a[]` multipliziert um das Array `u[]` zu erhalten.

```
// mmxL/mmyL/mmxR/mmyR
// distance values for x-/y-axis and left/right Sensor in mm
a[0][0] = mmxL*SpeedTrig.cos(180) - mmyL*SpeedTrig.sin(180);
a[1][0] = mmxL*SpeedTrig.sin(180) + mmyL*SpeedTrig.cos(180);
a[2][0] = mmxR*SpeedTrig.cos(180) - mmyR*SpeedTrig.sin(180);
a[3][0] = mmxR*SpeedTrig.sin(180) + mmyR*SpeedTrig.cos(180);

// compute position and rotation from distance values
Matrix.Multiply((float*)pseudoInverseA,(float*)a,3,4,1,(float*)u);
```

Die 3 Δ-Werte ($\Delta x, \Delta y, \Delta\theta$) aus dem Array `u[]` werden in Form von 2 Arrays (`transFloatMsg[]` und `rotFloatMsg[]`) gespeichert, und dann mit der Methode `writeTrkPoseMsg()` an ein Arduino UNO mit Funkextension gesendet.

```
// write values from u[] to translation and rotation array
// (x, y, z) z is always zero
transFloatMsg[0] = u[0];
transFloatMsg[1] = u[1];
transFloatMsg[2] = 0;

// x Convention (z, x', z'')
rotFloatMsg[0] = u[2];
rotFloatMsg[1] = 0;
rotFloatMsg[2] = 0;

// send translation and rotation to Arduino UNO
writeTrkPoseMsg(USB_TRK_POSE, transFloatMsg, rotFloatMsg);
```

6.2.2. Arduino UNO Funkempfänger

Von dort werden die Tracking-Daten über USB an ein Android-Gerät weitergeleitet, auf dem das ACTO-Framework läuft.

```
case USB_TRK_POSE:
Serial.write(inMirfBuffer,27);
break;
```

Generell mussten für die Kommunikation vom ACTO bis zum ACTO-Framework zusätzliche Message-Arten eingeführt werden. Die notwendigen Messages wurden im ACTO- und Arduino UNO-Sketch und im ACTO-Framework implementiert. In jeder Message definiert das erste Byte die Art der gesendeten Daten. In diesem Fall wurde der Message-Art, die die Translation und Rotation beinhaltet, die Nummer 56 im ersten Byte zugewiesen. Damit ist die Anzahl und Bedeutung der folgenden Bytes festgelegt. Es folgen die ACTO-ID und die eigentlichen Tracking-Daten. Am Schluss steht ein Separator-Byte.

```
// tracker data from ACTO POSE
// format: CMD, ID, translation float[3], rotationEUL float[3], SEP
const byte USB_TRK_POSE = 56;
```

6.2.3. ACTO-Framework

Auf dem Android-Gerät (im Falle der Arbeit ein Asus-Transformer TF201 Tablet), wird aus der letzten Position und dem Array $u[\Delta x, \Delta y, \Delta\theta]$, das sich aus dem Translations- und Rotations-Array zusammensetzt, die neue Position berechnet. Sollte das absolute Tracking (basierend auf dem Kamera-Tracking mittels Markern) eine valide Tracking-Position/Ausrichtung ermitteln, wird die aktuelle Position/Ausrichtung des ACTOs sofort mit dieser überschrieben. Wenn das nächste Mal Tracking-Daten vom ACTO (basierend auf den Maus-Sensoren) empfangen werden, wird die neue Position auf Basis der aktualisierten Position berechnet. Diese Vorgehensweise trägt der Annahme Rechnung, dass das absolute Tracking zwar seltener, dafür aber zuverlässige Daten liefert.

Deswegen wurde im ACTO-Framework des Android-Geräts der zweite Teil der Tracking-Logik und die Einbindung in das bereits existierende Marker-Tracking realisiert. Die Hauptschwierigkeit stellten dabei die unterschiedlichen Formate der Tracking-Daten dar. Die Rotation des Markers liegt als Quaternion und die des Speckle-Tracking-Systems als eulersche Rotation vor. Die Berechnung der neuen Position wird anhand der folgenden Formel durchgeführt:

$$\begin{bmatrix} X_t \\ Y_t \\ \Theta_t \end{bmatrix} = \begin{bmatrix} X_{t-1} + \Delta x \cos\Theta_{t-1} - \Delta y \sin\Theta_{t-1} \\ Y_{t-1} + \Delta x \sin\Theta_{t-1} + \Delta y \cos\Theta_{t-1} \\ \Theta_{t-1} + \Delta\theta \end{bmatrix}$$

Dabei wird, im unten angeführten Code auf dem ACTO, die neue Position `speckleTrackingSystemPose[]` aus der alten Position (ebenfalls `speckleTrackingSystemPose[]`)

und dem Array `u[]` berechnet. Im selben Schritt wird die Variable der alten Position `speckleTrackingSystemPose[]` mit der Aktualisierten überschrieben.

```
// Updating SpeckleTrackingSystemPose (X, Y, Theta)
speckleTrackingSystemPose[0] = (float) (speckleTrackingSystemPose[0]-
   playPoseByActo.translation[0]*Math.cos(speckleTrackingSystemPose[2])+
   playPoseByActo.translation[1]*Math.sin(speckleTrackingSystemPose[2]));

speckleTrackingSystemPose[1] = (float) (speckleTrackingSystemPose[1]+
   playPoseByActo.translation[0]*Math.sin(speckleTrackingSystemPose[2])+
   playPoseByActo.translation[1]*Math.cos(speckleTrackingSystemPose[2]));

speckleTrackingSystemPose[2] =
   speckleTrackingSystemPose[2] + playPoseByActo.rotation[0];
```

Außerdem befinden sich weder der Marker noch der Mittelpunkt des Speckle-Tracking-Systems im Mittelpunkt des ACTOs. Der Marker befindet sich einige Zentimeter über der Oberfläche (oben auf dem ACTO) und die Sensoren sind außerhalb des ACTOs montiert. Daher waren zusätzliche Berechnung notwendig, um immer Tracking-Daten für den Mittelpunkt des ACTOs (playPose.translation) auf der Oberfläche zu erhalten.

```
// translate System Position to Marker/Acto Position
speckleTrackingMarkerPose[0] = (float)
      (speckleTrackingSystemPose[0]-
      Math.cos(playPoseByActo.rotation[0]-
      Math.PI/2) * markerSpeckleTrackingSystemDistance);

speckleTrackingMarkerPose[1] = (float)
   (speckleTrackingSystemPose[1]+
   Math.sin(playPoseByActo.rotation[0]-
   Math.PI/2) * markerSpeckleTrackingSystemDistance);

speckleTrackingMarkerPose[2] = speckleTrackingSystemPose[2];

playPose.translation = speckleTrackingMarkerPose;
```

Sobald Tracking-Daten vom absoluten Tracking verfügbar sind, werden die im Framework vorliegenden Tracking-Daten aktualisiert. Das relative Speckle-Tracking wird dann auf Basis dieser korrigierten Werte fortgesetzt. Im Zuge dieser Korrektur muss einerseits wieder vom Quarternion umgerechnet werden. So wird die Rotation Θ als Variable `yaw` aus der Position in der Quarterniondarstellung berechnet. Andererseits muss berücksichtigt werden, dass sich weder der Marker noch das Speckle-Tracking-System im Mittelpunkt des ACTO-Bezugssystems befinden. Daher wird aus `yaw` und den x- und y-

Koordinaten, unter Einbeziehung der Markerpostion relativ zum ACTO-Bezugsystem, die aktualisierte ACTO-Postion `speckleTrackingSystemPose[]` berechnet. Diese Umrechnungen werden wie folgt durchgeführt:

```
// compute Yaw from Quaternion
float yaw = (float) (getYawFromQuaternion(new Quaternion(
        (double)worldPose.rotation[3],
        (double)worldPose.rotation[0],
        (double)worldPose.rotation[1],
        (double)worldPose.rotation[2])) + Math.PI);

// translate MarkerPosition (WorldPose) to speckleTrackingSystem
speckleTrackingSystemPose[0] = (float)
        (worldPose.translation[0] +
        Math.cos(yaw - Math.PI/2) * markerSpeckleTrackingSystemDistance);

speckleTrackingSystemPose[1] = (float)
        (worldPose.translation[1] -
        Math.sin(yaw - Math.PI/2) * markerSpeckleTrackingSystemDistance);

speckleTrackingSystemPose[2] = yaw;
```

6.2.4. GUI

Anschließend wird die Position, im Rahmen der Evaluierung, am Bildschirm ausgegeben. Daher wurde das GUI um drei Textfelder für die Position und Ausrichtung des ACTOs erweitert. Ein Button um diese Werte zurücksetzen zu können wurde ebenfalls implementiert. Die Anpassung des GUIs war allerdings nicht für das Speckle-Tracking-System an sich notwendig, sondern wurde ausschließlich für die Evaluierung und das Debugging verwendet. Für den laufenden Betrieb ist diese Änderung nur von untergeordneter Bedeutung.

6.3. Kalibrierung

Da die Sensoren (unter anderem abhängig von der Oberfläche) von Haus aus einen relativ konstanten Messfehler aufweisen, müssen sie kalibriert werden. Der Einfluss und die Notwendigkeit der Kalibrierung werden auch anhand der Ergebnisse in Kapitel 7 deutlich. Die Kalibrierung fließt bereits vor der Berechnung von $u[\Delta x, \Delta y, \Delta\theta]$ auf dem ACTO, für jede der 4 Bewegungsrichtungen beider Sensoren, in die Tracking-Daten mit ein.

Wenn beispielsweise der linke Sensor in negativer x-Richtung (`mmxL < 0`) immer um 10 Prozent zu viel misst (also das 1.1 fache), kann daraus ein Bias-Wert `negLXBias` errechnet werden:

```
negLXBias = mmxL / 1.1;
```

Registriert das ACTO das nächste Mal einen negativen `mmxL`-Wert, wird dieser mit dem zuvor definierten Bias-Wert `negLXBias` korrigiert:

```
if (mmxL < 0) mmxL *= negLXBias;
```

Da es für jeden der beiden Sensoren 4 mögliche Bewegungsrichtung gibt, werden 8 Bias-Werte während der Kalibrierung berechnet. Im Folgenden werden die Voraussetzungen und die Durchführung der Kalibrierung beim ACTO beschrieben.

6.3.1. Voraussetzungen für die Kalibrierung

- Das Tracking-System muss mit dem ACTO fix verbunden (verschraubt) sein. Wenn sich das Tracking-System relativ zum ACTO später verschiebt, ist die Kalibrierung sehr wahrscheinlich nicht mehr korrekt.
- Das Tracking-System muss am ACTO angeschlossen sein. Sowohl die Stromversorgung, als auch die SPI-Pins müssen angeschlossen sein.
- Das ACTO muss über USB mit dem PC verbunden sein, um den Arduino-Sketch einspielen und die Tracking-Werte auslesen zu können.

6.3.2. Durchführung der Kalibrierung

Zuerst muss sich das ACTO im Kalibrierungsmodus befinden. Der Kalibrierungsmodus muss im Arduino-Sketch durch das Setzen einer entsprechenden Variable aktiviert werden. Zusätzlich muss die Länge des Kalibrierungstests in *mm* in einer Variable definiert werden. Die Länge ergibt sich durch den verfügbaren Platz. Das ACTO muss später um diese Entfernung entlang einer Achse verschoben werden. 300 oder 500 Millimeter wären sinnvolle Werte. Anschließend wird der Sketch auf das ACTO hochgeladen und gestartet.

Nach dem Start des ACTOs werden im Serial-Monitor einige Daten zur ADNS-Firmware angezeigt. Sofort danach wird die Messung der zurückgelegten Strecke gestartet. Im Terminal werden jetzt 2 x- und 2 y-Werte automatisch aktualisiert. Diese Werte stellen die von den Sensoren gemessene Strecke in x- und y-Richtung in *mm* dar. Jeweils ein x- und ein y-Wert gehören zum linken (Lx/Ly), beziehungsweise zum rechten Sensor (Rx/Ry). Die Messwerte beziehen sich auf den Punkt an dem die Messung gestartet, also das ACTO eingeschaltet, wurde.

Das ACTO muss jetzt genau entlang der x-Achse entlang eines Lineals nach rechts um die vorher definierte Länge des Kalibrierungstests verschoben werden. Dabei darf es nicht gedreht werden. Eine Verschiebung nach rechts sollte (abhängig von der Ausrichtung der Sensoren) negative Werte liefern, während eine Verschiebung nach links

positive Werte zur Folge hat. Im Idealfall stimmen die Lx und Rx Werte fast mit der angegebenen Länge überein. In der Praxis werden sie aber abweichen. Die beiden der Achse entsprechenden Werte müssen (inklusive Vorzeichen) notiert werden. Die beiden Werte der anderen Achse sollten sich abhängig von der genauen Lage der Sensoren um 0 herum bewegen. Eine leichte (bei der Montage kaum vermeidbare) Drehung der Sensoren bewirkt hier Werte abweichend von 0. Diese Messung wird auf der x-Achse auch nach links und auf der y-Achse ebenfalls in beide Richtungen durchgeführt. Dabei müssen die beiden korrespondierenden Werte immer notiert werden. Vor jeder Messung muss das ACTO an den Nullpunkt des Lineals gesetzt und neu gestartet werden, da nur so die interne Messung auf 0 zurückgesetzt wird. Zwischen dem Start des ACTOs und dem Beginn der manuellen Bewegung entlang des Lineals, darf das ACTO nicht bewegt werden, da die Messung bereits mit dem Start des ACTOs beginnt. Mit anderen Worten: Es ist wichtig, dass sich das ACTO beim Einschalten bereits am Nullpunkt befindet und dann bis zur Messung nicht mehr bewegt wird. Am Ende jeder Messung, vor dem Notieren der Werte, empfiehlt es sich das ACTO abzuschalten, da dann die Daten im Serial-Monitor einfacher lesbar sind.

Die 8 so gemessenen und notierten Werte müssen schließlich in die entsprechenden Variablen im Arduino-Sketch eingetragen werden. Es sollte für die Lx/Ly/Rx/Ry-Werte jeweils einen positiven und einen negativen Wert in der ungefähren Höhe der Testlänge geben.

Abschließend wird der Kalibrierungsmodus durch das Setzen der entsprechenden Variable im Arduino-Sketch wieder deaktiviert. Der mit so konfigurierte Sketch kann jetzt auf das ACTO hoch geladen werden. Von nun an werden bei jedem Start aus der Soll-Länge und den gemessenen/eingetragenen Bewegungslängen Kalibrierungsfaktoren (Bias-Werte) berechnet. Mit diesen Faktoren wird, abhängig von der Bewegungsrichtung, jede von den Sensoren gemessene Verschiebung multipliziert und somit korrigiert.

7. Evaluierung

In diesem Abschnitt werden die verschiedenen Tests besprochen, die notwendig sind, um die Zuverlässigkeit, die Genauigkeit und die Grenzen des Tracking-Systems einschätzen zu können. Direkt anschließend werden für jeden Tests die dafür nötigen Testparameter beschrieben und es wird dargelegt, warum gerade diese Werte gewählt wurden. Danach werden die Testläufe ausgewertet und interpretiert.

Grundsätzlich gibt es bei dieser Art von Tracking-Systemen 2 Fehlerquellen. Einerseits weist der Algorithmus, mit dessen Hilfe aus den 2 Δx- und 2 Δy-Werten der zurückgelegte Pfad berechnet wird, eine Ungenauigkeit auf (siehe Kapitel 7.1.7). Andererseits können die gemessenen Δx- und Δy-Werte Abweichungen von der tatsächlich vom Sensor zurückgelegten Strecke aufweisen. Die möglichen Gründe für eine solche Ungenauigkeit können vielfältig sein. In dieser Arbeit werden als systematische Fehler der Ladezustand des Akkus und die Bewegungsgeschwindigkeit des Systems behandelt, obwohl durchaus auch andere Einflüsse, wie zum Beispiel die Entfernung der Linse zum Untergrund, denkbar sind.

