Programmierhandgerät für simulierte Industrieroboter

Teach Pendant for Simulated Industrial Robots

Semesterarbeit
an der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München.

Themenstellende/r	Prof. Dr.-Ing. Dr. h. c. Ralph Kennel
	Lehrstuhl für Elektrische Antriebssysteme und Leistungselektronik
Betreuer/Betreuerin	Dr. Bernhard Thiele
	DLR - Institut für Systemdynamik und Regelungstechnik
Eingereicht von	Thomas Bernhofer
	Enzianstr. 7
	85748 Garching b. München
	+49 1575 6996376
Eingereicht am	31.12.2020 in Garching bei München

Abstract

For the development of a universal robot controller, the prototype of a teach pendant was built in this thesis. For this purpose, it was integrated into the modeling and simulation environment Dymola. With the help of Qt Creator, a GUI was implemented, which serves as a user interface and as a communication interface between the teach pendant and the simulation environment. Several input devices were incorporated to control the robot with the prototype. These are a 3D mouse, a game controller and an IMU. A tablet computer serves as the control module. For this a housing was designed and built, in which the necessary hardware is installed to achieve the functionality of a teach pendant. The housing can be attached to the back of the tablet computer with rails and can thus be removed at any time. The implemented prototype enables the control of a simulated industrial robot and the execution of robot programs.

Zusammenfassung

Zur Entwicklung einer universellen Robotersteuerung wurde in dieser Arbeit der Prototyp eines Programmierhandgeräts aufgebaut. Dafür wurde dieser in die Modellierungs- und Simulationsumgebung Dymola integriert. Mithilfe von Qt Creator wurde eine GUI implementiert, welche als Bedienoberfläche und als Kommunikationsschnittstelle zwischen dem Teach-in-Modul und der Simulationsumgebung dient. Für die Steuerung des Roboters mit dem Prototypen wurden verschiedenen Eingabegeräte eingebunden. Diese sind eine 3D-Maus, ein Gamecontroller und eine IMU. Als Bedienmodul dient ein Tablet-Computer. Für diesen wurde ein Gehäuse konstruiert und aufgebaut, in dem die nötige Hardware verbaut ist, um die Funktionalität eines Programmierhandgeräts zu erreichen. Das Gehäuse kann über Schienen an der Rückseite des Tablet-Computers befestigt werden und ist somit jederzeit abnehmbar. Mit dem umgesetzten Prototyp wird die Steuerung eines simulierten Industrieroboters und die Ausführung von Roboterprogrammen ermöglicht.

Inhaltsverzeichnis

1. Einleitung

In vielen Produktionsprozessen sind Industrieroboter heutzutage nicht mehr wegzudenken. Deutschland ist in Europa laut dem „World Robotics Report 2020“ der IFR (engl. International Federation of Robotics) das Land mit den meisten Industrierobotern im Einsatz. Jährlich werden über 20 000 neue Roboter installiert. Im weltweiten Vergleich liegt Deutschland damit an fünfter Stelle. [25]
Für den Betrieb eines Industrieroboters wird immer eine Robotersteuerung benötigt. Darin befindet sich ein Computer für die Bahnplanung und die nötige Hardware, um die Motoren des Roboters anzutreiben. Es wird außerdem ein sogenanntes Programmierhandgerät benötigt. Damit ist es möglich, den Roboter zu konfigurieren, zu steuern und zu programmieren.
Am DLR-Institut für Systemdynamik und Regelungstechnik (DLR-SR) wird eine neue universelle Robotersteuerung entwickelt. Diese Arbeit beschäftigt sich mit dem Aufbau des Prototyps eines Programmierhandgeräts für diese Robotersteuerung.
Dafür soll zuerst der Stand der Technik für Teach-in-Verfahren und Programmierhandgeräte ermittelt werden, um mit dessen Hilfe ein Konzept für ein Programmierhandgerät zur intuitiven Roboterprogrammierung eines simulierten Roboters zu erstellen. Dabei soll auch die Übertragbarkeit der Roboterprogrammierung mit dem Programmierhandgerät von der Simulation auf den realen Roboter geprüft werden. Die nächste Aufgabe ist der Aufbau eines Prototyps, welcher zur Bedienung der Simulation genutzt werden kann. Dabei spielt die Kommunikationsschnittstelle zwischen Bedienmodul und Simulationsumgebung eine wichtige Rolle. Zuletzt soll eine Bewertung und Diskussion des Prototypen im Vergleich zum Stand der Technik erfolgen.
Der Prototyp des Programmierhandgeräts wird in eine modular erweiterbare Modellierungs- und Simulationsumgebung für physikalische Multidomänen-Systeme mit dem Namen Dymola integriert, welche auf der Programmiersprache Modelica [7] aufbaut. Modelica wiederum ist eine objektorientierte Programmiersprache zur Modellierung von physikalischen Systemen. Zur Simulation von seriellen Robotern wurde am DLR-SR eine Modelica-Bibliothek mit dem Namen „DLR Robots library“ [2] aufgebaut. Damit ist es unter anderem möglich, einen simulierten Roboter zu visualisieren und mithilfe verschiedener Bahnplanungsalgorithmen in eine gewünschte Position zu bewegen. Das Bahnplanungsmodul der Modelica-Bibliothek wird in dieser Arbeit so erweitert, dass die Simulation des Roboters über das Programmierhandgerät bedient werden kann.
Mit dem Programmierhandgerät soll eine intuitive Programmierung des Roboters durch verschiedene Eingabegeräte und einer GUI (engl. Graphical User Interface) möglich sein. In modernen Bedienmodulen gibt es meist zwei Eingabegeräte. Jedes Bedienmodul besitzt Eingabetasten, mit denen der Roboter einzeln in jeder Koordinatenachse bewegt werden kann. Zusätzlich gibt es zum Beispiel im Programmierhandgerät von Kuka eine 3D-Maus, oder bei der Firma ABB einen Drei-Achs-Joystick, um den Roboter zu bewegen. Weiter bietet Yaskawa beispielsweise eine IMU (engl. Inertial Measurement Unit), mit der die Orientierung des Roboters verändert werden kann. Außerdem werden im Bereich der Forschung verschiede-

ne Eingabegeräte getestet. Darauf wird in einer Literaturrecherche in Abschnitt 2.5 näher eingegangen. In dieser Arbeit werden zusätzlich zu den Eingabetasten eine 3D-Maus, ein Gamecontroller und eine IMU verwendet.
Die GUI für das Programmierhandgerät wird mithilfe des Qt-Frameworks aufgebaut. Dabei handelt es sich um eine C++-Erweiterung zur Entwicklung von grafischen Benutzeroberflächen. Zur Ausführung der GUI wird ein Tablet-Computer benutzt, welcher das zentrale Element des Programmierhandgeräts darstellt. Auf dem Tablet-Computer kann über Schienen ein Gehäuse aufgeschoben werden, in dem die nötige Hardware verbaut ist, um die Funktionalität eines Programmierhandgeräts zu erreichen. Mithilfe der GUI werden die Signale der Eingabegeräte und der Hardware im Gehäuse eingelesen und verarbeitet. Zusätzlich dient diese auch als Kommunikationsschnittstelle zwischen Bedienmodul und Simulationsumgebung.

Im nächsten Kapitel wird in die allgemeine Thematik eines Programmierhandgeräts eingeführt und in Form einer Literaturrecherche im Detail auf Steuerungskonzepte aus dem Bereich der Forschung eingegangen. Danach wird in Kapitel 3 der Aufbau des entwickelten Programmierhandgeräts vorgestellt, dabei wird das Konzept der Hardware und der GUI präsentiert. Weiter wird in Kapitel 4 die Implementierung dieser Konzepte erläutert. In Kapitel 5 wird die Bedienung des Roboters mit den Eingabegeräten erklärt und die Ergebnisse diskutiert. Kapitel 6 fasst schließlich die wichtigsten Aspekte der Arbeit zusammen.

2. Einführung in Programmierhandgeräte

Ein Programmierhandgerät ist eine Mensch-Maschine-Schnittstelle, welche benutzt wird um einen Industrieroboter zu steuern und zu programmieren. Fast jeder Hersteller von Industrierobotern hat ein eigenes Handbediengerät für seine Robotersteuerung, weshalb eine große Bandbreite auf dem Markt existiert. Beispiele für moderne Programmierhandgeräte sind das „smartPAD“ von Kuka (Abb. 1a) oder das „Smart Pendant“ von Yaskawa (Abb. 1b).

(a) smartPAD, aus [27].