7.1. Algorithmustests

Der verwendete Algorithmus weist einen gewissen Fehler auf, der sowohl mit der Größe von Δx und Δy, als auch mit der Differenz der beiden Δy-Werte (welche die Rotation bestimmt), zusammenhängt. In den folgenden Tests wird die gleiche Hardwareumgebung wie beim realen Tracking verwendet. Der größte Unterschied besteht darin, dass die gemessenen Werte durch vorgefertigte ersetzt werden.

Damit verwenden die Algorithmustests keinen realen Input der Sensoren, sondern simulieren optimale Δx- und Δy-Werte. Es ist bekannt welche Endposition aus diesen optimalen Input-Werten resultieren sollte. Diese bekannte Endposition wird mit der vom Algorithmus berechneten Endposition verglichen. Dadurch kann die Präzision des zugrundeliegenden Algorithmus, unabhängig von der schwankenden Input-Qualität, getestet werden. Ziel der Algorithmustests ist es festzustellen, wie stark die Beschaffenheit des Algorithmus die Genauigkeit des Tracking-Systems beeinflussen kann.

In der Praxis sind kompliziertere als die getesteten Bewegungsmuster möglich. Die Algorithmustests wurden allerdings möglichst einfach gehalten, um nachvollziehbar und überprüfbar zu bleiben.

In diesem Kapitel wird zuerst die Wahl der Parameter basierend auf der Dauer eines Schleifendurchlaufes in der Software erläutert. Anschließend werden die getesteten Bewegungsmuster erläutert und die Resultate in Tabellenform dargestellt und interpretiert. Die Interpretation und Gegenüberstellung der Resultate erfolgt abschließend gesammelt in 7.1.7.

7.1.1. Bedeutung und Messung der Schleifendurchlaufzeiten

Mit diesem Test wird überprüft, wie lange die einzelnen Teile des Algorithmus zur Ausführung benötigen und wo Optimierungspotential vorhanden ist. Dies ist notwendig um

sinnvolle Parameter für die simulierten Bewegungen der Algorithmustests abschätzen zu können.

Der ADNS 9500 misst die zurückgelegte Strecke bis zu 11750 mal pro Sekunde. [4] Diese zurückgelegte Entfernung wird so lange aufsummiert und intern gespeichert bis die Δx- und Δy-Werte abgerufen werden. Werden die Werte vom Arduino oder ACTO abgerufen, werden sie im ADNS 9500 wieder auf 0 gesetzt und bis zum nächsten Lesevorgang erneut aufsummiert.

Das Auslesen der Deltawerte vom Arduino benötigt allerdings Zeit und kann deswegen nur wesentlich seltener als die Messrate erfolgen. Umso seltener die Deltawerte ausgelesen werden, umso größer werden sie (bei gleich bleibender Bewegungsgeschwindigkeit). Theoretisch wird der Algorithmus genauer, umso kleiner die Deltawerte sind. Damit ist eine hohe Ausleserate wichtig, um auch bei höheren Geschwindigkeiten niedrige Deltawerte zu erhalten.

Um festzustellen wie oft Δx und Δy pro Sekunde ausgelesen werden können, ist es notwendig zu wissen, wie lange ein Schleifendurchlauf benötigt, da die Werte grundsätzlich pro Durchlauf genau einmal ausgelesen und verarbeitet werden. Sollte also ein Durchlauf $50ms$ dauern, werden die Werte 20 mal pro Sekunde ausgelesen. Bei einer angenommenen Bewegungsgeschwindigkeit von $50mm/s$ werden damit pro Schleifendurchlauf (in den folgenden Tabellen als „Loop "bezeichnet) $2.5mm$ gemessen. Die gemessene Zeit stellt einen kompletten Durchlauf der Schleife dar und beinhaltet damit auch die Schleifenmethoden der anderen Programmteile, wie zum Beispiel die der Motorsteuerung.

Um die Schleifendurchlaufzeit zu ermitteln, wurde die Funktion millis() benutzt. Diese Funktion gibt die Millisekunden aus die seit dem Start des ACTOs vergangen sind. Die Differenz zwischen zwei millis()-Werten ist die Zeit die für einen Schleifendurchlauf benötigt wird. Die Schleifendurchlaufzeit wurde inklusive eines $20ms$ andauernden Delays in der Tracking-Schleife gemessen. Diese Verzögerung ist notwendig um den Mikrocontroller nicht unnötigerweise komplett auszulasten. Daher müssen von den gemessenen Werten diese $20ms$ noch abgezogen werden, um die theoretisch minimale Schleifendurchlaufzeit zu erhalten.

Für die Berechnung, wurde der Durchschnitt der Schleifendurchlaufzeiten errechnet und in Tabelle 4 dargestellt. Die Durchlaufzeiten schwanken zwischen $24ms$ und $25ms$. Daraus lässt sich schließen, dass ein kompletter Schleifendurchlauf ohne Delay mindestens $4 - 5ms$ dauert.

gemessen	**Delay**	**theoretisches Minimum**
24.5ms	20ms	4.5ms

Tabelle 4: Durchschnittliche Schleifendurchlaufzeit

7.1.2. Allgemeine Überlegungen zu den verwendeten Testparametern

Grundsätzlich werden bei den Algorithmustests 2 Arten von Bewegungen gemessen. Einerseits wird eine gerade Bewegung gemessen wenn beide Sensoren das gleiche Δx, beziehungsweise das gleiche Δy messen. Andererseits wird eine Kreisbahn gemessen, wenn kein Δx gemessen wird, dafür unterschiedliche Δy-Werte.

In der Praxis wird sehr häufig eine dritte Bewegungsform auftreten, die für Algorithmustests aber nicht geeignet ist. Wenn Δx-Werte (die aufgrund der fixen Ausrichtung der Sensoren zueinander bei beiden Sensoren gleich sein müssen) und gleichzeitig unterschiedliche Δy-Werte gemessen werden, bedeutet das Rotation und Translation gleichzeitig. Bei dieser Art der Bewegung lässt sich allerdings die Abweichung bei den Algorithmustests nur schwierig berechnen. Es wäre sehr aufwendig passende Input-Werte für eine so komplexe Bewegung zu finden, bei der die theoretische Endposition immer noch trivial zu berechnen ist.

Bei einer geraden Bewegung ohne Rotation lässt sich die Endposition einfach aus der Summe der einzelnen Messungen berechnen. Zum Beispiel ergibt eine Geschwindigkeit von $50mm/s$ bei einer Schleifendurchlaufzeit von $50ms$ pro Messung ein Δ von $2.5mm$. Die Anzahl der Messungen und damit der Iterationen im Algorithmus oder die Schrittweite sollten sich, bei einer geraden Bewegung entlang einer der beiden Achsen, nicht auf die Präzision auswirken. Daher wurde einfach eine Bewegungsweite von insgesamt $100mm$ angenommen. Die Schrittweite hängt von der angenommenen Geschwindigkeit ab.

Bei einer Kreisbewegung bewegen sich entweder einer oder beiden Sensoren auf einer Kreisbahn. Insgesamt findet so eine Rotation des Systems statt. Um möglichst praxisnahe Bedingungen zu schaffen und eine exakte Berechnung der Sollwerte zu ermöglichen, wird als Kreisradius für die Bewegung der Sensoren (nicht der des Gesamtsystems) bei 7.1.4 die halbe Distanz zwischen Sensor A und Sensor B und bei 7.1.5 die Distanz zwischen Sensor A und Sensor B angenommen. Die Anzahl der Schritte und die Schrittweite werden so angenommen, dass der Endpunkt des Systems wieder der Anfangspunkt ist (es wird also eine Drehung von 360° durchgeführt). Des Weiteren soll die geforderte Testgeschwindigkeit möglichst exakt erreicht werden. Um eine trotz der vorgegebenen Geschwindigkeit und Schleifendurchlaufzeit ganze Zahl an Schritten zu erhalten, wurde die Zahl der Schritte gerundet und die Schrittweite auf Basis dieses Wertes berechnet. Damit sind die Geschwindigkeit in mm/s und die Schleifendurchlaufzeit Näherungswerte. So ergibt sich bei 7.1.5, bei einer ungefähren Geschwindigkeit von $50mm/s$, eine Schrittweite von $2.5117mm$, um bei dem gegebenen Radius den Ausgangspunkt wieder zu erreichen.

Für die Simulation einer Kreisbewegung mit 2 bewegten Sensoren wurde als Radius für den inneren Sensor der einfache Abstand und für den äußeren Sensor der doppelte Abstand der Sensoren zueinander gewählt.

Da, wie sich im Verlauf der Tests herausstellte, 360° Kreise unabhängig von der Schrittweite und Anzahl zu keinen relevanten Fehlern führen, wurden zusätzlich praxisrelevantere Halb- und Viertelkreise getestet. Weiters wurde zur besseren Vergleichbarkeit bei der Simulation von Halb- und Viertelkreisen, eine fixe Schleifendurchlaufzeit von $25ms$ gewählt, um eine bessere Vergleichbarkeit zu gewährleisten.

7.1.3. Gerade Bewegung

Bei diesem Testsetup wird das Tracking-System virtuell gerade entlang der x- oder y-Achse bewegt (Abbildung 32). Dadurch bleiben immer entweder die Δx- oder Δy-Werte 0. Bei einer simulieren 100 Prozent geraden Bewegung sollten keine Fehler bedingt durch den Algorithmus auftreten.

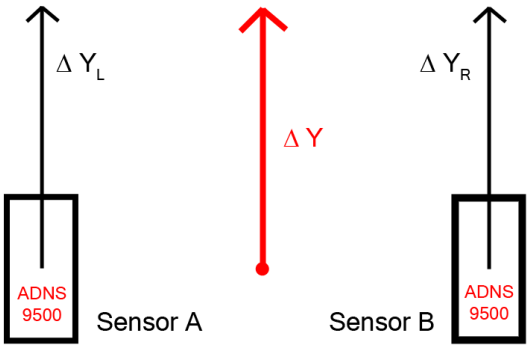


Abbildung 32: Gerade Strecke mit 2 bewegten Sensoren

Um eine gerade Bewegung in y-Richtung zu simulieren, messen beide Sensoren keine Bewegung in x-Richtung und bei jedem Schritt die gleichen Werte in y-Richtung. Um Rechenfehler auszuschließen wurden ausschließlich Werte für die Schrittanzahl und Schrittweite gewählt die maximal 2 Nachkommastellen aufweisen. Die Tabellen 5 und 6 zeigen die verwendeten Input-Werte und Ergebnisse.

v mm/s	Loop ms	Schrittanzahl	Schrittweite mm
10	5	2000	0.05
10	20	500	0.2
50	20	100	1
100	50	20	5

Tabelle 5: Gerade Bewegung - Schrittanzahl und Schrittweite bei $\Delta x = 0; \Delta y = 100mm;$

v mm/s	Loop ms	errechnet mm			Abweichung mm		
		X_t	Y_t	Θ	Δx	Δy	Θ
10	5	0	100.001502	0	0	0.001502	0
10	20	0	99.999618	0	0	0.000382	0
50	20	0	100.000000	0	0	0	0
100	50	0	100.000000	0	0	0	0

Tabelle 6: Ergebnis Gerade Bewegung - Soll: $X_t = 0; Y_t = 100; \Theta = 0;$

7.1.4. Drehung um den Mittelpunkt des Systems

Wenn die Δx-Werte beider Sensoren 0 bleiben und Δy von Sensor A sich im gleichen Maß vergrößert wie sich Δy von Sensor B verkleinert, dreht sich das Tracking-System am Stand (Abbildung 33).

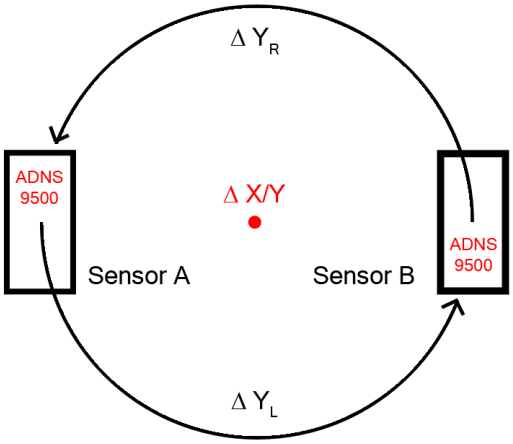


Abbildung 33: Drehung ohne Positionsänderung mit 2 bewegten Sensoren

Um eine Drehung ohne Positionsänderung zu simulieren, messen beide Sensoren keine Bewegung in x-Richtung. Sensor A misst eine negative Bewegung in y-Richtung und Sensor B eine positive Bewegung in y-Richtung. Die Bewegung der Sensoren in y-Richtung erfolgt mit den folgenden Geschwindigkeiten, aus denen sich die Schrittweite ergibt. Die Tabellen 7 und 8 zeigen die verwendeten Input-Werte und Ergebnisse.

v mm/s	Loop ms	Schritte	Schrittweite mm
10	5	1633	0.0499888376175
50	5	327	0.2496384459615
100	5	163	0.5008084161313
10	20	408	0.2000778721309
50	20	82	0.9955094125538
100	20	41	1.9910188251075
10	50	163	0.5008084161313
50	50	33	2.4736900554366
100	50	16	5.1019857393380

Tabelle 7: Drehung um Mittelpunkt des Systems - Schrittanzahl und Schrittweite bei Radius: $SensorA = 12.9921mm; SensorB = 12.9921mm;$

v mm/s	Loop ms	errechnet mm X_t	Y_t	Θ	Abweichung mm Δx	Δy	Θ
10	5	0	0	6.283141	0	0	0.000044
50	5	0	0	6.283184	0	0	0.000001
100	5	0	0	6.283173	0	0	0.000012
10	20	0	0	6.283174	0	0	0.000011
50	20	0	0	6.283188	0	0	0.000003
100	20	0	0	6.283181	0	0	0.000004
10	50	0	0	6.283173	0	0	0.000012
50	50	0	0	6.283184	0	0	0.000001
100	50	0	0	6.283181	0	0	0.000004

Tabelle 8: Ergebnis Drehung um Mittelpunkt des Systems - Soll: $X_t = 0; Y_t = 0; \Theta = 6.283185;$

7.1.5. Drehung um einen Sensor

Wenn Sensor A des Tracking-Systems keine Bewegung misst, während Sensor B eine Bewegung nur in y-Richtung registriert, bewegt sich das Tracking-System offensichtlich auf einer Kreisbahn, mit Sensor A als Mittelpunkt (Abbildung 34). Aufgrund der Eigenheiten des Algorithmus sollte hier eine umso größere Abweichung auftreten je größer Δy von Sensor B wird.

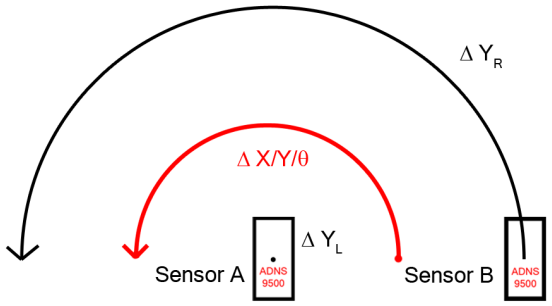


Abbildung 34: Kreisbahn mit einem bewegtem Sensor

Um eine Kreisbewegung mit einem bewegtem Sensor zu simulieren, messen beide Sensoren keine Bewegung in x-Richtung und nur ein Sensor eine Bewegung in y-Richtung. Ursprünglich waren Algorithmustests nur für eine komplette 360° Bewegung geplant.

Da sich aber im Verlauf der Tests herausstellte, dass unabhängig von der Schrittweite und Schrittanzahl, bei einer kompletten Kreisbewegung keine signifikanten Abweichungen auftreten, wurden die Tests (in reduziertem Umfang) auch mit Halbkreisen und Viertelkreisen durchgeführt.