(b) Smart Pendant, aus [41].

Abbildung 1 Beispiele für Programmierhandgeräte.

Grundlegend hat jedes Programmierhandgerät die gleichen Funktionen. Sie unterscheiden sich jedoch in der Art und Weise wie die Benutzeroberfläche und das Gerät selbst gestaltet ist. Des Weiteren können sie über Zusatzfunktionen verfügen, wie beispielsweise eine Simulation des Roboters direkt am Handbedienterminal oder verschiedene Eingabegeräte mit denen der Roboter bewegt werden kann.
In den folgenden Unterkapiteln werden die grundlegenden Bestandteile von Programmierhandgeräten für Industrieroboter beschrieben, welche hier in Steuerung, Programmierung, Konfiguration und Sicherheit gegliedert werden. Im Anschluss wird in Form einer Literaturrecherche genauer auf Steuerungskonzepte von Robotern eingegangen.

2.1. Steuerung

Die Steuerung dient der Bewegung des Roboters. Dabei wird zwischen Gelenkkoordinaten, bzw. Achskoordinaten, und kartesischen Koordinaten unterschieden.
Bei einer Steuerung mittels Gelenkkoordinaten werden die Gelenkwinkel der einzelnen Roboterachsen vorgegeben. Diese werden direkt an die Robotersteuerung übergeben und er-

fordern daher keinen großen Rechenaufwand. Der Nachteil daran ist, dass es schwierig sein kann den TCP (engl. Tool Center Point) durch Achsbewegungen in eine gewünschte Position zu bringen.
Im Falle einer Steuerung durch kartesische Koordinaten wird eine Pose im Raum vorgegeben, welche aus einer Position in X-, Y- und Z-Richtung und einer Orientierung um die drei Raumachsen besteht. Diese wird über die inverse Kinematik in Gelenkkoordinaten umgerechnet und der Robotersteuerung übergeben, welche den Roboter an die gewünschte Position fährt. So kann die Position intuitiv vorgegeben werden, es wird jedoch ein zusätzlicher Rechenschritt zum Lösen der indirekten Kinematik benötigt.
Um die Position des TCP, bzw. des Werkzeugs, zu bestimmen, muss die direkte Kinematik des Roboters gelöst werden. Dies wird zum Beispiel benötigt, um die aktuelle Position des Werkzeugs am Roboter auf dem Programmierhandgerät anzeigen zu können.
Der Roboter kann über verschiedene Eingabegeräte gesteuert werden. Diese variieren je nach Handbediengerät. Mit dem „smartPAD“ von Kuka (Abb. 1a) kann beispielsweise die Position des Roboters über Verfahrenstasten oder über eine 3D-Maus verändert werden. Das „Smart Pendant“ von Yaskawa (Abb. 1b) verfügt über Verfahrenstasten und eine inertiale Messeinheit oder IMU mit der die Orientierung des TCP eingestellt werden kann.
Außerdem kann bei der Steuerung von Industrierobotern noch bestimmt werden bezüglich welchem Referenzkoordinatensystem sich der TCP bewegen soll. Jeder Hersteller benennt die Koordinatensysteme etwas anders, aber im Grunde handelt es sich immer um dieselben. Am Beispiel von Kuka werden im Folgenden die vier wichtigsten Koordinatensysteme beschrieben, welche in Abbildung 2 dargestellt sind.

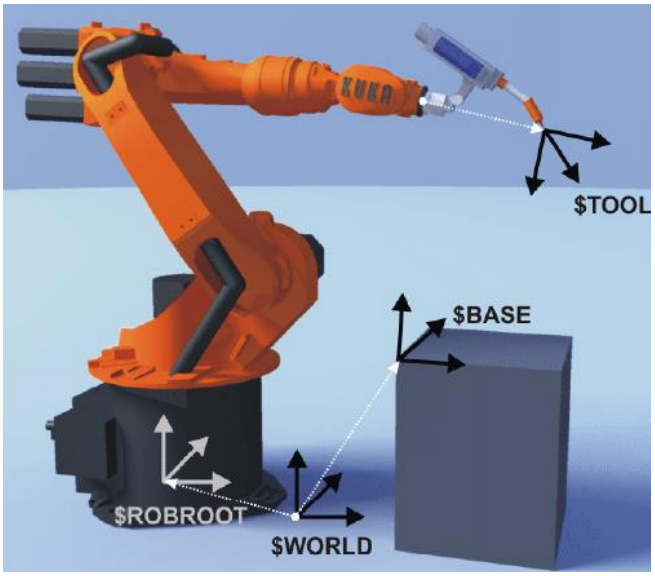


Abbildung 2 Koordinatensysteme eines Kuka 6-Achs Roboters, aus [35, S. 48].

Das WORLD Koordinatensystem bezeichnet ein Inertialsystem welches als Referenz für ROBROOT und BASE dient. ROBROOT liegt im Ursprung des Roboters und beschreibt

die Position des Roboterfundaments in Bezug zum WORLD Koordinatensystem. Oft werden ROBROOT und WORLD gleichgesetzt. Das BASE Koordinatensystem beschreibt die Positon eines Tisches oder eines Werkstücks bezüglich WORLD. Um die Position des TCP zu beschreiben wird das TOOL Koordinatensystem festgelegt, welches an der Spitze des Werkzeugs auf dem Roboter liegt. Die Z-Achse des TOOL Koordinatensystems wird meist so gewählt, dass diese in die Stoßrichtung des Werkzeugs zeigt. Das bedeutet, wenn beispielsweise ein Fräskopf auf dem Roboter montiert ist, zeigt die Z-Achse in die Richtung des Spannfutters.
Es ist möglich den Roboter in einem gewünschten Koordinatensystem translatorisch entlang der Achsen und rotatorisch um die Koordinatenachsen zu bewegen.
Koordinatensysteme wie BASE und TOOL können vom Benutzer festgelegt werden. Dafür gibt es mehrere Verfahren bei denen durch positionieren des Roboters in mehreren Stellungen ein Koordinatensystem eingemessen wird. Dadurch wird es viel einfacher zum Beispiel eine Position auf einem Tisch anzufahren oder ein Werkstück zu bearbeiten.
Das „Smart Pendant“ (Abb. 1b) von Yaskawa hat zum Beispiel ein weiteres Koordinatensystem, welches sich „Smart Frame“ nennt. Dieses befindet sich im Programmierhandgerät. Durch die IMU hat die Robotersteuerung immer Kenntnis wie das Handbediengerät zum Roboter ausgerichtet ist und kann so den Roboter bezüglich des „Smart Frame“ bewegen. Das erleichtert die Bedienung des Roboters, besonders für Anfänger, enorm.
Weitere Funktionen sind die Einstellung der Bewegungsgeschwindigkeit und das Ansteuern von digitalen oder analogen Ausgänge, beziehungsweise das Auslesen von Eingänge. Damit können zum Beispiel Greifer bewegt oder Sensoren ausgelesen werden.

2.2. Programmierung

Es gibt eine Vielzahl von Programmiersprachen für Industrieroboter , welche sich in der Auswahl der Befehle stark ähneln. Einige bekannte Sprachen sind RAPID von ABB, INFORM von Yaskawa und KRL (engl. Kuka Robot Language) von Kuka.
Ein Roboterprogramm besteht immer aus hintereinander geschalteten Befehlen zum Abfahren von Punkten. Diese Punkte im Raum können auf verschiedene Arten angefahren werden. Dabei wird mindestens zwischen drei verschiedenen Befehlen unterschieden. Bei Kuka werden diese Befehle PTP (engl. Point to Point), LIN (Linearinterpolation) und CIRC (Zirkularinterpolation) genannt. LIN und CIRC sind sogenannte CP-Steuerungen (engl. continuous path).
Die folgenden Erläuterungen zu Bewegungsbefehlen von Industrierobotern beruhen auf Weber (2019, S. 75-97). PTP ist der schnellste Bewegungsbefehl um von einer Position in eine andere zu fahren. Der Pfad von einer Pose in die nächste ist aber nicht vorhersehbar, da sich alle Achsen bewegen können und somit keine gerade Bahn des TCP entsteht. Bei der PTP Bewegung werden die Achsen meist synchronisiert, so dass alle Achsen gleichzeitig starten und stoppen. Das bedeutet alle Achsen werden an die langsamste Achse, welche auch Leitachse genannt wird, angepasst. Diese Methode wird auch synchrone PTP genannt. Point to Point wird benutzt wenn die Bewegung so schnell wie möglich ausgeführt werden soll, im Ar-