Kreis

Die Tabellen 9 und 10 zeigen die verwendeten Input-Werte und Ergebnisse für eine 360° Drehung.

v mm/s	Loop ms	Schritte	Schrittweite mm
10	5	3265	0.0500041481344
50	5	653	0.2500207406720
100	5	327	0.4992768919230
10	20	816	0.2000778721309
50	20	163	1.0016168322627
100	20	82	1.9910188251075
10	50	327	0.4992768919230
50	50	65	2.5117468255202
100	50	33	4.9473801108732

Tabelle 9: Drehung um einen Sensor - Schrittanzahl und Schrittweite bei Radius: $SensorA = 0mm; SensorB = 25.9842mm;$

v mm/s	Loop ms	errechnet mm			Abweichung mm		
		X_t	Y_t	Θ	Δx	Δy	Θ
10	5	-0.001103	-0.003746	6.283418	0.001103	0.003746	0.000233
50	5	-0.000176	-0.000413	6.283210	0.000176	0.000413	0.000025
100	5	-0.000001	0.000029	6.283184	0.000001	0.000029	0.000001
10	20	-0.000004	0.000164	6.283173	0.000004	0.000164	0.000012
50	20	0.000046	0.000187	6.283173	0.000046	0.000187	0.000012
100	20	-0.000065	0.000001	6.283188	0.000065	0.000001	0.000003
10	50	-0.000001	0.000029	6.283184	0.000001	0.000029	0.000001
50	50	-0.000020	-0.000014	6.283185	0.000020	0.000014	0.000000
100	50	-0.000029	0.000036	6.283184	0.000029	0.000036	0.000001

Tabelle 10: Ergebnis Drehung einen Sensor - Soll: $X_t = 0; Y_t = 0; \Theta = 6.283185;$

Halbkreis

Die Tabellen 11 und 12 zeigen die verwendeten Input-Werte und Ergebnisse für eine 180° Drehung.

v mm/s	Loop ms	Schritte	Schrittweite mm
10	5	815	0.1001616832263
50	5	320	0.2550992869669
100	5	150	0.5442118121961
10	20	82	0.9955094125538
50	20	32	2.5509928696690
100	20	16	5.1019857393380
10	50	2	40.8158859147040

Tabelle 11: Drehung um einen Sensor - Schrittanzahl und Schrittweite bei Radius: $SensorA = 0mm; SensorB = 25.9842mm;$

v mm/s	Loop ms	errechnet mm			Abweichung mm		
		X_t	Y_t	Θ	Δx	Δy	Θ
10	5	-25.98401	-0.04999	3.141610	0.00019	0.04999	0.000017
50	5	-25.98407	-0.12758	3.141580	0.00013	0.12758	0.000013
100	5	-25.98333	-0.27218	3.141585	0.00087	0.27218	0.000008
10	20	-25.98102	-0.49777	3.141594	0.00318	0.49777	0.000001
50	20	-25.96334	-1.27553	3.141592	0.02086	1.27553	0.000001
100	20	-25.90068	-2.55102	3.141590	0.08352	2.55102	0.000003
10	50	-20.40794	-20.40796	3.141591	5.57626	20.40796	0.000002

Tabelle 12: Ergebnis Drehung um einen Sensor - Soll: $X_t = -25.9842; Y_t = 0; \Theta = 3.141593;$

Viertelkreis

Die Tabellen 13 und 14 zeigen die verwendeten Input-Werte und Ergebnisse für eine 90° Drehung.

v mm/s	Loop ms	Schritte	Schrittweite mm
8.16	25	400	0.1020397147868
20.41	25	160	0.2550992869669
43.54	25	75	0.5442118121961
79.64	25	41	0.9955094125538
204.08	25	16	2.5509928696690
408.16	25	8	5.1019857393380
3265.27	25	1	40.8158859147040

Tabelle 13: Drehung um einen Sensor - Schrittanzahl und Schrittweite bei Radius: $SensorA = 0mm; SensorB = 25.9842mm;$

v mm/s	Loop ms	errechnet mm			Abweichung mm		
		X_t	Y_t	Θ	Δx	Δy	Θ
8.16	25	-12.96658	-13.01759	1.570797	0.02552	0.02549	0.000001
20.41	25	-12.92822	-13.05577	1.570797	0.06388	0.06367	0.000001
43.54	25	-12.85557	-13.12769	1.570794	0.13653	0.13559	0.000002
79.64	25	-12.74163	-13.23940	1.570795	0.25047	0.24730	0.000001
204.08	25	-12.34391	-13.61942	1.570795	0.64819	0.62732	0.000001
408.16	25	-11.67483	-14.22584	1.570795	1.31727	1.23374	0.000001
3265.27	25	0.00000	-20.40794	1.570795	12.99210	7.41584	0.000001

Tabelle 14: Ergebnis Drehung um einen Sensor - Soll: $X_t = -12,9921; Y_t = -12,9921; \Theta = 1.570796;$

7.1.6. Drehung um einen externen Punkt

Analog dazu kann eine Kreisbewegung auch mit 2 bewegten Sensoren stattfinden (Abbildung 35). Dazu muss Δy von Sensor A um einen konstanten Faktor von Δy von Sensor B abweichen, während Δx von beiden Sensoren 0 bleibt.

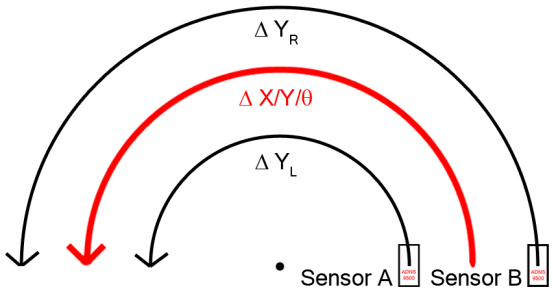


Abbildung 35: Kreisbahn mit 2 bewegten Sensoren

Messen beiden Sensoren keine Bewegung in x-Richtung und um einen gleich bleibenden Faktor unterschiedliche Werte in y-Richtung, bedeutet das eine Kreisbewegung. Hier wurden ebenso wie in 7.1.5 zusätzlich auch Halb- und Viertelkreise getestet.

Kreis

Die Tabellen 15 und 16 zeigen die verwendeten Input-Werte und Ergebnisse für eine 360° Drehung.

v mm/s	Loop ms	Schritte	Schrittweite mm
10	5	6531	0.0499964917038
50	5	1306	0.2500207406720
100	5	653	0.5000414813440
10	20	1633	0.1999553504701
50	20	327	0.9985537838460
100	20	163	2.0032336645254
10	50	653	0.5000414813440
50	50	131	2.4925731856308
100	50	65	5.0234936510405

Tabelle 15: Drehung um einen externen Punkt - Schrittanzahl und Schrittweite bei Radius: $SensorA = 25.9842mm; SensorB = 51.9684mm;$

v mm/s	Loop ms	errechnet mm			Abweichung mm		
		X_t	Y_t	Θ	Δx	Δy	Θ
10	5	-0.01496	-0.00576	6.283559	0.01496	0.00576	0.000374
50	5	0.00294	0.00127	6.283107	0.00294	0.00127	0.000078
100	5	-0.00057	-0.00122	6.283210	0.00057	0.00122	0.000025
10	20	0.00022	0.00211	6.283141	0.00022	0.00211	0.000044
50	20	-0.00004	0.00007	6.283184	0.00004	0.00007	0.000001
100	20	0.00013	0.00056	6.283173	0.00013	0.00056	0.000012
10	50	-0.00057	-0.00122	6.283210	0.00057	0.00122	0.000025
50	50	0.00011	0.00044	6.283175	0.00011	0.00044	0.000010
100	50	-0.00007	-0.00005	6.283185	0.00007	0.00005	0.000000

Tabelle 16: Ergebnis Drehung um einen externen Punkt - Soll: $X_t = 0; Y_t = 0; \Theta = 6.283185;$

Halbkreis

Die Tabellen 17 und 18 zeigen die verwendeten Input-Werte und Ergebnisse für eine 180° Drehung.

v mm/s	Loop ms	Schritte	Schrittweite mm
10	5	320	0.5101985739338
50	5	150	1.0884236243921
100	5	82	1.9910188251075
10	20	32	5.1019857393380
50	20	16	10.2039714786760
100	20	2	81.6317718294079

Tabelle 17: Drehung um einen externen Punkt - Schrittanzahl und Schrittweite bei Radius: $SensorA = 25.9842mm; SensorB = 51.9684mm;$

v mm/s	Loop ms	errechnet mm			Abweichung mm		
		X_t	Y_t	Θ	Δx	Δy	Θ
10	5	-77.95226	-0.38275	3.141580	0.00035	0.38275	0.000013
50	5	-77.94993	-0.81653	3.141585	0.00267	0.81653	0.000008
100	5	-77.94303	-1.49332	3.141594	0.00957	1.49332	0.000001
10	20	-77.89002	-3.82657	3.141592	0.06258	3.82657	0.000001
50	20	-77.70204	-7.65306	3.141590	0.25056	7.65306	0.000003
100	20	-61.22383	-61.22389	3.141591	16.72877	61.22389	0.000002

Tabelle 18: Ergebnis Drehung um einen externen Punkt - Soll: $X_t = -77.9526; Y_t = 0; \Theta = 3.141593;$

Viertelkreis

Die Tabellen 19 und 20 zeigen die verwendeten Input-Werte und Ergebnisse für eine 90° Drehung.

v mm/s	Loop ms	Schritte	Schrittweite mm
8.16	25	400	0.2040794295735
20.41	25	160	0.5101985739338
43.54	25	75	1.0884236243921
79.64	25	41	1.9910188251075
204.08	25	16	5.1019857393380
408.16	25	8	10.2039714786760
3265.27	25	1	81.6317718294079

Tabelle 19: Drehung um einen externen Punkt - Schrittanzahl und Schrittweite bei Radius: $SensorA = 25.9842mm; SensorB = 51.9684mm;$

v mm/s	Loop ms	errechnet mm			Abweichung mm		
		X_t	Y_t	Θ	Δx	Δy	Θ
8.16	25	-38.89972	-39.05277	1.570797	0.07658	0.07647	0.000001
20.41	25	-38.78466	-39.16730	1.570797	0.19164	0.19100	0.000001
43.54	25	-38.56669	-39.38307	1.570794	0.40961	0.40677	0.000002
79.64	25	-38.22488	-39.71819	1.570795	0.75142	0.74189	0.000001
204.08	25	-37.03173	-40.85826	1.570795	1.94457	1.88196	0.000001
408.16	25	-35.02450	-42.67751	1.570795	3.95180	3.70120	0.000001
3265.27	25	0.000000	-61.22383	1.570795	38.97630	22.24753	0.000001

Tabelle 20: Ergebnis Drehung um einen externen Punkt - Soll: $X_t = -38.9763; Y_t = -38.9763; \Theta = 1.570796;$

7.1.7. Interpretation der Ergebnisse der Algorithmustests

Aus den Ergebnissen der Algorithmustests lassen sich mehre Schlüsse ziehen.

Am Arduino-Framework entstehen bei Sinus- und Cosinus-Berechnungen grundsätzlich Rechenfehler in den Nachkommastellen. Das lässt sich gut an den Testergebnissen für eine gerade Strecke ablesen. Theoretisch sollte der Algorithmus hier ohne jegliche Abweichung arbeiten. Allerdings summieren sich die Fehler in den Nachkommastellen bei jedem Schritt. Dadurch wächst die Fehlerrate mit einer höheren Messdichte (und damit kleineren Schritten) sogar, während sie für größere Schrittweiten sinkt. In 7.1.3 zeigt sich, dass bei 100 Rechenschritten bis zu 6. Nachkommastelle noch kein Fehler auftritt. Allerdings treten bei 2000 Schritten bereits Fehler an der 3. Nachkommastelle auf. Insgesamt bleiben die Rechenfehler allerdings vernachlässigbar klein. Da in der Praxis ein Teil der Berechnungen nicht auf dem ACTO sondern im ACTO-Framework auf dem Android-Gerät ausgeführt wird, könnte sich der Fehler weiter reduzieren. Weiters könnten geringe Abweichungen auch durch die, auf 13 Nachkommastellen gerundeten, Schrittweiten bedingt sein.

Die Berechnung des Winkels Θ hat sich, unabhängig von der Schrittweite, als sehr exakt herausgestellt. Die geringen Abweichungen wachsen ebenfalls mit der Schrittanzahl, nicht aber mit der Schrittweite, und legen Fehler bei der Fließkommaberechnung als Ursache nahe. Auch hier treten Fehler an der dritten Kommastelle erst bei bei sehr großen Schrittzahlen auf (siehe 7.1.5 und 7.1.6).

Im Verlauf der Tests stellte sich heraus, dass aufgrund des Algorithmus bei einer Drehung von 360°, unabhängig von der Schrittweite, kaum Abweichungen auftreten. Die Fehler in den Nachkommastellen lassen sich durch die oben beschriebenen Rechenfehler erklären. Auch bei extrem großen Schrittweiten und geringen Schrittzahlen (z.B. 4 für eine 360°Kreisbewegung), traten keine wesentlichen Abweichungen auf.

Da die Simulation eines ganzen Kreises offensichtlich wenig aussagekräftig ist, wurden zusätzlich Viertel- und Halbkreise simuliert. Erst bei diesen Tests zeigten sich die, durch den Algorithmus selbst verursachten und erwarteten, Abweichungen. Bei den Halbkreistests sind die Fehler größer in y-Richtung, während bei Viertelkreisen die Fehler entlang beider Achsen ähnlich ansteigen. Ausgenommen sind hier nur extreme Werte mit nur 1 oder 2 Schritten und damit Schrittweiten zwischen $40mm$ und $80mm$. Hier zeigt sich bei den Halbkreisen ebenfalls ein starker Fehleranstieg entlang der x-Achse. Bei der Simulation von Viertelkreisen gehen bei einer derart geringen Schrittzahl die Fehlergrößen in x- und y-Richtung wieder auseinander. Solche extremen Werte werden aber in der Praxis kaum auftreten. Bei der Verwendung realistischer Testparameter steigt die Genauigkeit wie erwartet mit der Anzahl der Schritte und einer sinkenden Schrittlänge.

Da in der Praxis kaum reine 360°oder 180°Drehungen auftreten werden, erscheinen die Ergebnisse der Viertelkreissimulationen am besten geeignet, um ungefähre Grenzen zu ziehen, bis zu denen der Algorithmus zuverlässige Ergebnisse liefert. Nimmt man als vertretbaren Fehler eine Abweichung von $1mm$ sowohl in x- als auch y-Richtung, sowie eine Schleifendurchlaufzeit von $25ms$ an, ergibt sich für einen Viertelkreis mit einem bewegtem Sensor eine maximale Geschwindigkeit von ca. $300mm/s$ (siehe 7.1.5). Für einen Viertelkreis mit 2 bewegten Sensoren beträgt die Maximalgeschwindigkeit ca.

$100mm/s$ (siehe 7.1.6). Bei diesen Ergebnissen muss allerdings angemerkt werden, dass konstruktionsbedingt die Testdistanz bei einem Viertelkreis mit 2 bewegten Sensoren mindestens doppelt so lang wie bei einem Viertelkreis mit nur einem bewegtem Sensor ist. Des Weiteren bezieht sich die Maximalgeschwindigkeit auf den äußeren und damit schnelleren Sensor und nicht auf das Gesamtsystem.

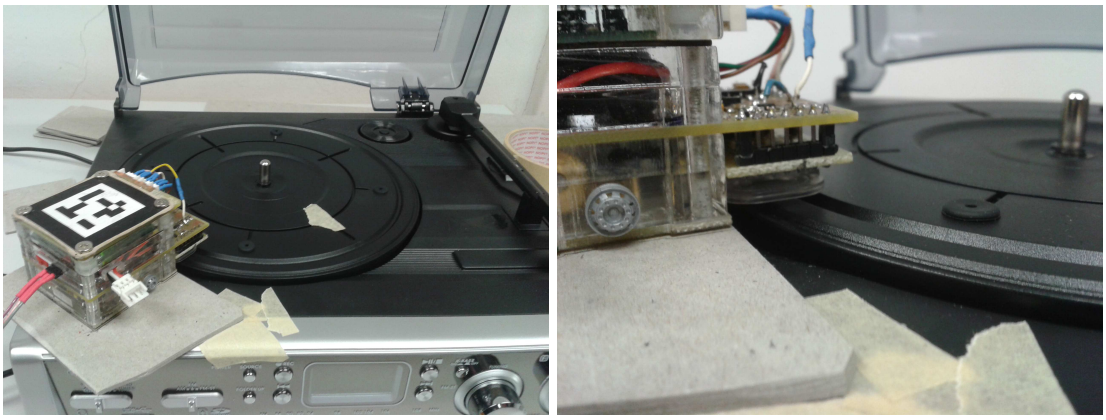
Eine Fehlerquelle stellt die Berechnung der ACTO-Position (und damit der Markerposition) relativ zur Position des Tracking-Systems dar. Dieser Fehler dürfte allerdings nur durch nummerische Fehler durch die Gleitkommadarstellung entstehen, sich nicht aufsummieren und daher vernachlässigbar gering bleiben.

Aus den Testergebnissen lässt sich folgern, dass bei einer angenommenen Schleifendurchlaufzeit von $25ms$, der Algorithmus für die Anwendung in einem Tracking-System in Arduino-Größe (größere Geschwindigkeiten sind hier nicht zu erwarten) ausreichend exakt und geeignet ist. Über die Genauigkeit des Gesamtsystems lassen die Algorithmustests alleine noch keine Aussage zu, da davon auszugehen ist, dass die Messungen der optischen Sensoren ebenfalls mit einem Fehler behaftet sind (siehe 7.3.1).

7.2. Konsistenz-Tests

Dieser Abschnitt untersucht, ob die Bewegungsgeschwindigkeit des Tracking-Systems oder der Akkuladestand einen Einfluss auf die Genauigkeit der Sensoren haben.

In der gängigen Praxis werden zu diesem Zweck teilweise aufwändige und teure Konstruktionen eingesetzt, beispielsweise mit einem Roboterarm [42] oder einer vergleichbaren Vorrichtung, mit einer Führungsschiene und Servos, die das System mit reproduzierbaren Parametern bewegen. Um eine konstante Bewegungsgeschwindigkeit zu gewährleisten wurde stattdessen für diese Tests ein *PL-520USB* Plattenspieler [43] von *Soundmaster* benutzt (Abbildung 36) dessen Rotationsgeschwindigkeit vor den Tests überprüft wurde (siehe unten).



(a) (b)

Abbildung 36: (a) PL-520USB mit Markierung und ACTO, (b) Messbereich des ACTOs am PL-520USB

Dabei bewegt sich allerdings nicht das Tracking-System sondern der Untergrund. Gemessen wird dabei nicht über eine fixe Entfernung, sondern über eine festgelegte Zeit. Da der Radius nicht genau genug bekannt ist (und die exakte Ausrichtung des Systems eine gravierende Rolle spielt), lassen sich die absolute Bewegungsweite und Genauigkeit nicht ermitteln. Da aber die Rotationsgeschwindigkeit bekannt ist und das System zwischen den Tests nicht bewegt wird, kann auf das Verhältnis der Distanzen zueinander rückgeschlossen werden. 15 Sekunden bei $40rpm$ entsprechen der doppelten Distanz von 15 Sekunden bei $20rpm$. Damit lässt sich nicht die Genauigkeit, aber sehr wohl die Konsistenz - abhängig von der Geschwindigkeit oder der Akkuleistung - überprüfen.

Der Plattenspieler besitzt drei verschiedene Geschwindigkeitseinstellungen: $33.33rpm$, $45rpm$ und $78rpm$. Um zu überprüfen ob diese Angaben exakt der Realität entsprechen, wurde mit einer Stoppuhr und einem Marker auf dem Teller die Rotationsgeschwindigkeit gemessen. Dabei ergab sich eine leichte, aber konstante Abweichungen in der Geschwindigkeit (siehe Tabelle 21). Mit den so gemessenen Rotationsgeschwindigkeiten wurden die Tests durchgeführt.

rpm Soll	rpm gemessen
33.33	34.75
45	44.69
78	80.32

Tabelle 21: Plattenspieler Geschwindigkeit Soll und gemessen

7.2.1. Geschwindigkeit

Wenn man das Tracking-System entlang eines fixen Pfades mit der Hand bewegt, kann die Geschwindigkeit des Tracking-Systems schwanken. Um den Einfluss der Geschwindigkeit auf die Genauigkeit des Tracking-Systems zu messen, war es notwendig Tests mit verschiedenen definierten Geschwindigkeiten über eine gleich bleibende Entfernung durchzuführen.

Dafür wurde der Plattenspieler mit unterschiedlichen Geschwindigkeiten betrieben und die gemessene Strecke pro Umdrehung ausgewertet. Es wurden pro Geschwindigkeitsstufe 16 Durchläufe durchgeführt und der Mittelwert ausgewertet. Bei den Ergebnissen ist nicht die absolut gemessene Strecke relevant, da die wirklich zurückgelegte Entfernung nicht präzise gemessen werden kann. Wichtig ist die Schwankungsbreite zwischen den Ergebnissen.

rpm	Linker Sensor mm		Rechter Sensor mm	
	X	Y	X	Y
34.75	-48.43 (SD=0.02)	-7.26 (SD=0.01)	-47.81 (SD=0.02)	7.64 (SD=0.01)
44.69	-48.99 (SD=0.04)	-7.22 (SD=0.01)	-47.56 (SD=0.03)	7.94 (SD=0.01)
80.32	-47.18 (SD=0.06)	-6.64 (SD=0.01)	-48.71 (SD=0.05)	7.64 (SD=0.01)

Tabelle 22: Konsistenztest Geschwindigkeit

Von Tabelle 22 und Grafik 37 lässt sich ablesen, dass die gemessenen Werte zwar leicht schwanken, aber keinen eindeutigen Trend in eine bestimmte Richtung aufweisen. Daraus lässt sich schließen, dass die Geschwindigkeit des Systems keinen systematischen Fehler induziert. Da, aufgrund der Ausrichtung der Sensoren über dem Plattenspieler, vom Tracking-System eine Kreisbewegung gemessen wird, ist der y-Wert des rechten Sensors positiv, während alle anderen Werte negativ sind. Um die Ergebnisse kompakt in einem Diagramm (Abbildung 37) darstellen zu können, wurden die y-Werte des rechten Sensors invertiert (also mit -1 multipliziert).

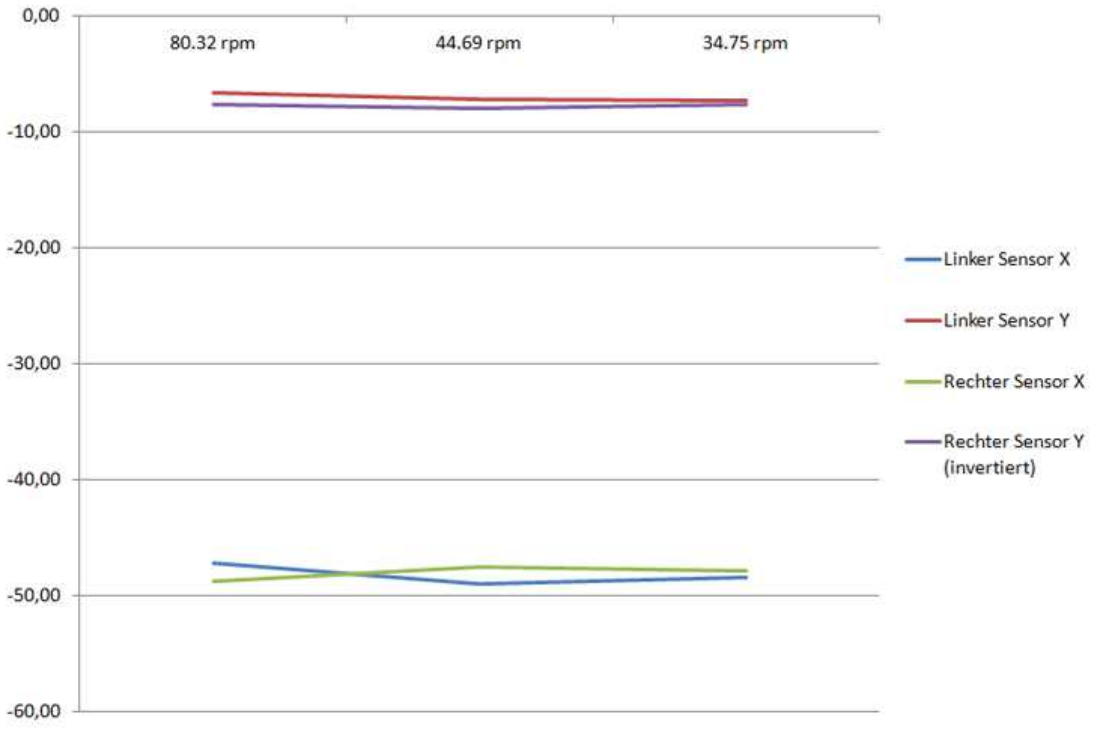


Abbildung 37: Veränderung abhängig von der Geschwindigkeit

7.2.2. Akkuleistung

Um festzustellen ob der Ladezustand des Akkus einen relevanten Einfluss auf die Messgenauigkeit hat und einen systematischen Fehler verursacht, wurde die gemessene Strecke bei unterschiedlichen Ladegraden und ansonsten gleichbleibenden Parametern ausgewertet. Das Tracking-System wurde dafür über dem Plattenspieler, der mit einer Geschwindigkeit von $44.69rpm$ rotiert, fixiert (siehe 7.2). Für die Auswertung wurden die Mittelwerte und Standardabweichungen über jeweils 16 Durchgänge, der jeweils über 5 Sekunden zurückgelegten Strecke, herangezogen. In diesen 5 Sekunden wurde, bei insgesamt 80 Durchgängen, der Chip in 77 Fällen 200 mal und in 3 Fällen nur 199 mal ausgelesen. Dies ist durch die minimal schwankende Schleifendurchlaufzeit zu erklären.

Akku	Linker Sensor mm	
	X	Y
100%	-2199.94 (SD=3.09)	-315.12 (SD=1.13)
80%	-2185.86 (SD=4.34)	-313.78 (SD=0.93)
60%	-2195.66 (SD=3.01)	-318.33 (SD=0.65)
40%	-2186.35 (SD=3.54)	-314.62 (SD=1.55)
20%	-2186.06 (SD=2.74)	-303.33 (SD=1.22)

Tabelle 23: Konsistenztest Akkuladung: linker Sensor

Akku	Rechter Sensor mm	
	X	Y
100%	-2135.99 (SD=2.07)	363.09 (SD=1.18)
80%	-2140.73 (SD=4.56)	358.48 (SD=1.42)
60%	-2138.91 (SD=1.81)	356.73 (SD=1.25)
40%	-2142.83 (SD=2.84)	357.26 (SD=1.16)
20%	-2140.81 (SD=2.64)	368.00 (SD=1.22)

Tabelle 24: Konsistenztest Akkuladung: rechter Sensor

Die Werte aus den Tabellen 23 und 24 wurden für eine bessere Vergleichbarkeit grafisch dargestellt. Aus den Tabellen und den folgenden Diagrammen ist ersichtlich, dass die gemessene Entfernung abhängig von der Akkuleistung entlang der x-Achse (bei der die größte Bewegung gemessen wurde), um nicht einmal 1 Prozent schwankt. Entlang der y-Achse ist die Schwankungsbreite etwas größer (ca 3% - 4%). Insgesamt ist aber anhand der Grafiken (Abbildung 38 und 39) kein systematischer Fehler aufgrund des Ladezustands des Akkus erkennbar. Die Abweichungen erscheinen zufällig und innerhalb einer geringen Schwankungsbreite.

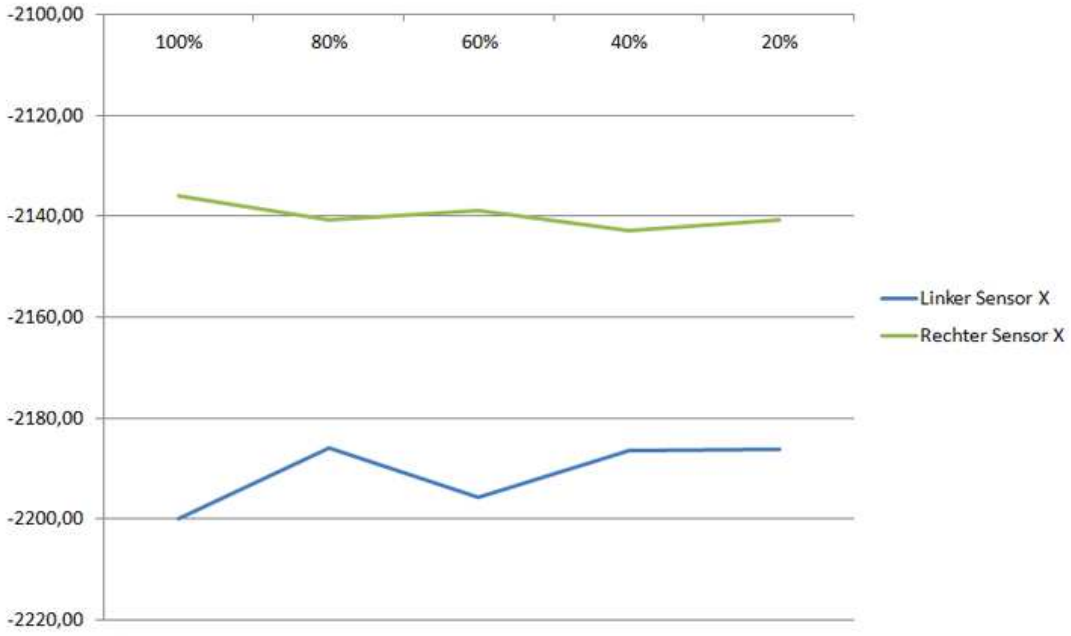


Abbildung 38: Veränderung abhängig von der Akkuleistung - x-Richtung

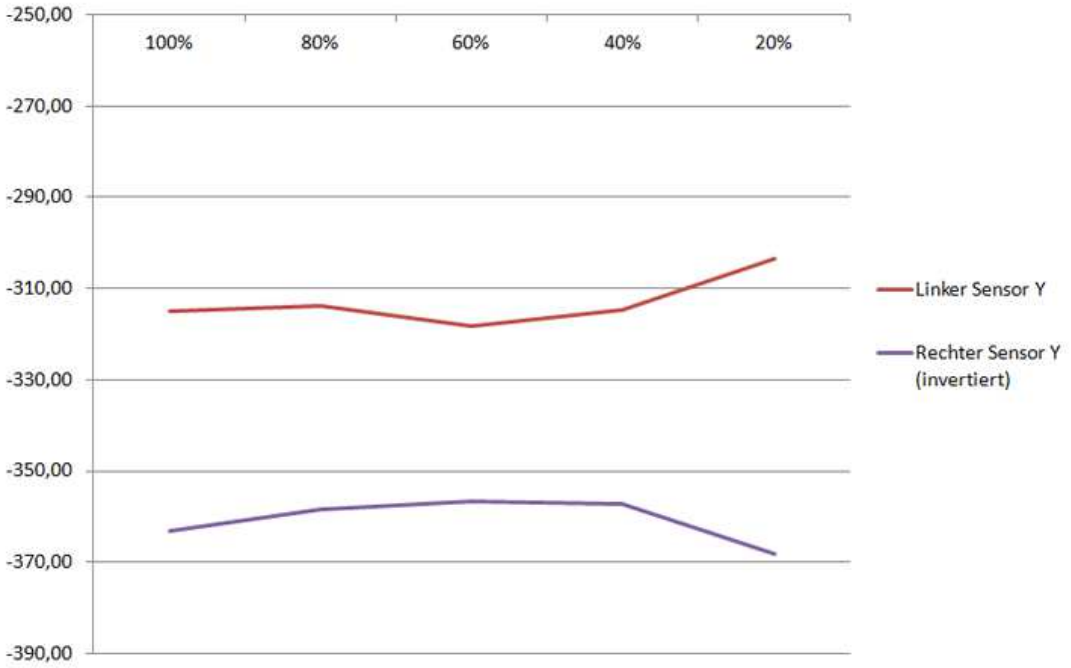


Abbildung 39: Veränderung abhängig von der Akkuleistung - y-Richtung (die y-Werten des rechten Sensors sind auch hier invertiert dargestellt)

7.3. Performance Tests

Um die Genauigkeit der Sensoren in der Praxis einschätzen zu können, werden Tests mit reproduzierbaren Bewegungsmustern benötigt. Dafür wird das Tracking-System entlang einer vorgegebenen Linie verschoben.