beitsraum genug Platz vorhanden ist und die Richtung aus welcher der Zielpunkt angefahren wird keine Rolle spielt.
Bei einer Bewegung mit Linearinterpolation ergibt sich eine Bahn entlang einer Geraden zwischen einem Anfangs- und Endpunkt. Da sich der TCP geradlinig bewegen soll, können nicht alle Achsen gleichzeitig losfahren, sondern müssen aufeinander abgestimmt werden. Dadurch wird die Bewegung langsamer als beim PTP. Um eine geradlinige Bewegung des TCP zu erreichen, wird zwischen zwei Punkten mit einer bestimmten Schrittweite linear Interpoliert. Jeder Interpolationsschritt muss über die inverse Kinematik in Gelenkwinkel umgerechnet und dann der Robotersteuerung übergeben werden. Linearinterpolation wird benutzt wenn die Richtung, von welcher der Zielpunkt angefahren wird relevant ist und wenn die Bahn, auf der sich der TCP bewegt eine Gerade sein muss, wie beispielsweise beim Schweißen.
Bei der Zirkularinterpolation wird eine Kreisbahn abgefahren. Um eine Kreisbahn zu generieren müssen drei Punkte im Raum festgelegt werden. Ein Startpunkt, ein Zielpunkt und ein Hilfspunkt. Dieser Befehl ermöglicht das Abfahren eines Kreises, was wiederum beim Schweißen oder beim Kleben von Vorteil ist.
Zusätzlich gibt es meist die Möglichkeit Zwischenpunkte zu durchfahren. Normalerweise würde der Roboter an jedem Zielpunkt komplett abbremsen. Mithilfe von Überschleifen oder mithilfe von Splines kann der Stillstand am Zielpunkt verhindert werden und dadurch Zeit und Energie gespart werden. Weiter sinkt auch die Belastung auf den Roboter und ruckartige Bewegungen werden vermieden. Wenn Überschleifen aktiviert ist, bekommt der Roboter die Möglichkeit den Zielpunkt in einem Befehl nicht direkt anzufahren. Beim Unterschreiten einer gewissen Geschwindigkeit (Geschwindigkeitsüberschleifen) oder eines gewissen Abstands (Positionsüberschleifen) wird das nächste Bahnsegment begonnen. So kann der Zielpunkt mit einer konstanten Geschwindigkeit umfahren werden. In Abbildung 3) ist Überschleifen am Beispiel einer Linearinterpolation dargestellt.

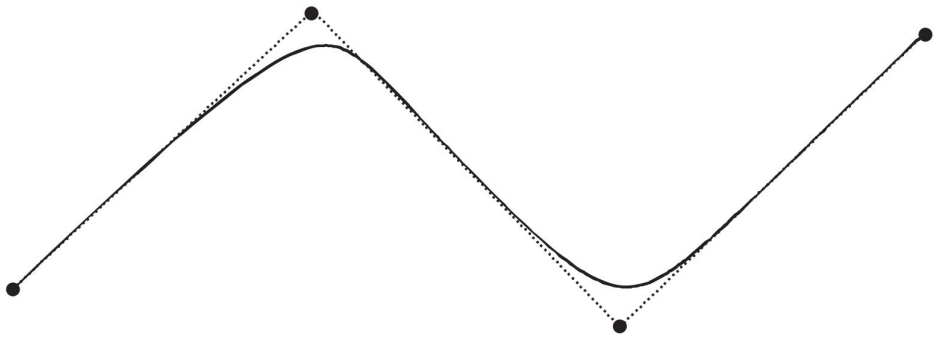


Abbildung 3 Darstellung von Überschleifen, aus [17, S. 93].

Beim Durchfahren eines Punktes mit Splines wird der Zielpunkt direkt durchfahren, wobei die Geschwindigkeit aber nicht auf null sinkt. Diese kann hier vorgegeben werden. Ein Beispiel einer Spline Interpolation ist in Abbildung 4) dargestellt.
Für einen Bewegungsbefehl werden immer mehrere Parameter benötigt. Diese sind meist

die Position zu der der Roboter bewegt werden soll, die maximale Geschwindigkeit und ob Überschleifen aktiviert ist oder nicht.

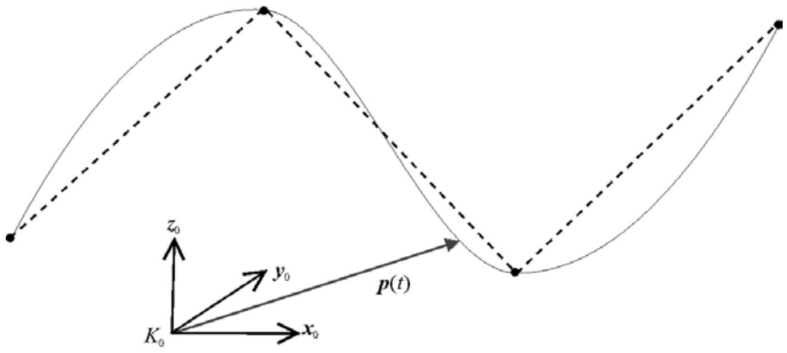


Abbildung 4 Darstellung einer Spline Interpolation, aus [17, S. 95].

Bei der Programmierung gibt es neben den Bewegungsbefehlen auch logische Befehle, um zum Beispiel auf Sensordaten zu reagieren. Diese bestehen normalerweise aus den Anweisungen, wie sie in den meisten Programmiersprache zu finden sind. Beispiele sind das Anweisungen wie if/else, switch/case, for-Schleifen oder while-Schleifen. Zusätzlich sind noch „wait“ Befehle vorhanden welche das Programm anhalten bis ein gewisses Ereignis eintritt, oder ein „halt“ Befehl, welcher das Programm anhält bis der Startknopf gedrückt wird.
Um diese logischen Befehle sinnvoll nutzen zu können, gibt es noch die Möglichkeit Eingänge auszulesen und somit ein Ereignis zu erzeugen auf das reagiert werden kann. Damit beispielsweise Greifer geöffnet und geschlossen werden können gibt es weitere Befehle um Ausgänge zu schalten. Diese Eingänge und Ausgänge gibt es immer in digitaler und analoger Form.
Um das Programm abzufahren, gibt es zwei Modi in die das Programmierhandgerät geschaltet werden kann. Den Manuellen-Modus und den Automatik-Modus. Dazu mehr in Abschnitt 2.4. Im Manuellen-Modus wird oft noch mal unterschieden, wie das Programm beim Drücken des Start Knopfes abgefahren wird. Es gibt meist einen Kontinuierlichen-Modus, bei dem das Programm durchgefahren wird und einen Einzelschritt-Modus, der entweder nach jeder Bewegung oder nach jeder Programmzeile anhält und dann wieder auf das Startkommando wartet.
Komplizierte Roboterprogramme werden üblicherweise offline in einer Simulationssoftware programmiert. Anschließend werden diese über das Programmierhandgerät im Manuellen-Modus am realen Roboter getestet. Dabei werden gegebenenfalls Zielpunkte und Geschwindigkeiten angepasst.

2.3. Konfiguration

Um einen Roboter in Betrieb zu nehmen, muss dieser konfiguriert werden. Dafür müssen verschiedene Einstellungsmöglichkeiten vorhanden sein. Eine sehr wichtige Konfigurationsmöglichkeit ist die Vermessung des TCP. Durch die direkte Kinematik ist ein Koordinatensystem bekannt, welches auf dem Flansch der sechsten Achse liegt. Wenn ein Werkzeug darauf montiert wird gibt es zwei Möglichkeiten wie das TCP Koordinatensystem festgelegt werden kann. Zum einen können die Abmessungen und der Verdrehwinkel manuell eingegeben werden. Diese Daten lassen sich aus einem Datenblatt oder einem CAD Programm ermitteln. Eine weitere Möglichkeit ist das Vermessen eines Koordinatensystems. Dabei wird ein feststehender Punkt in mehreren Posen angefahren. Daraus kann der translatorische Abstand und auch die Rotation gegenüber des Flansches ermittelt werden. Die Vermessung wird benötigt, wenn zum Beispiel das Werkzeug am Roboter ausgetauscht wird, oder nach einer Kollision.
Es muss weiter die Möglichkeit geben ein Koordinatensystem festzulegen, in dem sich der Roboter bewegen soll. Hierfür sind drei Punkte auf einer Ebene nötig. So kann beispielsweise ein Koordinatensystem über drei fixe Punkte eines Tisches definiert werden. Wenn der Tisch bewegt wird muss nicht ein ganzes Roboterprogramm neu geschrieben werden, sondern es muss lediglich das Koordinatensystem neu vermessen werden.
Weitere Einstellmöglichkeiten gibt es für digitale sowie analoge Inputs und Outputs, für Softwareendschalter, für Geschwindigkeits- bzw. Beschleunigungsbeschränkungen und für die Inbetriebnahme des Roboters.