Es werden sowohl das Tracking-System als Ganzes, als auch die Sensoren gesondert getestet. Bei den Sensortests wird auf die Rohdaten der Sensoren zurückgegriffen. Das Resultat sind 4 Werte: Für beide Sensoren jeweils die zurückgelegte Distanz in x- und y-Richtung. Somit wird genau genommen anstatt der Endposition die zurückgelegte Strecke gemessen. Da keine Rotation berechnet wird, werden die Sensortests nur auf einer geraden Strecke durchgeführt. Dabei wird das Tracking-System sowohl vorwärts, als auch rückwärts bewegt, danach um 90° gedreht und erneut in beide Richtungen verschoben. Jede dieser 4 Bewegungen wird 5 mal wiederholt. Die Ergebnisse dieser Tests lassen Rückschlüsse auf die Ausrichtung der Sensoren innerhalb des Tracking-Systems zu.

Bei den Systemtests hingegen wird die Position des Tracking-Systems (relativ zum Ausgangspunkt) aus den Sensordaten mit Hilfe des Algorithmus berechnet. Hier wird die Position auf die gleiche Art wie auch beim Einsatz in der Praxis berechnet. Man erhält 3 Werte: X/Y (Koordinaten der Endposition) und Θ (Rotation). Der Ausgangspunkt wird mit den Koordinaten 0/0 angenommen. Die Rotation beträgt am Ausgangspunkt ebenfalls 0. Es kommen 2 verschiedene Bewegungsmuster zum Einsatz: eine Bewegung entlang einer geraden Linie und eine Bewegung entlang eines Viertelkreises. Das Tracking-System wird wie bei den Sensortests sowohl vorwärts, als auch rückwärts bewegt, danach um 90° gedreht und erneut in beide Richtungen verschoben. Jede dieser 4 Bewegungen wird 5 mal wiederholt.

Durch den manuellen Einbau der Sensoren können Ungenauigkeiten in der Ausrichtung der Sensoren nicht ausgeschlossen werden. Außerdem weisen die Sensoren, abhängig von der Beschaffenheit des Untergrunds und des Sensors, ebenfalls eine Abweichung auf. Die Art und das Ausmaß dieser Messfehler lassen sich mit den Sensortests abschätzen. Für einen produktiven Einsatz müssen diese Abweichungen möglichst vollständig durch die Kalibrierung kompensiert werden. Daher werden bei den Sensor- und Systemtests, die Ergebnisse sowohl mit, als auch ohne Kalibrierung mit den Sollwerten verglichen. So lässt sich die Bedeutung der Kalibrierung abschätzen. Für eine genaue Beschreibung des Kalibrierungsvorganges siehe 6.3.

7.3.1. Sensortests

Das Tracking-System wird entlang einer Schiene jeweils in x-, negativer x-, y- und negativer y-Richtung mit einer Geschwindigkeit von ca. $100mm/s$ über eine Distanz von $785,4mm$ verschoben (Abbildung 40). Die exakte Länge ergibt sich aus dem Radius von $500mm$ der Kreisbewegung (siehe 7.3.2). Gemessen wurde die zurückgelegte Distanz beider Sensoren in mm. Um einen Durchschnitt und eine Standardabweichung zu erhalten wurde jeder Testlauf 5 mal durchgeführt (siehe Tabellen 25, 27, 26 und 28).

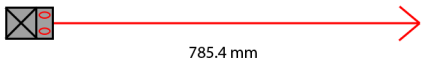


Abbildung 40: Bewegung entlang einer geraden Linie - 785.4mm

Soll		Ist		Ist	
X	Y	X	X_c	Y	Y_c
785.4	0	815.75	786.60	-11.60	-13.91
-785.4	0	-815.26	-785.93	12.41	15.10
0	785.4	10.62	5.82	807.21	785.60
0	-785.4	-10.23	-5.72	-805.14	-786.11

Tabelle 25: Bewegung entlang gerader Linie - Linker Sensor; c = kalibriert

Soll		Ist_{SD}		Ist_{SD}	
X	Y	X	X_c	Y	Y_c
785.4	0	2.19	1.68	0.54	1.56
-785.4	0	2.54	0.21	0.48	0.43
0	785.4	0.36	1.05	2.16	1.60
0	-785.4	0.98	1.12	2.42	0.65

Tabelle 26: Standardabweichung entlang gerader Linie - Linker Sensor; c = kalibriert

Soll		Ist		Ist	
X	Y	X	X_c	Y	Y_c
785.4	0;	777.27	789.80	8.42	3.70
-785.4	0;	-777.47	-787.16	-8.51	-4.81
0	785.4;	10.28	4.98	778.71	786.40
0	-785.4;	-9.54	-3.54	-778.53	-787.73

Tabelle 27: Bewegung entlang gerader Linie - Rechter Sensor; c = kalibriert

Soll		Ist_{SD}		Ist_{SD}	
X	Y	X	X_c	Y	Y_c
785.4	0	3.78	2.17	0.88	2.63
-785.4	0	4.71	1.09	1.95	0.94
0	785.4	0.84	1.31	2.08	1.57
0	-785.4	1.01	3.18	3.44	2.47

Tabelle 28: Standardabweichung entlang gerader Linie - Rechter Sensor; c = kalibriert

In Tabelle 29 sind die durchschnittlichen Abweichungen der gemessenen Strecke vom Sollwert in Prozent aufgeführt. In der Spalte *raw* finden sich die nicht kalibrierten und daneben die kalibrierten Werte der Sensoren. Die kalibrierten Werte sind um den Sollwert gestreut, während die unkalibrierten Werte alle größer, beziehungsweise kleiner sind. Deswegen wurden für die Ergebnisse der kalibrierten Sensoren sowohl die durchschnittliche Abweichung (*kalibriert*), als auch der durchschnittliche Betrag der Abweichungen ($kalibriert_{abs}$) angegeben. Die Werte für die Abweichung von der Solldistanz $0mm$ (also 90° zur relevanten Messrichtung von 785.4mm) werden nicht berücksichtigt, da sich einerseits keine Abweichung in Prozent berechnen lässt und diese Werte andererseits auch wenig aussagekräftig sind. Diese Werte sagen mehr über die Ausrichtung der Sensoren im Testsystem, als über die Genauigkeit der Entfernungsmessung aus.

Richtung	Linker Sensor			Rechter Sensor		
	raw	$kalibriert_{abs}$	*kalibriert*	*raw*	$kalibriert_{abs}$	*kalibriert*
X	3.86%	0.24%	0.15%	-1.03%	0.56%	0.56%
-X	3.80%	0.07%	0.07%	-1.01%	0.22%	0.22%
Y	2.78%	0.16%	0.03%	-0.85%	0.21%	0.13%
-Y	2.51%	0.11%	0.09%	-0.87%	0.38%	0.30%

Tabelle 29: Abweichung in Prozent von Solldistanz ($785.4mm$)

Die Tabelle zeigt, dass die Genauigkeit der Sensoren, durch eine von der Oberfläche abhängige Kalibrierung, drastisch gesteigert werden kann. Die durchschnittliche Abweichung konnte von 0.85 bis 3.86 auf 0.03 bis 0.56 Prozent gesenkt werden. Auch für jede Messachse und Messrichtung alleine stellt Kalibrierung eine Verbesserung dar. Deswegen ist die Kalibrierung für den Einsatz in der Praxis unbedingt zu empfehlen.

7.3.2. Systemtests

Bei den Systemtests werden nicht die einzelnen Messwerte der Sensoren, sondern die daraus errechneten Positionsdaten zur Auswertung herangezogen.

Bewegung entlang einer geraden Linie 300mm

Bei diesem Test wird das Tracking-System entlang einer gerade Linie für genau $300mm$, mit einer Geschwindigkeit von ca. $100mm/s$ bewegt (Abbildung 41). Die Bewegung erfolgt exakt entlang der x- beziehungsweise y-Achsen, in alle 4 Richtungen. Daher betragen die Sollwerte einer Achsenrichtung und der Rotation immer 0, während entlang der anderen Achse genau $300mm$ gemessen werden sollten. Die Abweichung wird für unkalibrierte und kalibrierte Sensoren angegeben. Durchgeführt werden pro Richtung 5 Testläufe deren Mittelwert dann für die Auswertung verwendet wird.

Abbildung 41: Bewegung entlang einer geraden Linie - $300mm$

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	300	302.90 (SD=2.02)	300.73 (SD=0.33)	2.90	0.73
Y	0	-31.50 (SD=9.96)	-25.02 (SD=3.68)	-31.50	-25.02
Θ	0	0.2262 (SD=0.05)	0.1851 (SD=0.02)	0.2262	0.1851

Tabelle 30: Bewegung in x-Richtung $300mm$

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	-300	-303.48 (SD=0.48)	-299.74 (SD=2.75)	-3.48	0.26
Y	0	-35.63 (SD=2.71)	-27.62 (SD=7.86)	-35.63	-27.62
Θ	0	-0.2164 (SD=0.02)	-0.1706 (SD=0.05)	-0.2164	-0.1706

Tabelle 31: Bewegung in neg. x-Richtung $300mm$

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	0	42.74 (SD=4.17)	-18.84 (SD=3.13)	42.74	-18.84
Y	300	299.85 (SD=1.53)	300.97 (SD=0.10)	-0.15	0.79
Θ	0	0.3071 (SD=0.04)	-0.1212 (SD=0.01)	0.3071	-0.1212

Tabelle 32: Bewegung in y-Richtung $300mm$

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	0	45.13 (SD=5.51)	-14.00 (SD=4.74)	45.13	-14.00
Y	-300	-299.33 (SD=0.90)	-299.72 (SD=1.51)	0.67	0.28
Θ	0	-0.3780 (SD=0.19)	0.0878 (SD=0.02)	-0.3780	0.0878

Tabelle 33: Bewegung in neg. y-Richtung $300mm$

Die Ergebnisse in den Tabellen 30, 31, 32 und 33 zeigen, dass die Kalibrierung (mit einer Ausnahme) zu geringeren Abweichungen und damit einer verbesserten Genauigkeit führt. Die Abweichungsdistanz entlang der x-Achse als Hauptbewegungsrichtung konnte um 75% bzw. 93% reduziert werden. Die Abweichungsdistanz entlang der y-Achse als Hauptbewegungsrichtung in negativer Richtung wurde um 58% reduziert. In positiver Richtung entlang der y-Achse als Hauptbewegungsrichtung verschlechterte sich das Ergebnis hingegen.

Die Rotation sollte eigentlich durchgehend 0 betragen. Kleine Ungenauigkeiten bei der Messung in y-Richtung ergeben jedoch eine Drift der Rotation. Dieser Fehler summiert sich mit der Zeit und verursacht eine Abweichung 90° zur Hauptbewegungsrichtung. Das Tracking-System misst also eine Kurve. Die Abweichung der Rotation konnte ebenfalls durch die Kalibrierung bedeutend reduziert werden. Im Falle der Bewegung entlang der y-Achse wurde sie sogar etwas überkompensiert.

Diese Reduzierung des Rotationsfehlers verringert auch die Abweichung 90° zur Hauptbewegungsachse. Die Verbesserung fällt im Vergleich zur Hauptbewegungsachse geringer aus, bleibt aber bedeutend.

Bewegung entlang einer geraden Linie 785.4mm

Dieser Tests ist gleich aufgebaut wie der vorhergehende. Statt $300mm$ wird das Tracking-System allerdings $785.4mm$ bewegt (Abbildung 42). Die exakte Entfernung ergibt sich durch die Länge des Viertelkreises in diesem Kapitel weiter oben.

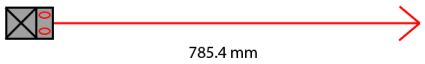


Abbildung 42: Bewegung entlang einer geraden Linie - 785.4mm

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	785.4	750.09 (SD=7.67)	736.13 (SD=15.18)	-35.31	-49.27
Y	0	-216.80 (SD=10.95)	-236.54 (SD=35.73)	-216.80	-236.54
Θ	0	0.5785 (SD=0.03)	0.6079 (SD=0.07)	0.5785	0.6079

Tabelle 34: Bewegung in x-Richtung 785.4mm

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	-785.4	-760.14 (SD=18.79)	-762.76 (SD=4.61)	25.26	22.64
Y	0	-201.59 (SD=53.28)	-168.49 (SD=16.06)	-201.59	-168.49
Θ	0	-0.5297 (SD=0.11)	-0.4598 (SD=0.02)	-0.5297	-0.4598

Tabelle 35: Bewegung in neg. x-Richtung 785.4mm

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	0	345.07 (SD=35.25)	-61.20 (SD=33.47)	345.07	-61.20
Y	785.4	676.48 (SD=31.21)	781.73 (SD=1.76)	-108.92	-3.67
Θ	0	0.9789 (SD=0.13)	-0.1387 (SD=0.07)	0.9789	-0.1387

Tabelle 36: Bewegung in y-Richtung 785.4mm

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	0	331.46 (SD=26.53)	-58.64 (SD=43.72)	331.46	-58.64
Y	-785.4	-688.24 (SD=17.51)	-784.24 (SD=3.49)	97.16	1.16
Θ	0	-0.9129 (SD=0.07)	0.1190 (SD=0.06)	-0.9129	0.1190

Tabelle 37: Bewegung in neg. y-Richtung 785.4mm

Die Ergebnisse in den Tabellen 34, 35, 36 und 37 zeigen für die Bewegung entlang der x-Achse keine relevante Verbesserung. In positiver Richtung sind die kalibrierten Werte etwas genauer, in negativer Richtung jedoch etwas ungenauer, als ihre unkalibrierten Entsprechungen.

Für Bewegungen entlang der y-Achse zeigt sich jedoch ein gänzlich anderes Bild. Hier wird die Positionsbestimmung durch die Kalibrierung in beiden Richtungen wesentlich genauer. Die Abweichung verringert sich hierbei sowohl bei der Rotation, als auch entlang der x- und y-Achse.

Bewegung entlang eines Viertelkreises

Das Tracking-System wird entlang eines Viertelkreises, mit einer Geschwindigkeit von ca. $100mm/s$ bewegt (Abbildung 43). Der Abstand vom Mittelpunkt des Kreises zum Mittelpunkt des Tracking-Systems definiert sich als $r = 500mm$. Daraus ergibt sich, wie weiter oben beschrieben, eine Bewegungslänge von $785.4mm$.

Das System wird dabei entlang seiner x- und y-Achse, jeweils 5 mal vor und 5 mal zurück bewegt und dabei um 90° gedreht. Daraus ergibt sich in Weltkoordinaten eine Bewegung entlang eines Viertelkreises.

Gemessen wird der Radius vom Kreismittelpunkt zum Markermittelpunkt, welcher auch gleichzeitig den Nullpunkt des Tracking-Systems darstellt. Um eine präzise und gleichmäßige Kreisbewegung zu erhalten, wird keine Schiene verwendet. Stattdessen wird das Tracking-System an einer, um einen Angelpunkt schwenkbaren Stange, fixiert (siehe 5.2.2).

In den folgenden Tabellen bezeichnen die Sollwerte Koordinaten im Weltkoordinatensystem.