2.4. Sicherheit

Bei Industrierobotern gibt es verschiedene Modi in denen der Roboter ein Roboterprogramm abfährt. Dabei wird immer mindestens zwischen einem Manuellen-Modus und einem Automatischen-Modus unterschieden. Meist gibt es dafür einen mechanischen Schalter der umgelegt wird, um zwischen den Modi umzuschalten.
Der Manuelle-Modus wird benötigt um ein Roboterprogramm einzurichten und zu testen. Um zu überwachen, ob die Posen des Roboters auch richtig angefahren werden, ist es oft nötig sich im Bewegungsbereich des Roboters aufzuhalten. Dafür müssen die Sicherheitseinrichtungen einer Roboterzelle oft umgangen werden. Wenn sich beispielsweise der Bediener in der Roboterzelle aufhält, kann die Sicherheitstür nicht geschlossen werden. Im Manuellen-Modus kann der Roboter trotzdem mit begrenzter Geschwindigkeit bewegt werden, wenn der Zustimmtaster gedrückt wird.
Im Automatik-Modus ist es nicht möglich den Roboter zu bewegen, wenn die Sicherheitseinrichtungen der Roboterzelle auslösen. In diesem Modus wird ein Roboterprogramm in voller programmierter Geschwindigkeit abgefahren.
Bei einem Programmierhandgerät für Industrieroboter gibt es üblicherweise die folgenden zwei Sicherheitseinrichtungen. Zum einen ist das ein Not-Halt-Schalter, welcher die Anlage

beim Drücken sofort in den Stillstand versetzt. Zum anderen gibt es einen Zustimmtaster , welcher im Manuellen-Modus verwendet wird. Das ist ein mechanischer Knopf, der drei Positionen einnehmen kann. Wenn der Schalter nicht gedrückt wird, kann der Roboter im Manuellen-Modus nicht bewegt werden. Wird er halb durchgedrückt lässt sich der Roboter mithilfe der Eingabegeräte oder des Roboterprogramms bewegen. Wenn der Schalter ganz durchgedrückt wird, ist der Roboter im gleichen Zustand, wie wenn der Schalter nicht gedrückt ist.
Es ist außerdem festzuhalten, dass moderne Programmierhandgeräte wie das „smartPAD“ von Kuka (Abb. 4) oft zusätzliche Funktionen beinhaltet, wie beispielsweise die Simulation eines Programms am Bedienmodul.

2.5. Literaturrecherche

Im Bereich der Forschung wird versucht Methoden zu finden mit denen Roboter intuitiver bedient werden können. Dabei werden verschiedene Ansätze benutzt, welche sich zum einen im Eingabegerät unterscheiden und zum anderen in der Programmierweise.

In der Arbeit von Colombo, Dellafrate und Molinari Tosatti [4] wird ein Gamecontroller verwendet, wie er für Computer- und Konsolenspiele eingesetzt wird, um einen Industrieroboter mit sieben Achsen zu steuern. In der mobilen Robotik finden diese Geräte schon verbreitet Einsatz, da diese günstig und zuverlässig sind.

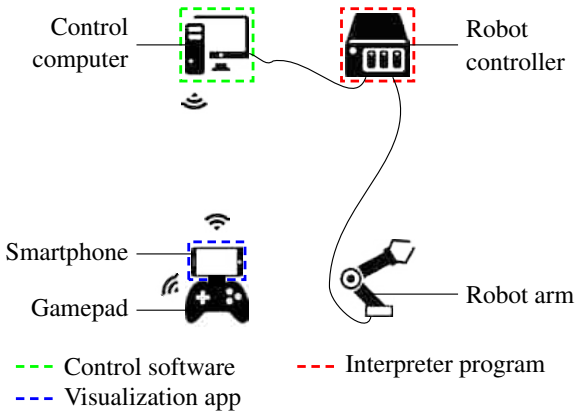


Abbildung 5 Schema einer Steuerung von einem Industrieroboter mithilfe eines Gamepads, aus [16, S. 2].

In der Veröffentlichung von Wagner, Avdic und Heß [16] wird untersucht, wie gut sich ein Industrieroboter mithilfe eines Gamecontrollers programmieren lässt. In Abbildung 5 ist ein Schema des Testaufbaus dargestellt. Der Controller ist kabellos mit dem „Control computer“ verbunden, welcher über den „Robot controller“ den Roboter steuert. Zusätzlich werden

auf einem Smartphone, welches auf dem Controller befestigt ist, Informationen des Roboters angezeigt. Bei einem Usability-Test mussten Probanden zwei einfache Aufgaben mithilfe eines Programmierhandgeräts und mithilfe des Gamecontrollers durchführen. Die erste Aufgabe war eine einfache Pick and Place Aufgabe und die zweite, den Roboter entlang eines aufgezeichneten Pfades fahren zu lassen. Dabei wurden die Aufgaben von jedem der zehn Probanden zweimal ausgeführt und die Zeiten gestoppt. Das Resultat des Usability-Tests zeigt mit großer Wahrscheinlichkeit, dass das Programmieren mithilfe des Gamecontrollers schneller ist als mit einem Programmierhandgerät.

Weiter wird in der Arbeit von Hein, Hensel und Worn [8] und der Arbeit von Hsien-I und Yu-Hsiang [9] wird ein Stift, welcher mit einem optischen Tracker ausgestattet ist, benutzt, um die Bahn des TCP vorzugeben. Damit die Position des Stifts bestimmt werden kann, muss eine Kamera aufgestellt werden, welche den Tracker erfasst. Diese muss den Stift immer im Sichtfeld haben. Wird anschließend eine Bahn mit dem Stift vorgegeben, nimmt die Kamera die Bewegung auf und überträgt diese auf den Roboter. So müssen keine Bewegungsbefehle in den Code geschrieben werden, da diese automatisiert durch die Vorgabe der Bewegung generiert werden. In Abbildung 6 ist der Stift abgebildet. Zur Übertragung der Bewegung muss ein Koordinatensystem auf der „Teach pen platform“ festgelegt werden, welches über die in der Abbildung gezeigten Koordinatensysteme auf die „Roboter platform“ übertragen wird. In beiden Arbeiten kommen die Autoren zu dem Schluss, dass es mit dem getrackten Stift für Beginner einfacher und intuitiver ist einen Roboter zu programmieren als mit einem Programmierhandgerät.

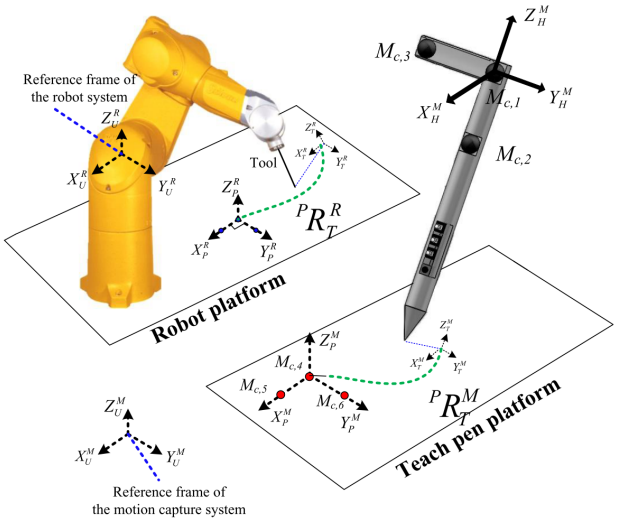


Abbildung 6 Schema einer Steuerung von einem Industrieroboter mithilfe eines visuellen Trackers, aus [9, S. 5].

Das Start-up Wandelbots hat ein kommerzielles Produkt mit dem Namen „TracePen“ [40] entwickelt, welches auf die gleiche Weise funktioniert. Dieser ist in Abbildung 7 dargestellt.