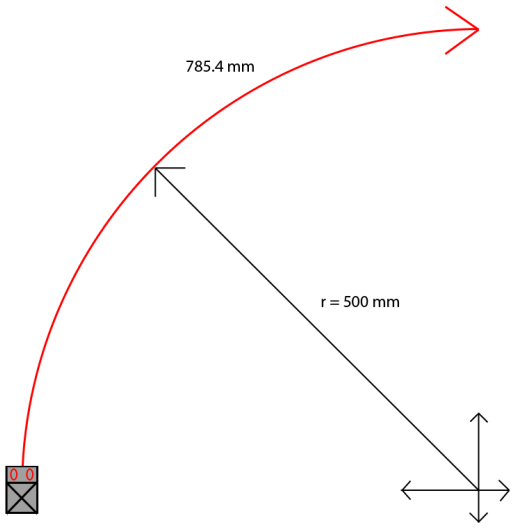


Abbildung 43: Bewegung entlang eines Viertelkreises

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	500	525.58 (SD=41.89)	540.87 (SD=27.42)	25.58	40.87
Y	-500	-536.33 (SD=13.73)	-521.92 (SD=14.21)	-36.33	-21.92
Θ	1.5708	1.6271 (SD=0.12)	1.5957 (SD=0.09)	0.0563	0.0249

Tabelle 38: Bewegung in x-Richtung 785.4mm

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	-500	-536.05 (SD=18.75)	-521.07 (SD=23.25)	-36.05	-21.07
Y	-500	-534.82 (SD=13.49)	-544.61 (SD=17.96)	-34.82	-44.61
Θ	-1.5708	-1.5530 (SD=0.04)	-1.5897 (SD=0.05)	0.0178	-0.0189

Tabelle 39: Bewegung in neg. x-Richtung 785.4mm

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	-500	-218.98 (SD=37.86)	-532.46 (SD=10.59)	281.02	-32.46
Y	500	-721.19 (SD=22.24)	-440.62 (SD=21.27)	221.19	-59.38
Θ	-1.5708	-0.5189 (SD=0.09)	-1.6872 (SD=0.10)	1.0519	-0.1164

Tabelle 40: Bewegung in y-Richtung 785.4mm

	Soll	raw	kalibriert	Abw. raw	Abw. kalibriert
X	-500	-204.75 (SD=40.46)	-499.37 (SD=10.96)	295.25	0.63
Y	-500	-735.52 (SD=23.71)	-486.72 (SD=19.75)	-235.52	13.28
Θ	1.5708	0.5940 (SD=0.14)	1.6747 (SD=0.07)	-0.9768	0.1039

Tabelle 41: Bewegung in neg. y-Richtung 785.4mm

Ähnlich wie bei der geraden Bewegung über 785.4mm zeigt sich anhand der Tabellen 38, 39, 40 und 41, dass es bei Bewegungen entlang der x-Achse des Tracking-Systems durch die Kalibrierung zu keiner relevanten Verbesserung der Genauigkeit kommt. Andererseits wird die Abweichung, bei der Verschiebung entlang der y-Achse des Tracking-Systems, erheblich reduziert.

Die Rotation wird ausschließlich durch die Messungen der Sensoren entlang der y-Achse des Systems berechnet. Die Bewegung entlang der x-Achse des Systems spielt keine Rolle. Auffällig ist daher, dass die Rotation (und damit der gesamte Bewegungspfad) wesentlich stärker abweicht, wenn die Hauptbewegung entlang der x-Achse des Systems stattfindet und deswegen in y-Richtung nur sehr kleine Messwerte zu erwarten sind.

8. Weitere Verbesserungsmöglichkeiten

Ziel dieser Arbeit ist es, anhand eines Prototyps die prinzipielle Eignung von optischen Maussensoren zum Tracking von ACTOs zu zeigen. Dieses Kapitel befasst sich mit möglichen Verbesserungen, durch eine Erweiterung der Testverfahren und eine Überarbeitung der Hard- und Software.

8.1. Weitere Tests

Da der Umfang einer Diplomarbeit begrenzt ist, konnte im Verlauf der Tests nur auf die wichtigsten Einflussfaktoren eingegangen werden. Aufwändigere Testverfahren würden helfen die Fehlerquellen, die zu der beobachteten Ungenauigkeit führen, weiter einzugrenzen.

- Für ADNS 9500-Sensoren kann eine Lift-Detection von $1mm - 5mm$ festgelegt werden. Überschreitet der Abstand der Linse zum Untergrund diesen Wert, misst der Sensor keine Bewegung. Obwohl der Wert definiert werden kann, schwankt die reale Entfernungsschwelle aufgrund der unterschiedlichen Oberflächenbeschaffenheit [4]. Für das Tracking wurde die Lift-Detection auf die maximale Entfernung gestellt. Theoretisch sollte es bei Schwankungen der Distanz zur Oberfläche unterhalb dieses festgelegten Wertes, zu keinen systematischen Messungenauigkeiten kommen. Zusätzliche Tests wären notwendig um diese Annahme zu bestätigen.

- Optische Maussensoren erlauben eine Zeigersteuerung, abhängig von der Oberfläche mit schwankender Qualität. Auf Oberflächen wie Glas liefern die meisten Sensoren keine validen Messwerte. Mit weiteren Tests wäre zu klären, welche Oberflächen besonders gut oder besonders schlecht geeignet sind. Des Weiteren wurde die Kalibrierung für eine bestimmte Oberfläche vorgenommen. Damit müsste der Einfluss der Oberflächenbeschaffenheit auf die Kalibrierung ebenfalls verifiziert werden.

- Da bei den Performance-Tests für jedes Bewegungsmuster eine eigene Führungskonstruktion entworfen werden musste, um eine konstante Testumgebung zu schaffen, wurde auf die Evaluierung komplizierterer Bewegungsmuster verzichtet. Bewegungsmuster abseits von geraden Linien und exakten Kreisen, erwiesen sich mit den vorhandenen Mitteln als nicht ausreichend exakt reproduzierbar. Trotzdem wären ausführlichere Tests in diesem Gebiet wünschenswert. Ein frei programmierbarer und präziser Roboterarm wäre zum Beispiel eine Möglichkeit weitere Bewegungsmuster zu simulieren.

8.2. Hardwareverbesserungen

Der Prototyp des Tracking-Systems hat einen evolutionsartigen Prozess durchlaufen. Beim Design jeder neuen Version flossen die gesammelten Erfahrungen der vorigen Stufe mit ein. In diesem Abschnitt werden weitere mögliche Verbesserungen erläutert. Dazu gehören auch solche Verbesserungen, die mit der aktuellen Konstruktionsweise des ACTOs nicht kompatibel wären.

- Aus Zeit und Kostengründen wurde das System in Handarbeit gefertigt. Das bedeutet die Platine wurde manuell geätzt, zugeschnitten und gebohrt. Alle Bauteile wurden von Hand aufgelötet. Dadurch ergibt sich zwangsläufig eine Ungenauigkeit in der Positionierung und Ausrichtung der Sensoren. Im Algorithmus kann diese Ungenauigkeit nicht hinreichend kompensiert werden, da die tatsächliche Position der Bauteile nicht exakt genug bekannt ist. Eine Herstellung des Systems durch einen Auftragsfertiger wäre vor allem für kleine Stückzahlen wesentlich teurer. Allerdings würde diese Vorgehensweise die Abweichungen in der Hardware minimieren. Alternativ wäre auch eine exaktere Vermessung des bestehenden Systems mittels Laserscan denkbar.

- Die meisten optischen Maussensoren (wie auch der hier benutzte ADNS 9500) funktionieren nicht auf transparenten Materialien wie Glas. Deswegen kann das ACTO mit dem Tracking-System nicht, wie bisher durch, eine transparente Platte mit einem Marker getrackt werden. Da ein absolutes Tracking aber weiterhin notwendig bleibt, muss der Marker auf der Oberseite des ACTOs angebracht werden. Dadurch können viele Erweiterungen, die sich ebenfalls auf der Oberseite befinden würden, nicht verwendet werden. Die Verwendung von Lasersensoren die auch auf Glas funktionieren (zum Beispiel der ADNS 8020 [35]) würde das Tracking mittels Marker auf der Unterseite und damit flexiblere Extensions zulassen. Allerdings würde damit immer noch ein Glastisch für den Einsatz benötigt werden.

- In Abschnitt 7.3 zeigt sich, dass eine Bewegung entlang einer geraden Linie über $785.4mm$ entlang der x-Achse der Sensoren, zu größeren Abweichungen bei der Rotation führt, als entlang der y-Achse. Aus diesem Grund, und weil bei nur 2 Sensoren keine Fehler durch Redundanz erkannt und kompensiert werden können, würden zusätzlichen Sensoren, um 90° gedreht, die Genauigkeit steigern. Nachteilig würden sich die höheren Kosten, der steigende Platzverbrauch und kürzere Akkulaufzeiten auswirken.

- Um eine Fehlerquelle auszuschließen wurden, wie im Datenblatt vorgesehen, passende Linsen für den Sensor verwendet. Eventuell sind diese nicht zwangsläufig notwendig. Es wäre zu klären, inwiefern sich ein Weglassen der Linsen auf die Präzision oder die vertikale Reichweite der Sensoren auswirkt. Der Vorteil wäre ein einfacheres Design und ein verringerter Platzverbrauch. Allerdings würde die Lift-Detection nicht mehr wie vorgesehen funktionieren. Außerdem stellen die Linsen auch einen gewissen Schutz der Sensoren gegenüber einer mechanischen Einwirkung von außen dar.

- Die Anordnung der Sensoren wird weitgehend durch das ACTO-Design bestimmt. Aus Platzmangel mussten die Sensoren an der Außenseite befestigt werden. Außerdem konnte der Abstand maximal die Breite des ACTOs betragen. Für das Tracking wäre ein größerer Abstand zwischen den Sensoren (zum Beispiel ein Sensor auf jeder Seite) von Vorteil. Eine andere Herangehensweise wäre, die Sensoren noch platzsparender im Gehäuse selbst zu integrieren. Dafür müsste allerdings die

Antriebseinheit neu organisiert werden. Außerdem wäre der dann noch geringere Abstand zwischen den Sensoren der Präzision abträglich.

8.3. Softwareverbesserungen

Bei der eingesetzten Software existiert durchaus noch Potential für Verbesserungen. Diese Verbesserungen lassen sich mit einem abschätzbaren Aufwand durchführen und auch einzeln implementieren ohne andere Teile des System zu beeinflussen.

- Im Laufe der Arbeit hat sich herausgestellt, dass verschiedene Teile der Hardware (Platinen, Kabel, Akku, etc...) die Zuverlässigkeit der Funkverbindung beeinflussen. Derzeit wird nicht überprüft ob eine Funknachricht des ACTOs an das Framework auch wirklich ankommt. Da das relative Tracking-System aber darauf angewiesen ist, dass alle Messwerte berücksichtigt werden, kann dadurch ein erheblicher Fehler entstehen. Hardwareseitig wurden die Fehlerquellen für den Funk zwar weitgehend beseitigt (siehe Kapitel 4.6 und 4.7). Da aber nicht ankommende Nachrichten trotzdem auftreten können, sollten hier softwareseitig weitere Vorkehrungen getroffen werden.

- Kapitel 7.1 zeigt, dass der Algorithmus mit einem abschätzbaren Fehler behaftet ist. Eventuell lässt sich die Genauigkeit verbessern, wenn durch eine Erweiterung des Algorithmus nicht nur die aktuellen, sondern auch 2 oder 3 vorhergehende Messwerte, für die Positionsbestimmung berücksichtigt werden.

- Die vom Sensor aufgenommenen Bilder, die Speckle-Patterns, können zwar direkt ausgelesen werden, allerdings ist dies mit einem Neustart des Sensors und einem erneutem Upload der Firmware verbunden. Theoretisch wäre es damit möglich die absolute Position anhand eines bekannten Speckle-Patterns zu bestimmen. Zum Beispiel in Kombination mit der Verwendung von kariertem Papier (bei dem der Abstand der Linien bekannt ist) als Untergrund, wäre dieser Ansatz vielversprechend. Anhand dieser fixen, beobachtbaren Linien könnte der sich aufsummierende Fehler des relativen Speckle-Trackings regelmäßig, anhand einer bekannten absoluten Position, zurückgesetzt werden. Beispielsweise treten Linienkreuze nur exakt alle $5mm$ auf, wodurch auf die aktuelle Abweichung der Position in x- als auch y-Richtung geschlossen werden kann. Auch die Rotation würde sich so sehr einfach korrigieren lassen. Leider dauert der Neustart und das Laden der Firmware bei dem ADNS 9500 zu lange, um eine akzeptable Schleifendurchlaufzeit zu erhalten. Ebenso ist es fraglich, ob das Auslesen des Speckle-Patterns bei anderen Sensoren weniger Zeit in Anspruch nimmt. Ein Nachteil wäre auch die Notwendigkeit eines speziellen Untergrundes (z.B.: kariertes Papier).

- Die Berechnungen auf dem Arduino UNO weisen nur eine begrenzte Genauigkeit auf. Das liegt einerseits daran, dass das Arduino UNO nur mit Float- und nicht mit Double-Werten rechnet. Andererseits sind auch die trigonometrischen Berechnungen mit einer Abweichung in den Nachkommastellen behaftet. Für dieses Problem gibt es Lösungsansätze. Zum Beispiel kann versucht werden, Berechnungen mit

Double-Precision softwareseitig zu emulieren. Das würde allerdings den Rechenaufwand steigern und sich negativ auf die Schleifendurchlaufzeit auswirken. Eine Alternative wäre, die Rohdaten an das Framework zu übermitteln und die Berechnungen dort durchzuführen. Eine hardwareseitige Lösung stellt ein arithmetischer Co-Prozessor dar. Dieser würde allerdings noch mehr des ohnehin knappen Platzes beanspruchen und mehr Strom verbrauchen.

- Die erwähnte Ungenauigkeit durch die manuelle Konstruktion resultiert unter anderem in leicht gedrehten Sensoren. Theoretisch könnte der dadurch entstehende Fehler durch eine aufwändigere Kalibrierung weitgehend kompensiert werden. Allerdings werden dabei in der Praxis sehr viele trigonometrische Berechnungen mit sehr kleinen Zahlen durchgeführt. Da mit dem Arduino UNO wie erwähnt nur eine begrenzte Genauigkeit zu erwarten ist (Single- statt Double-Precision und Lookup-Tables für trigonometrische Funktionen), führt die Aufsummierung dieser entstehenden Fehler zu einer noch größeren Abweichung, anstatt sie zu reduzieren. Sollte also die Genauigkeit bei den Berechnungen gesteigert werden können, erscheint auch eine aufwändigere Kalibrierung sinnvoll.

9. Fazit

In dieser Arbeit wurde ein funktionstüchtiger Prototyp eines Tracking-Systems für ACTOs entwickelt, basierend auf handelsüblichen optischen Maussensoren. Es wird der gesamte Design- und Konstruktionsprozess beschrieben. Der Entwurf und die Konstruktion der Hardware werden ebenso, wie der Algorithmus und die Integration in das existierende Framework abgehandelt. Abschließend wird die Zuverlässigkeit und Genauigkeiten in verschiedenen Testverfahren überprüft und ausgewertet.

Diese Arbeit demonstriert das Potential für das Tracking von ACTOs, das von der Nutzung gewöhnlicher Maussensoren ausgeht. Es ist möglich mit frei im Handel verfügbaren Bauteilen, ein optisches Tracking-System zu konstruieren. Die Herstellung der Leiterplatte erfordert zwar spezialisierte Werkzeuge, die allerdings ebenfalls günstig erhältlich sind. Der Nachbau des in dieser Arbeit vorgestellten Systems ist also ohne Spezialwissen und mit einem rechtfertigbarem Aufwand zu bewerkstelligen. Die Materialkosten für einen Prototyp liegen bei ca. 35 Euro. Außerdem kann der Marker für das absolute Tracking auf der Oberseite des ACTOs befestigt werden, da eine kurzfristige Verdeckung des Markers mit der Hand nicht mehr automatisch zum temporären Verlust der Positionsbestimmung führt. Deswegen ist die aufwändige Glastischkonstruktion nicht mehr notwendig.

Abschnitt 7.1.7 zeigt, dass der verwendete Algorithmus bei Kurvenfahrten mit zunehmender Geschwindigkeit und Schrittlänge ungenauer wird. Die Berechnung des Winkels θ stellte sich als sehr exakt und unabhängig von der Schrittlänge heraus. Eine Schleifendurchlaufzeit von $25ms$ erschien im Laufe der Arbeit praxisnah und realistisch. Bei einer Schleifendurchlaufzeit von $25ms$ und einer tolerierbaren Abweichung von $1mm$ auf eine Bewegung von $61.22mm$ entlang eines Viertelkreises, ergibt sich so eine maximale Geschwindigkeit von ca. $100mm/s$. Damit bietet der Algorithmus zwar Verbesserungspotential, ist aber für die Anwendung hinreichend genau.

Zusätzlich musste geklärt werden, welche äußeren Einflüsse sich in welchem Ausmaß auf die Genauigkeit des Tracking-Systems auswirken. Daher wurde der Einfluss der Akkuladung und der Bewegungsgeschwindigkeit auf die Genauigkeit der Sensoren untersucht. Kapitel 7.2 zeigt, dass weder der Ladezustand des Akkus, noch die Bewegungsgeschwindigkeit einen Einfluss auf die Messgenauigkeit der Sensoren haben. Die Abweichungen erscheinen zufällig und folgen keinem Trend in eine bestimmte Richtung.

Von den Performance Tests (siehe 7.3) lässt sich ableiten, dass die Kalibrierung der Sensoren für eine bestimmte Oberfläche einen wichtigen Beitrag für die Genauigkeit des Systems leistet. Bei den Sensortests, bei denen die Genauigkeit der einzelnen Sensoren behandelt wird, konnte durch die Kalibrierung die Abweichung erheblich reduziert werden. Betrachtet man die Systemtests, bei denen die vom Algorithmus berechnete Position des Gesamtsystems überprüft wird, bietet sich ein differenzierteres Bild. Über kurze Strecken verringert sich die Abweichung durchwegs. Über längere Strecken ist eine erhebliche Verbesserung der Präzision bei einer Bewegung entlang der y-Achse des Systems zu beobachten. Da die Rotation aus den Δy-Werten der Sensoren berechnet wird und einen großen Einfluss auf die Genauigkeit über größere Distanzen hat, liegt der Schluss nahe, dass viele sehr kleine Δy-Werte (wie sie bei einer Bewegung entlang der

x-Achse auftreten) zu Abweichungen bei der Rotationsmessung führen. Dadurch lassen sich Genauigkeitsunterschiede, abhängig von der Bewegungsrichtung, erklären. Zusammenfassend lässt sich sagen, dass die Kalibrierung einen starken Einfluss auf die Präzision des Systems hat und damit unbedingt notwendig ist.

Abschließend muss beurteilt werden ob der Einsatz von Maussensoren zum Tracking der ACTOs sinnvoll erscheint. Da es sich um ein relatives Tracking-System handelt, gibt es einen mit der Zeit wachsenden Fehler. Daher scheint es nicht zielführend, ausschließlich Maussensoren für die Positionsbestimmung zu verwenden. Als Ergänzung eines absoluten optischen Trackings haben optische Maussensoren durchaus ihre Berechtigung. Das optische Tracking mittels Kamera und Markern weist im Moment eine große Schwäche auf: Bei schlechter Beleuchtung oder zu schnellen Bewegungen stößt es rasch an seine Grenzen. Hier kann das Tracking mittels Maussensoren eine wertvolle Überbrückung darstellen, da es von schnellen und ruckartigen Bewegungen bei weitem nicht so stark beeinflusst wird und damit immer noch valide Positionswerte liefern kann. Sobald die schnelle Bewegung stoppt, wird der sich aufsummierende Fehler des relativen Trackings durch die absolute Positionsbestimmung mittels Markern wieder kompensiert. Zusätzlich lässt sich mit optischen Maussensoren unkompliziert eine Lift-Detection-Funktion realisieren. Damit liegt die Stärke des Speckle-Trackings in der Ergänzung eines absoluten Trackings.

10. Ausblick

Die Evaluierung in dieser Arbeit hat gezeigt, dass der Einsatz von Maussensoren als Ergänzung zu einem absoluten Tracking verschiedene Vorteile bringt. Wenn der Marker für das absolute Tracking auf der Ober- statt Unterseite des ACTOs angebracht wird, kann die Position von oben, anstatt durch den Glastisch verfolgt werden. Durch das Tracking mit Maussensoren kommt es bei der Verdeckung des Markers (zum Beispiel durch die Hand) zu keinem automatischen Verlust der Positionsbestimmung mehr. Deswegen ist keine aufwändige Konstruktion mit einem Glastisch und einer Lichtquelle unter diesem notwendig. Dadurch kann beispielsweise das gesamte System (die ACTOs, der Arduino-Empfänger, das Android-System und dessen Stativ) selbstständig zu Hause für Rehabilitationsübungen genutzt werden. Dieses unkompliziertere und kostengünstigere Setup erlaubt im medizinischem Bereich zum Beispiel Hausbesuche, bei denen die ACTOs zu Rehab-Zwecken genutzt werden.

Auch die Spiele oder Übungen zur Rehabilitation selbst profitieren von dem zuverlässigeren Tracking. In Kombination mit der Lift-Detection die auch für jeden Sensor einzeln Informationen liefert, könnte zum Beispiel folgendes Rehab-Szenario zur Verbesserung der Motorik realisiert werden: Die Person mit VR-Brille hat den Auftrag mit dem ACTO eine Kugel aus einer Halterung zu holen, zu einem Loch zu transportieren und dort hineinfallen zu lassen. Auf dem Tisch ist nur das ACTO real. Kugel, Halterung und Loch werden über die VR-Brille an der richtigen Position dargestellt. Die Person bewegt das ACTO unter die Halterung mit der Kugel. Dort wird das ACTO als Ganzes angehoben, was die Lift-Detection erkennt. Dadurch wird virtuell die Kugel aus der Halterung gestoßen und auf dem ACTO fixiert. Jetzt kann das ACTO zum Ziel bewegt werden. Dort kann durch Kippen des ACTOs die Kugel in das Loch gerollt werden. Um das Kippen des ACTOs zuverlässig nur durch die Maussensoren zu messen, werden lediglich 3 statt 2 Sensoren auf der Unterseite benötigt. (Mehr als 2 Sensoren wäre aufgrund der höheren Genauigkeit in jedem Fall wünschenswert.) Dies ist nur ein Anwendungsbeispiel im umfangreichen Medizinbereich.

Des Weiteren wird das System auch für den reinen Entertainmentbereich attraktiver. Da es zu keinen abrupten Verlusten des Trackings kommt, können auch Spiele zur Anwendung kommen bei denen das ACTO schneller bewegt wird, oder der Marker kurze Zeit verdeckt ist. So können über dem ACTO mit einer VR-Brille verschiedene Formen dargestellt werden. Denkbare wäre damit ein animiertes VR-Schach oder jedes andere Brettspiel mit verhältnismäßig wenigen und hinreichend großen Spielfiguren. Besonders profitieren Anwendungen, bei denen das virtuelle Objekt veränderbar oder animiert sein muss. Die Fähigkeit des ACTOs sich selbstständig zu bewegen, beziehungsweise *greifbar* zu sein, ist ein weiterer Vorteil. Durch Speckle-Tracking lässt sich der Jitter in der Positionsbestimmung ebenfalls zuverlässiger reduzieren.

Eine Herausforderung für die Zukunft wird es sein, die Abhängigkeit von einem zusätzlichen, absoluten Tracking zu verringern. Eine Möglichkeit wäre es die Genauigkeit entsprechend zu steigern. Eine andere Variante wäre, eine speziell strukturierte Oberfläche (zum Beispiel ein Schachbrettmuster) zu verwenden. Da die Struktur der Oberfläche bekannt ist, kann so die Position, durch das Erkennen einer Markierung oder einer be-

stimmten Struktur, auf die dafür gespeicherten Koordinaten zurückgesetzt werden. Mit passenden Sensoren und Algorithmen wäre es damit möglich, komplett ohne ein absolutes Tracking auszukommen. Unter diesen Gesichtspunkten kommt eine weitere Stärke des Trackings mit Maussensoren zum Vorschein: Zur Positionsbestimmung wird theoretisch nur der Ausgangspunkt und die Oberfläche benötigt. Es sind keine weiteren Referenzpunkte, wie zum Beispiel Raumwände notwendig. Abseits der ACTOs wäre eine Anwendung in der Raumfahrt denkbar, etwa auf der Außenhülle einer Raumstation. Hier könnten Wartungsarbeiten vorgenommen werden, ohne auf andere Referenzpunkte als die Oberfläche angewiesen zu sein. Auch in Umgebungen wo andere Messarten mit Schwierigkeiten zu kämpfen haben (zum Beispiel unter Wasser), könnte Speckle-Tracking zum Einsatz kommen. Der Einsatz in einem Reinigungsroboter für weitläufige Räume (fehlende Referenzpunkte) wie in einem Turnsaal, oder in einem Pool unter Wasser (unzuverlässige Alternativen), scheint ebenfalls grundsätzlich möglich.

Des Weiteren könnte eine Maus die nicht nur ihre X- und Y-Bewegung, sondern auch ihre Rotation misst, als Eingabegerät für Design-Anwendungen aller Art Verwendung finden.

Wie breitgefächert das zukünftige Einsatzgebiet für Speckle-Tracking sein wird, hängt maßgeblich von der Möglichkeit ab die Genauigkeit zu steigern. Sollte dies gelingen, steht einer Verbreitung in den unterschiedlichsten Sparten nichts im Wege.

A. Benutzte Tools

Im folgenden Abschnitt werden die verschiedenen Werkzeuge vorgestellt, die verwendet wurden um die Prototypen zu designen und zu fertigen. Zu den benutzten Werkzeugen zählen sowohl die verwendeten Programme, wie auch das Verfahren zur Herstellung der Leiterplatten.

A.1. Fritzing

Fritzing [39] ist eine Open-Source-Prototyping-Software mit der elektronische Schaltungen designed werden können. Dabei orientiert sich das Programm grafisch an einem Breadboard mit den entsprechenden Bauteilen. Die Schaltung wird also grundsätzlich wie auf einem Breadboard geschaffen. In einem weiteren Schritt kann diese Schaltung virtuell auf eine Kupferplatte übertragen werden. Auf dieser werden dann die nötigen Leiterbahnen und Platzhalter, beziehungsweise Lötpads, für die Bauelemente eingezeichnet. Diese können frei verschoben und angepasst werden. Auf diese Art können Pläne sowohl für einfache, als auch sehr komplexe Platinen kreiert und im Maßstab 1:1 als PDF exportiert werden.

Fritzing wurde eingesetzt um sämtliche Prototypen und Tracking-Systeme, die für diese Arbeit Verwendung fanden, zu entwerfen.

A.2. Processing 2.0

Processing 2.0 [30] ist eine Programmiersprache, mit der visuelle Elemente kreiert und dargestellt werden können. Als eine der vielen Funktionen kann *Processing 2.0* Daten vom Serial-Bus auslesen. Im Rahmen dieser Arbeit wurde *Processing 2.0* benutzt, um die vom Tracking-System berechneten Daten darzustellen. So kann zum Beispiel, der zurückgelegte Weg am Bildschirm dargestellt werden. Dies stellt eine wichtige Hilfe sowohl bei der Entwicklung, als auch der Fehlersuche dar.

A.3. Autodesk 123D Design

Autodesk 123D [41] ist der Überbegriff für eine Reihe von Tools der Firma *Autodesk* mit unterschiedlichen Zielsetzungen, wie zum Beispiel der Modellierung von Schaltkreisen, oder der Generierung von 3D-Modellen auf Basis eines Fotos. *123D Design* ist ein kostenloses und einfaches 3D-Modellierungsprogramm. Der Fokus liegt auf der, durch die Community unterstützten, Modellierung von 3D-Modellen für die Produktion mit einem 3D-Drucker.

A.4. Ultimaker 2 3D-Drucker

Im Laufe der Arbeit wurden mehrere Bauteile mit *123D Design* modelliert und in einem *Ultimaker 2* 3D-Drucker ausgedruckt. [44] Dabei werden die von *123D Design* erstellten STL-Dateien, mit dem von *Ultimaker* entwickelten Programm *Cura* [45], in ein vom

3D-Drucker lesbares Format umgewandelt und auf eine SD-Card exportiert. [46] Anschließend wird das gewünschte Objekt anhand der Datei auf der SD-Karte, vom *Ultimaker 2* gedruckt. Alternativ kann der Drucker auch über den USB-Anschluss direkt am Computer angeschlossen werden. Das ist allerdings aufgrund der meist langen Druckzeiten wenig praktikabel. [46]

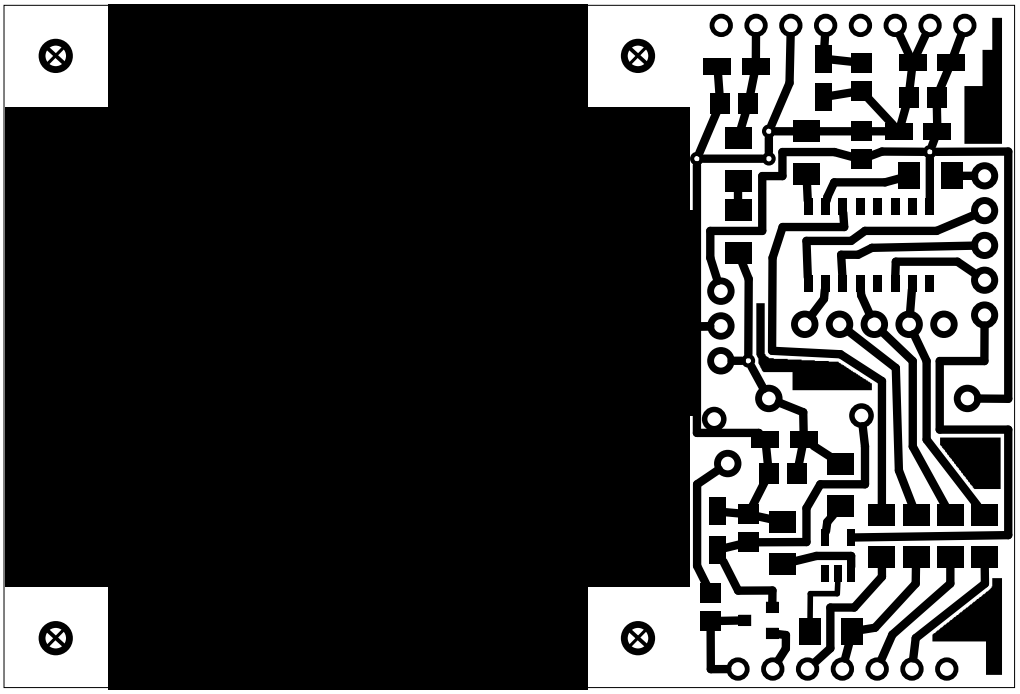
B. Finaler Prototyp Ätz-PDFs

Zum besserem Verständnis sind die Vorlagen vergrößert dargestellt.