Die Pose des Stifts wird durch eine IMU und ein optisches Infrarot-Trackingsystem bestimmt. Der optische Tracker des „TracePen“ befindet sich an der flachen Hinterseite, die sich in Abbildung 7 auf der rechten Seite befindet. An der Spitze des Stifts können verschiedene Aufsätze angebracht werden, welche das Werkzeug am Roboter darstellen. Nach einem Kalibriervorgang kann der Stift benutzt werden um die Bewegungen, die ein Roboter ausführen soll, zu demonstrieren. So wird dem Roboter gezeigt, wie er sich bewegen soll. Auf diese Art und Weise kann der Roboter auch von Personen programmiert werden, welche keine Programmierkenntnisse besitzen. Zur Programmierung wird nur Prozessverständnis benötigt. Am HMI (engl. Human Machine Interface) gibt es verschiedene Bausteine, die an bestimmten Punkten eingefügt werden können, wie zum Beispiel einen Greifer zu schließen oder Schleifen einzubauen.

Abbildung 7 Tracepen von Wandelbots, aus [40].

Eine weitere Methode einen Roboter zu bewegen ist die Handführung. Diese Methode wurde bereits in einigen Produkten umgesetzt und ist vor allem im Bereich der kollaborativen Roboter verbreitet. Es gibt außerdem die Möglichkeit schwere Industrieroboter ohne Drehmomentsensoren handgeführt zu betreiben, indem ein Kraft-Momenten-Sensor mit sechs Freiheitsgraden am Roboter angebracht wird. Dieses System bietet zum Beispiel Kuka mit dem „ready2_pilot“ [26] an. Dabei handelt es sich um eine 6D-Maus, welche auf dem Roboter in der Nähe des TCP montiert wird, um diesen zu bewegen. Es gibt auch Ansätze, bei denen der Motorstrom in Verbindung mit einem modellbasierten Regler verwendet wird, um Industrieroboter ohne Drehmomentsensoren handgeführt zu betreiben [18].

Eine andere Möglichkeit Roboter zu steuern wird in der Arbeit von Neto, Pires und Moreira [10] dargestellt. Hier wird ein Roboter über Gesten und Sprache gesteuert. Ein Schema der Funktionsweise wird in Abbildung 8 gezeigt. Dabei wird als Eingabegerät, um den Roboter zu steuern, ein Nintendo Wii-Controller und ein Headset benutzt, welche über Bluetooth mit einem Computer verbunden sind. Mit dem Controller wird der Roboter bewegt und mit dem Mikrofon des Headsets können Sprachbefehle gegeben werden, um zum Beispiel einen Punkt zu speichern oder die Motoren auszuschalten. Um die Bewegung des Bedieners über den Wii-Controller zu identifizieren, wurde ein neuronales Netz angelernt. Die Steuerung mit Gesten und Sprache wurde in der Arbeit schließlich mit einem handgeführten Roboter verglichen. Dabei kamen die Autoren zu dem Ergebnis, dass die Handführung des Roboters

intuitiver und schneller ist als die Gestensteuerung.
In einer anderen Veröffentlichung von Neto, Pires und Moreira [11] wird ein weiteres neuronales Netzwerk vorgestellt, welches Handbewegungen, bzw. Gesten, über die Sensoren in einem Handschuh auswertet, um auf diese Weise den Roboter zu bewegen.

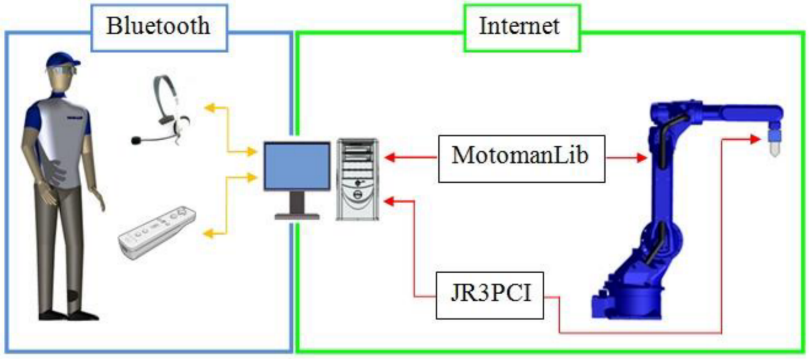


Abbildung 8 Schema einer Steuerung von einem Industrieroboter mithilfe eines Wii-Controlers und einem Headset, aus [10, S. 5].

In der Arbeit von Tang und Webb [15] wird eine weitere Gestensteuerung mit einem optischen Handverfolgungsmodul umgesetzt. Dadurch lässt sich ein Roboter mithilfe von verschiedenen Handbewegungen steuern. Die Gestensteuerung soll das Programmierhandgerät ergänzen um die Programmierung zu erleichtern.
In der Veröffentlichung von Cueva C., Torres und Kern [5] wird eine Gestensteuerung mithilfe einer Microsoft Kinect v2 umgesetzt. Mithilfe der Kamera wird ein Interface geschaffen, mit dem ein Roboter mit sieben Freiheitsgraden gesteuert werden kann.

Um die Programmierung von Robotern zu vereinfachen werden hauptsächlich zwei Ansätze verfolgt, welche im Folgenden erläutert werden sollen.
Beim ersten Ansatz wird versucht die Programmierung in einer aufgabenorientierten Ebene durchzuführen. Diese Art der Programmierung wird auch als TLP (engl. Task-Level Programming) bezeichnet. Dabei gibt es mehrere vorprogrammierte Fertigkeiten (engl. Skills), welche parametriert werden können und jeweils eine spezielle Folge von Befehlen ausführen. Durch Aneinanderreihung der Skills kann ein Roboter programmiert werden. Diese Aneinanderreihung wird meist in einer GUI ausgeführt. Beispiele dafür sind Franka Desk [24] oder RAZER [14]. Der Vorteil der aufgabenorientierten Programmierung ist, dass die erstellten Programme leicht nachvollziehbar und flexibel sind.
Ein weiterer Ansatz ist die Programmierung durch Demonstration. Diese wird auch als PbD (engl. Programming by Demonstration) oder als LfD (engl. Learning from Demonstration) bezeichnet. Das Ziel dabei ist, einem Roboter eine Aufgabe beizubringen, indem diese durch einen Menschen vorgeführt wird [1]. Dies kann auf mehrere Arten passieren. Zum Beispiel, indem ein Roboter handgeführt durch eine Aufgabe geleitet wird [3] oder indem die Demonstration eines Menschen über eine Kamera beobachtet wird [6]. Der Vorteil der Programmie-

rung durch Demonstration ist, dass es schnell und intuitiv ist.
Das Ziel in vielen Veröffentlichungen ist es die beiden Programmieransätze zu verbinden, um die Vorteile von TLP und PbD zu vereinen. Ein Beispiel ist die Veröffentlichung von Steinmetz, Nitsch und Stulp [13]. Um einen Roboter zu programmieren, muss hier nur die zu bewältigende Aufgabe handgeführt vorgemacht werden. Daraus werden automatisch Skills abgeleitet und ein aufgabenorientiertes Programm erstellt, welches nachträglich bearbeitet werden kann.

3. Aufbau

In diesem Kapitel wird der Aufbau des Programmierhandgeräts erklärt. In Abschnitt 3.1 werden die Hardwarekomponenten des Bedienmoduls erläutert und in Abschnitt 3.2 wird auf das Konzept der GUI eingegangen.
In Abbildung 9 ist der Aufbau der gesamten Arbeit schematisch dargestellt. Mit dem Programmierhandgerät kann über mehrere Eingabegeräte (siehe Abschnitt 3.1) ein simulierter Roboter gesteuert und programmiert werden. Die Simulation läuft auf einem Desktop-Computer und ist in der Programmiersprache Modelica [7] geschrieben. Um Daten zwischen der Simulation und dem Bedienmodul auszutauschen, wird eine am DLR entwickelte Kommunikationsmiddleware, das DLR ModelNet [12], verwendet. Diese sendet mithilfe des UDP-Protokolls Daten über ein Netzwerk.

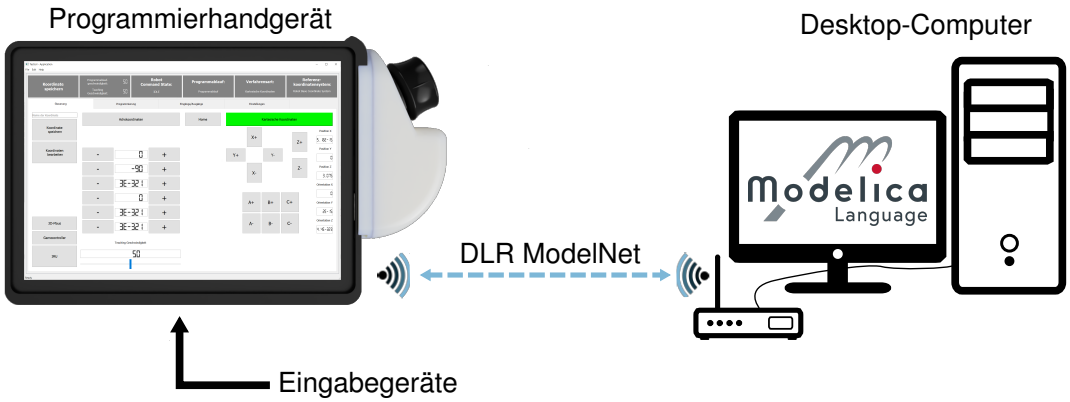


Abbildung 9 Aufbau der Arbeit.

3.1. Konzept Hardware

Bei der Hardwareauswahl wurde versucht, Komponenten auszuwählen, welche sich einfach einbinden lassen und Signale ausgeben, die wenig nachbearbeitet werden müssen. Weiter sollen weitestgehend Bauteile verwendet werden, welche es auf dem Markt zu kaufen gibt. Das zentrale Element der Hardware ist ein Laptop der Firma Dell [23], welcher eine abnehmbare Tastatur besitzt. Durch diese kann der Laptop auch als Tablet-Computer benutzt werden. Auf der Rückseite des Laptops wird ein abnehmbares Gehäuse befestigt, in dem die nötige Hardware verbaut ist, um die Funktionalität eines Programmierhandgeräts zu erreichen. Ein Überblick der gesamten Hardware des Bedienmoduls ist in Abbildung 10 dargestellt.

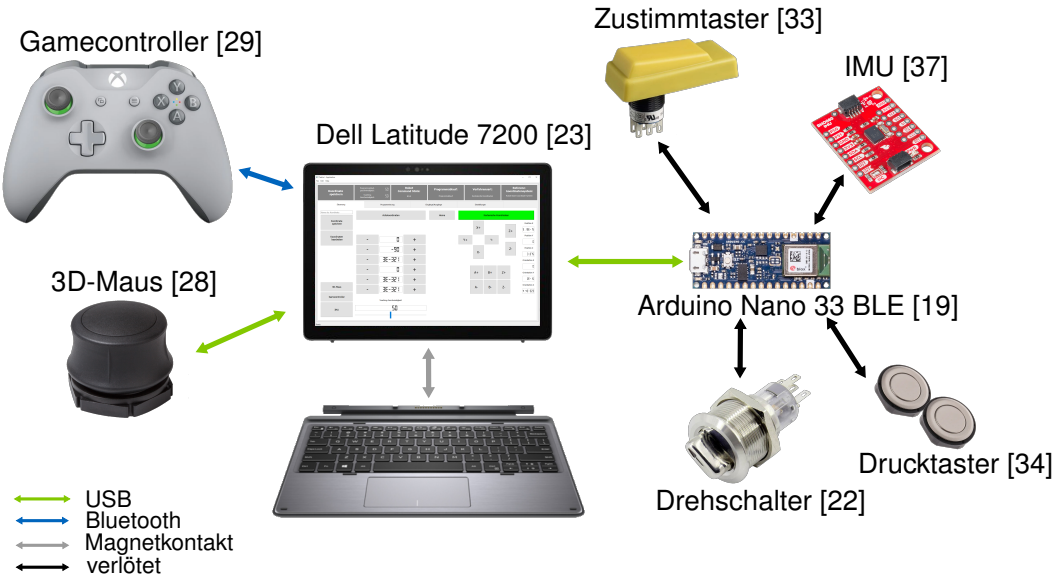


Abbildung 10 Hardwarekonzept der Arbeit.

Für die Steuerung des Roboters wurden drei Eingabegeräte ausgewählt. Diese sind ein Xbox Gamecontroller [29], eine 3D-Maus [28] und eine IMU [37].
Der Gamecontroller wird über Bluetooth mit dem Laptop verbunden und hat zwei Zwei-Achsen-Analogsticks, zwei analoge Druckknöpfe an der Hinterseite, zehn digitale Druckknöpfe und ein Steuerkreuz. Durch die große Menge an Eingabetasten am Controller eignet sich dieser gut, um einen Roboter zu steuern. Ein Nachteil ist, dass der Gamecontroller zweihändig bedient werden muss und dadurch der Zustimmtaster [33] nicht gleichzeitig betätigt werden kann, was aber für den Betrieb eines Industrieroboters notwendig ist.
Die 3D-Maus ist mit dem Laptop per USB-Kabel verbunden und hat sechs analoge Achsen (3x Translation, 3x Rotation). Dadurch können die sechs Freiheitsgrade des TCP im kartesischen Modus sehr einfach gesteuert werden.
Um die Signale der IMU auszulesen, wird ein Mikrocontroller benötigt. Dafür wird ein Arduino nano 33 BLE [19] verwendet. Die IMU hat neben dem Beschleunigungssensor, dem Gyroskop und dem Magnetometer einen Mikroprozessor, welcher die Signale der einzelnen Sensoren fusioniert und den Drift kompensiert. Durch die Sensordatenfusion direkt in der IMU muss der Arduino nur die Werte auslesen und anschließend an den Laptop weitersenden.
Um den Roboter zu Bedienen, werden weitere Schalter und Drucktaster benötigt, welche in Abbildung 10 dargestellt sind. Diese können auch an den Mikrocontroller angeschlossen werden, um so die Zustände an den Laptop zu senden.
Zusätzlich ist zu erwähnen, dass es sich um einen Prototyp handelt, welcher die Funktionsweise demonstrieren soll. Für sicherheitsrelevante Einrichtungen, wie den Zustimmtaster oder den Not-Halt-Schalter muss eine echtzeitfähige Übertragung der Zustände garantiert werden. Dies ist in diesem Prototyp nicht der Fall, weshalb auf einen Not-Halt-Schalter verzichtet wurde. Der Zustimmtaster wurde trotzdem integriert, da dieser zur Entwicklung der Robotersteuerung benötigt wird.

3.2. Konzept GUI

Ziel der GUI ist eine einfache und intuitive Bedienung des Roboters. Es werden nur grundlegende Funktionen umgesetzt, mit denen der simulierte Roboter bewegt und programmiert werden kann. Um die grafische Oberfläche übersichtlich zu halten, wird diese in eine Statusleiste am oberen Rand und vier Tabs, welche sich darunter befinden aufgeteilt (siehe Abbildung 11 - 13).
In der Statusleiste werden Funktionen und Informationen bereitgestellt, welche für den Betrieb des Roboters besonders wichtig sind und deshalb dauerhaft sichtbar sein sollen. Die sechs Segmente der Statusleiste werden im Folgenden von links nach rechts beschrieben. Im ersten Bereich lässt sich per Knopfdruck eine Koordinate abspeichern. Beim Drücken der Schaltfläche werden die aktuellen Koordinaten des Roboters in einer Liste eingetragen. Im nächsten Feld wird die eingestellte Geschwindigkeit in Prozent für das manuelle Steuern sowie für die Ausführung von Programmen dargestellt. Im dritten Segment wird der „Robot Command State“ angezeigt. Dieser sagt aus, ob der Roboter auf ein Kommando wartet (IDLE), ob ein Befehl abgeschlossen ist (FINISHED), ob der Roboter händisch bedient wird (TEACH), ob gebremst wird (BREAK), oder ob der Roboter mithilfe eines Bahnplanungsalgorithmus gesteuert wird. Weiter wird im vierten Bereich abgebildet, ob sich der Roboter im Manuellen-Modus oder im Automatik-Modus (siehe Kapitel 2.4) befindet. Darauf folgt die Darstellung, ob beim Teachen Achskoordinaten oder kartesische Koordinaten verwendet werden. Zuletzt kann noch eingestellt werden, ob als Referenzkoordinatensystem das World oder das TCP Koordinatensystem genutzt wird.