B.1. ADNS 9500 ACTO finaler Prototyp oben

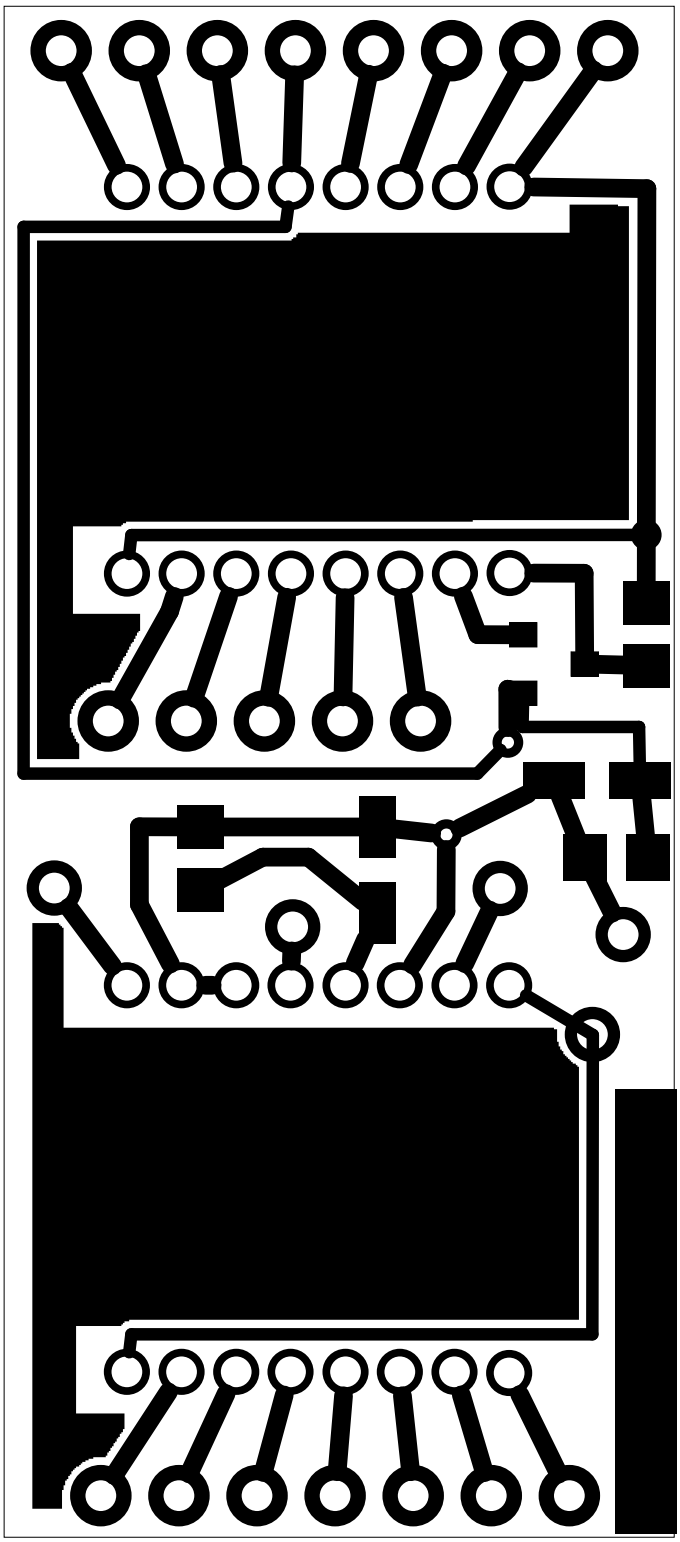
Da wie in Kapitel 4.7 beschrieben die $3.3V$ und $5V$ Leitungen vertauscht wurden, müssen diese Anschlüsse manuell mit kurzen Kabeln überbrückt werden.

Verhältnis ca. 2:1



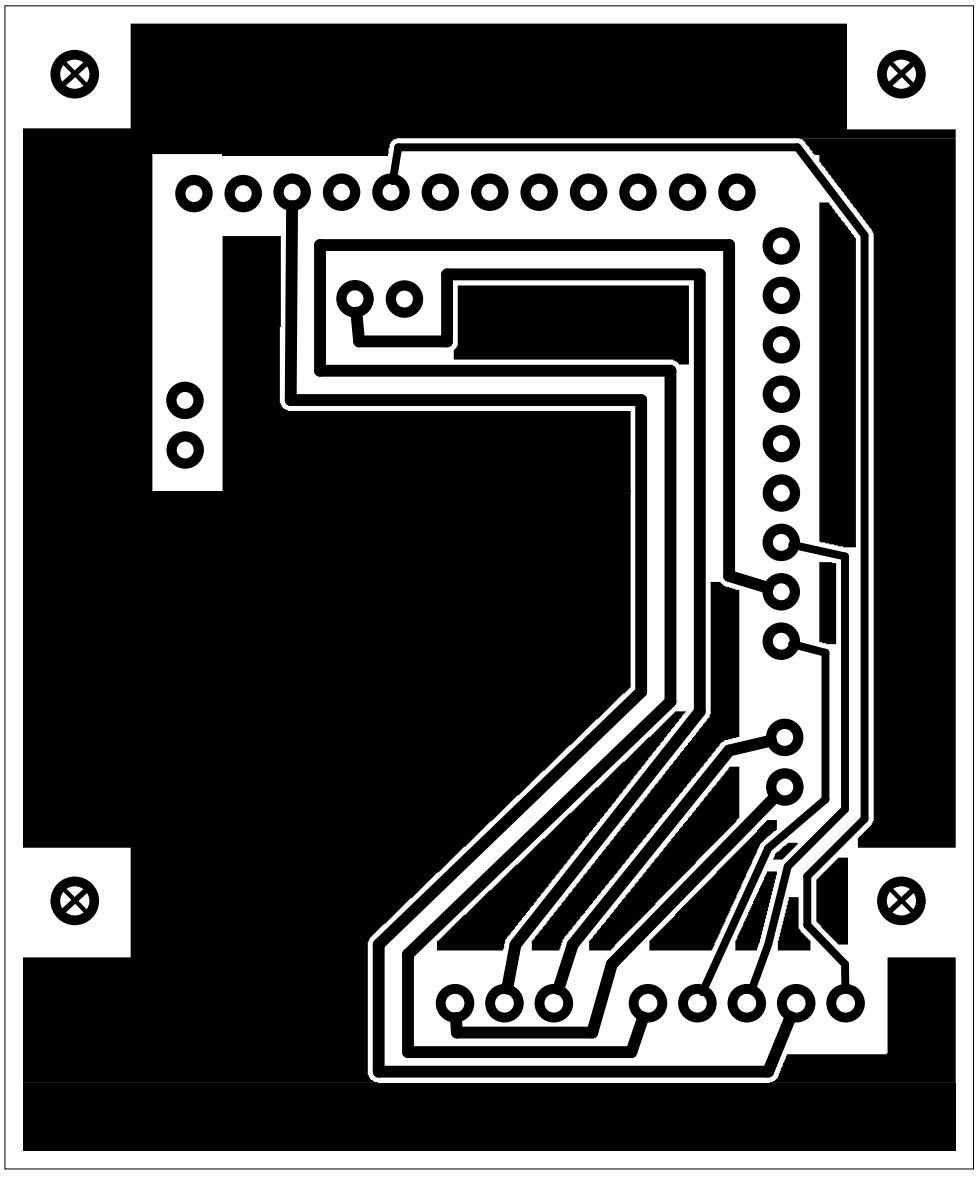
B.2. ADNS 9500 ACTO finaler Prototyp unten

Verhältnis ca. 4:1



B.3. Plug Extention

Verhältnis ca. 2.5:1



Abbildungsverzeichnis

Literatur

[1] J. Hertzberg, K. Lingemann, und A. Nüchter, *Mobile Roboter*. Springer Vieweg, 2012.

[2] O. Shaer und E. Hornecker, "Tangible User Interfaces: Past, Present, and Future Directions," *Foundations and Trends® in Human–Computer Interaction*, vol. 3, no. 1-2, pp. 1–137, 2010.

[3] E. Vonach, G. Gerstweiler, und H. Kaufmann, "ACTO: A Modular Actuated Tangible User Interface Object," *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, pp. 259–268, 2014.

[4] PixArt Imaging Inc., "ADNS-9500 Laser Gaming Sensor." [Online]. Verfügbar: http://www.pixart.com.tw/upload/ADNS-9500DS_S_V1.0_20130514144327.pdf [Zugriff am: 22.8.2016]

[5] D. Sekimori und F. Miyazaki, "Self-Localization for Indoor Mobile Robots Using Optical Mouse Sensor Values and Simple Global Camera Information," 2005.

[6] G. Fitzmaurice, H. Ishii, und W. Buxton, "Bricks: laying the foundations for graspable user interfaces," *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 442–449, 1995.

[7] H. Ishii und B. Ullmer, "Tangible bits," *Proceedings of the SIGCHI conference on Human factors in computing systems CHI 97*, vol. 39, pp. 234–241, 1997.

[8] G. Williams, "Pragmatic and Epistemic Action," 2008. [Online]. Verfügbar: https://philosophyandpsychology.wordpress.com/2008/02/20/pragmatic-and-epistemic-action/ [Zugriff am: 22.8.2016]

[9] L. Morgado, M. G. Cruz, und K. Kahn, "Radia Perlman – A pioneer of young children computer programming," *Current Developments in Technology-Assisted Education (2006)*, pp. 1903–1908, 2006.

[10] A. Blackwell, "Cognitive dimensions of tangible programming languages," *Software Engineering and Psychology of Programming*, pp. 391–405, 2003.

[11] J. Brewer, A. Williams, und P. Dourish, "A handle on what's going on: combining tangible interfaces and ambient displays for collaborative groups," *Computer*, pp. 15–17, 2007.

[12] H. Chung, C.-h. J. Lee, und T. Selker, "Lover ' s Cups : Drinking Interfaces as New Communication Channels," *CHI 06 extended abstracts on Human factors in computing systems*, pp. 375–380, 2006.

[13] L. E. Holmquist, J. Redström, und P. Ljungstrand, "Token-Based Access to Digital Information," *Huc 1999*, pp. 234–245, 1999.

[14] E. Mugellini, E. Rubegni, S. Gerardi, und O. A. Khaled, "Using personal objects as tangible interfaces for memory recollection and sharing," *Tei '07*, pp. 231–238, 2007.

[15] M. Glück, *MEMS in der Mikrosystemtechnik*. Springer Fachmedien Wiesbaden, 2005.

[16] D. Sekimori und F. Miyazaki, "Precise Dead-Reckoning for Mobile Robots Using Multiple Optical Mouse Sensors," pp. 145–151, 2007.

[17] M. Vincze, S. Olufs, P. Einramhof, und H. Wildenauer, "Roboternavigation in Büros und Wohnungen," *Elektrotechnik und Informationstechnik*, vol. 125, no. 1-2, pp. 25–32, 2008.

[18] H. Andreasson und T. Duckett, "Object Recognition by a Mobile Robot using Omnidirectional Vision," 2003.

[19] B. Jung und G. S. Sukhatme, "Real-time motion tracking from a mobile robot," *International Journal of Social Robotics*, vol. 2, no. 1, pp. 63–78, 2010.

[20] S. Singh und K. Waldron, "Design and evaluation of an integrated planar localization method for desktop robotics," *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 2, no. April, pp. 1109–1114, 2004.

[21] J. A. Cooney, W. L. Xu, und G. Bright, "Visual dead-reckoning for motion control of a Mecanum-wheeled mobile robot," *Mechatronics*, vol. 14, no. 6, pp. 623–637, jul 2004.

[22] Ilon Bengt Erland, "Wheels for a Course Stable Selfpropelling Vehicle Movable in any Desired Direction on the Ground or Some Other Base," 1975. [Online]. Verfügbar: http://worldwide.espacenet.com/publicationDetails/biblio;jsessionid=3D2EC6CCF390FC59D45C798C86C95A0F.espacenet_levelx_prod_0?FT=D&date=19750408&DB=&locale=de_EP&CC=US&NR=3876255A&KC=A&ND=1 [Zugriff am: 2016-08-22]

[23] A. Bonarini, M. Matteucci, und M. Restelli, "A kinematic-independent dead-reckoning sensor for indoor mobile robotics," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 4, pp. 3750–3755, 2004.

[24] A. Bonarini, M. Matteucci, und M. Restelli, "Automatic error detection and reduction for an odometric sensor based on two optical mice," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2005, no. September 2015, pp. 1675–1680, 2005.

[25] F. A. Kanburoglu, E. Kilic, M. Dölen, und B. Koku, "A Test Setup for Evaluating Long-Term Measurement Characteristics of Optical Mouse Sensors," p. 12, 2007.

[26] D.-H. Yi, T.-J. Lee, und D.-I. Cho, "Afocal Optical Flow Sensor for Reducing Vertical Height Sensitivity in Indoor Robot Localization and Navigation," *Sensors*, vol. 15, no. 5, pp. 11 208–11 221, 2015.

[27] PixArt Imaging Inc., "ADNS-6190-002 Laser Gaming Round Lens." [Online]. Verfügbar: http://www.pixart.com.tw/upload/ADNS-6190-002DS_S_V1.0_20130514144352.pdf [Zugriff am: 22.8.2016]

[28] Arduino, "Arduino." [Online]. Verfügbar: http://www.arduino.cc/ [Zugriff am: 2016-08-22]

[29] RFduino, "RFD22102 RFduino DIP." [Online]. Verfügbar: http://www.rfduino.com/product/rfd22102-rfduino-dip/index.html [Zugriff am: 2016-08-22]

[30] Processing Foundation, "Processing 2.0," 2015. [Online]. Verfügbar: https://processing.org/overview/ [Zugriff am: 2016-08-22]

[31] Nordic Semiconductor, "nRF24L01+ Datasheet," 2008. [Online]. Verfügbar: https://www.nordicsemi.com/kor/content/download/2726/34069/file/nRF24L01P_Product_Specification_1_0.pdf [Zugriff am: 22.8.2016]

[32] "SPI - Serial Peripheral Interface." [Online]. Verfügbar: http://www.mct.de/faq/spi.html [Zugriff am: 2016-08-22]

[33] A. Olwal, "SpeckleSense: Fast, Precis, Low-cost and Compact Motion Sensing using Laser Speckle," 2011.

[34] PixArt Imaging Inc., "ADNS-3530 Low Power LED Integrated Slim Mouse Sensor." [Online]. Verfügbar: http://www.pixart.com.tw/upload/ADNS-3530DS_S_V1.0_20130514110522.pdf [Zugriff am: 22.8.2016]

[35] PixArt Imaging Inc., "ADNS-8020 Track-on-Glass Laser Sensor." [Online]. Verfügbar: http://www.pixart.com.tw/upload/ADNS-8020DS_S_V1.0_20130514144946.pdf [Zugriff am: 22.8.2016]

[36] Wolfe, "SteelSeries Sensei RAW review," 2013. [Online]. Verfügbar: http://www.bestgamingmousecentral.com/steelseries-sensei-raw-review/ [Zugriff am: 2016-08-22]

[37] J. Bredendiek, "Platinenherstellung," 2012. [Online]. Verfügbar: http://sprut.de/electronic/platinen/index.htm [Zugriff am: 2016-08-22]

[38] CadSoft, "Was ist Eagle?" 2011. [Online]. Verfügbar: http://www.cadsoft.de/eagle-pcb-design-software/ueber-eagle/ [Zugriff am: 2016-08-22]

[39] Fritzing, "Fritzing," 2015. [Online]. Verfügbar: http://fritzing.org/home/ [Zugriff am: 2016-08-22]

[40] Corel, “CorelDRAW,” 2015. [Online]. Verfügbar: http://www.coreldraw.com/de/ [Zugriff am: 2016-08-22]

[41] Autodesk, “Autodesk 123D: 3D for Everyone,” 2015. [Online]. Verfügbar: http://www.123dapp.com/about [Zugriff am: 2016-08-22]

[42] G. Korak und G. Kucera, “Optical Tracking System,” *International Journal of Electronics and Telecommunications*, vol. 61, no. 2, pp. 165–170, 2015.

[43] Soundmaster, “Soundmaster PL-520USB.” [Online]. Verfügbar: https://www.libble.de/soundmaster-pl-520usb/p/631774/?page=0003 [Zugriff am: 2016-08-22]

[44] Ultimaker, “Ultimaker 2.” [Online]. Verfügbar: https://ultimaker.com/en/products/ultimaker-2-family/ultimaker-2 [Zugriff am: 2016-08-22]

[45] Ultimaker, “Cura Software.” [Online]. Verfügbar: https://ultimaker.com/en/products/cura-software [Zugriff am: 2016-08-22]

[46] Ashley Zelinskie, “Quick Start Guide: Printing with Ultimaker 2,” 2014. [Online]. Verfügbar: http://www.instructables.com/id/Quick-Start-Guide-Printing-with-Ultimaker-2/ [Zugriff am: 2016-08-22]