Im folgenden Teil dieses Kapitels wird näher auf die Tabs der GUI eingegangen, welche ähnlich gegliedert sind wie Kapitel 2.
Der erste Tab (siehe Abbildung 11) dient zur Steuerung des Roboters. Dabei kann per Knopfdruck zwischen einer Steuerung in Achskoordinaten und in kartesischen Koordinaten umgeschaltet werden. Zu jedem der beiden Modi gibt es einen Block von Schaltflächen, mit denen der Roboter in eine gewünschte Position gebracht werden kann. Direkt nebenan befindet sich jeweils eine Anzeige, auf der die aktuellen Koordinaten abgelesen werden können. Mit weiteren zwei Schaltflächen kann die Geschwindigkeit, in der sich der Roboter beim Teachen bewegt, eingestellt werden. Darüber hinaus kann durch Drücken der „HOME“-Schaltfläche der Roboter in eine zuvor festgelegte Pose gefahren werden.
Auf der linken Seite befinden sich weitere Schaltflächen, mit denen zum einen Koordinaten gespeichert und bearbeitet werden können und zum anderen ausgewählt werden kann, mit welchem Eingabegerät die Simulation gesteuert wird. Die Schaltfläche „Koordinate speichern“ speichert die aktuelle Pose des Roboters mit einem Namen, der im Eingabefeld darüber eingetragen wird, in einer Liste ab. Wird kein Name eingetragen, soll ein Standardname mit angehängter fortlaufender Zahl benutzt werden. Die Koordinaten können mithilfe der Schaltfläche „Koordinate bearbeiten“ in einem separaten Fenster bearbeitet werden. Mit den weiteren drei Schaltflächen links unten im Bild kann ausgewählt werden, welches Eingabege-

rät zur Steuerung benutzt wird. Beim Drücken der Schaltfläche „3D-Maus“ oder „Gamecontroller“ wird das jeweilige Gerät aktiviert und der Roboter kann bewegt werden. Dabei kann immer nur ein einzelnes Eingabegerät zu den Schaltflächen in der GUI benutzt werden. Um die Orientierung des Roboters mithilfe der IMU zu steuern, muss die Schaltfläche „IMU“ dauerhaft gedrückt bleiben.

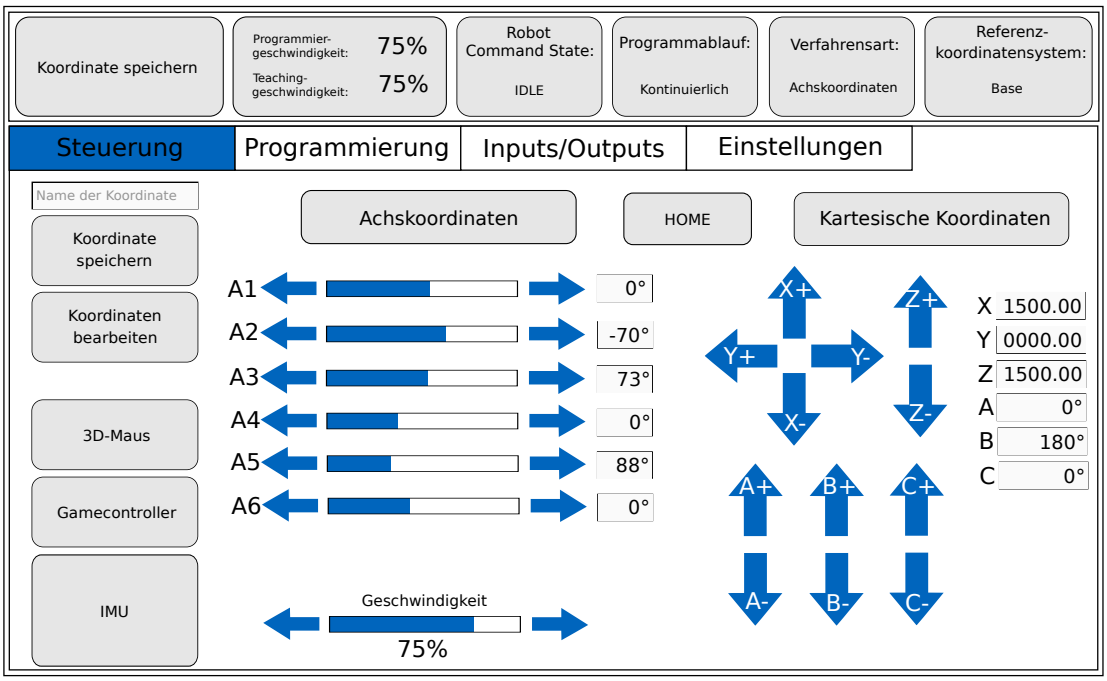


Abbildung 11 Konzept der Bedieneroberfläche (Steuerung).

Mithilfe des zweiten Tabs, welcher in Abbildung 12 dargestellt ist, kann der Roboter programmiert werden. Dafür befindet sich in der Mitte der Bedieneroberfläche ein großer Texteditor. Um die Programmierung übersichtlich zu halten, besitzt der Editor eine Zeilennummerierung sowie eine Hervorhebung der Zeile, die aktuell bearbeitet wird. Weiter gibt es eine Syntaxhervorhebung und die Möglichkeit Codebereiche zu reduzieren und zu erweitern. Um den Cursor an einer gewünschten Position zu setzen, wird die Touchscreeneingabe des Laptops benutzt, welche auch genutzt wird, um Bereiche eines Programms zu markieren.
Auf der linken Seite des Texteditors gibt es mehrere Schaltflächen, mit denen es möglich ist, den Code im Texteditor zu manipulieren und Programme zu speichern und zu laden. Die ersten zwei Schaltflächen erfüllen die gleichen Funktionen wie die Eingabetaste und die Backspace-Taste auf einer Tastatur. Mit den nächsten beiden Schaltflächen ist es möglich, einen markierten Bereich zu kopieren und einzufügen. Zusätzlich muss es noch die Möglichkeit geben, neue Programme zu öffnen und geschriebene Programme abzuspeichern. Dies ist mit den letzten drei Schaltflächen realisiert. Mit der „Neu“ Schaltfläche wird ein leeres Dokument geöffnet, mit „Öffnen“ kann ein Programm aus dem Speicher des Computers geladen werden und mit „Speichern“ lässt sich ein geschriebenes Programm abspeichern.

Um Code im Texteditor zu schreiben, kann zum einen die abnehmbare Tastatur des Laptops benutzt werden, gleichzeitig ist es auch möglich, einfache Programme mithilfe der GUI zu erstellen. Dafür werden auf der rechten Seite des Texteditors Schaltflächen integriert. Beim Drücken der Schaltfläche „Bewegungsbefehl“ öffnet sich ein neues Fenster, in dem ein Bewegungsbefehl und eine Koordinate ausgewählt werden kann. Diese werden bei Bestätigung an die aktuelle Position des Cursors geschrieben. Durch Betätigen der Schaltfläche „Anweisung“ öffnet sich ein weiteres Fenster, in dem zwischen verschiedenen Anweisungen wie if-else, for oder while ausgewählt werden kann. Diese werden anschließend in den Texteditor geschrieben. Um den Code zu ändern, gibt es die Schaltfläche „Bearbeiten“. Damit lässt sich zum Beispiel die Koordinate oder der Bewegungsbefehl anpassen.
Um den geschriebenen Code an den Roboter zu senden, gibt es eine Start- und eine Stop-Schaltfläche und die Schaltfläche „Zeilenbefehl ausführen“. Mit Letzterem lässt sich nur der Befehl einer einzelnen Zeile ausführen, wogegen mit Start das gesamte Programm ausgeführt wird.

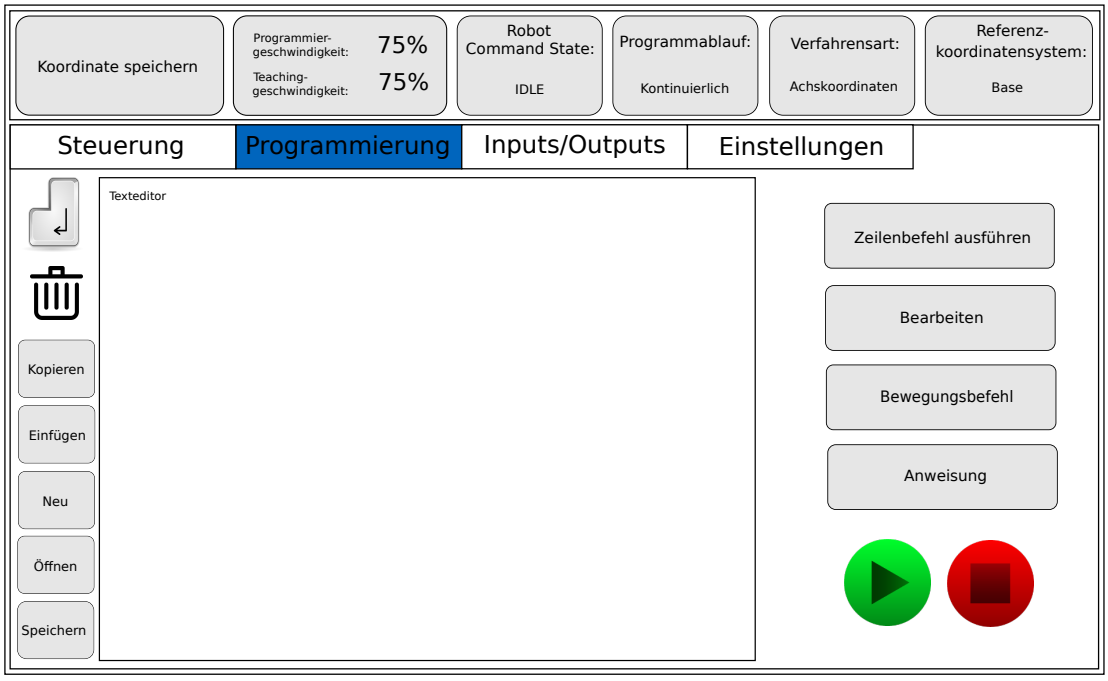


Abbildung 12 Konzept der Bedieneroberfläche (Programmierung).

Mit dem Tab Inputs/Outputs aus Abbildung 13 können Eingänge und Ausgänge der Robotersimulation angezeigt und verändert werden. Diese gibt es in digitaler und analoger Form. Im oberen Bereich des Fensters können die digitalen Ausgänge verändert werden und die digitalen Eingänge werden angezeigt. Im unteren Bereich wurden die gleichen Einstellmöglichkeiten und Anzeigen für die analogen Signale integriert.

Im letzten Tab können verschiedene Einstellungen vorgenommen werden. Dafür wird das Fenster in weitere Bereiche unterteilt. Als Erstes gibt es Einstellmöglichkeiten für den TCP. Dazu gehört zum Beispiel die Möglichkeit, den TCP in einem gewünschten Offset vom Flansch des Roboters zu platzieren. Als Nächstes können die Eingänge und Ausgänge angepasst werden. Dabei besteht die Möglichkeit, die Anzahl zu verändern und diese zu benennen. Um die Hardware zu testen und einzustellen, gibt es einen Tab für die Eingabegeräte. Darin kann der Status der einzelnen Druckknöpfe und Schalter des Gehäuses abgelesen werden. Weiter ist es möglich, die IMU zu Kalibrieren, den seriellen Bus zurückzusetzen und die Größe der Totzonen (siehe Kapitel 4.1.1) für die Analogsticks des Gamecontrollers sowie die 3D-Maus einzustellen. Zuletzt können noch Einstellungen für den Texteditor vorgenommen werden, wie zum Beispiel die Schriftgröße zu verändern.
Für ein kommerzielles Programmierhandgerät sind noch weitere Einstellmöglichkeiten notwendig wie maximale Beschleunigungen/Geschwindigkeiten oder Möglichkeiten zur Einrichtung des Roboters bei der Inbetriebnahme. Darauf wird aber hier nicht weiter eingegangen.

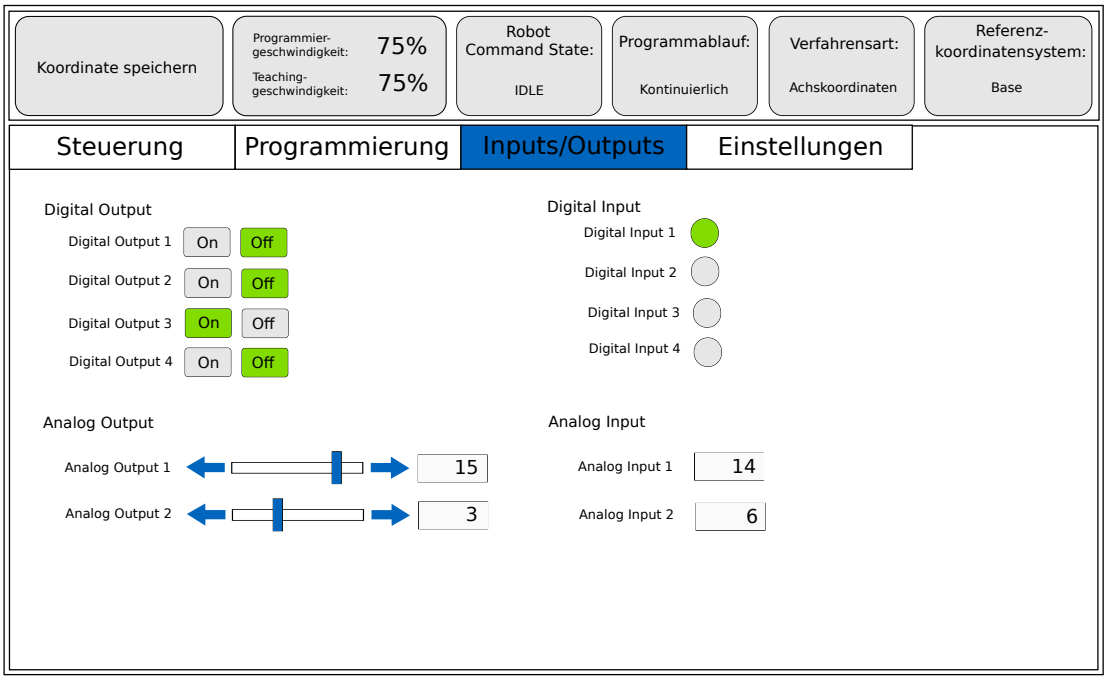


Abbildung 13 Konzept der Bedieneroberfläche (Eingänge/Ausgänge).

4. Implementierung

In diesem Kapitel wird die Umsetzung der Konzepte näher erläutert. Im ersten Unterkapitel wird auf die Implementierung der Software eingegangen und im Zweiten auf die Verkabelung des Arduinos und den Aufbau des Gehäuses.

4.1. Software

Die Software ist in dieser Arbeit in zwei große Bereiche zu unterteilen. Die Programmierung der GUI und die Programmierung des Simulationsmodells. Die GUI wird mit der C++/Qt-Entwicklungsumgebung „Qt Creator“ programmiert. Diese besitzt einen GUI-Designer (Qt Designer), in dem eine Benutzeroberfläche mithilfe von Bausteinen aufgebaut werden kann. Zur Programmierung der Simulation, welche mit dem Programmierhandgerät bedient werden soll, wird Dymola verwendet. Dies ist eine Modellierungs- und Simulationsumgebung, welche auf Modelica aufbaut. Bei Modelica handelt es sich um eine objektorientierte Programmiersprache, welche zum Modellieren von dynamischen Systemen verwendet wird [7].

4.1.1. Grafische Benutzeroberfläche

In Kapitel 3.2 wurde bereits das Konzept der Benutzeroberfläche vorgestellt. Hier wird zuerst auf grundlegende Konzepte des Qt-Frameworks eingegangen. Im Anschluss werden Details der tatsächlichen Implementierung erklärt.
Für eine Qt-Benutzeroberfläche werden in der Hauptfunktion (main()) immer mindestens zwei Objekte erstellt. Ein Objekt zur Verwaltung der Ressourcen (QApplication) und ein oder mehrere Objekte zur Darstellung von Widgets bzw. Fenstern. Das Wort Widget setzt sich aus „Window“ (Fenster) und „Gadget“ (Vorrichtung) zusammen und beschreibt ein sichtbares Element der Benutzeroberfläche. Ein Widget kann auch als Container für andere Widgets benutzt werden.
Im Folgenden ist die Hauptfunktion abgebildet:

```
1  #include "teachinmodule.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6  QApplication a(argc, argv);
7  TeachInModule w;
8  w.show();
9
10 return a.exec();
11 }
```

In den ersten beiden Zeilen des Codes werden die nötigen Header-Dateien eingebunden. In der sechsten Zeile wird ein Objekt zur Ressourcenverwaltung erstellt und in Zeile sieben ein

Objekt, welches das Hauptfenster der GUI definiert. Das Hauptfenster dient als Container für weitere Widgets, welche in der Klasse „TeachInModule“ definiert sind. Weiter wird mithilfe von Zeile acht das Hauptfenster angezeigt und mit der zehnten Zeile die Anwendung gestartet.
Beim Starten der Anwendung wir eine Ereignis-Schleife („event-loop“) gestartet, welche Ereignisse wie zum Beispiel Mauseingaben, Touchscreeneingaben oder Tastatureingaben in einem Objekt detektiert, worauf das Objekt dann reagieren kann. Zusätzlich arbeitet das Qt-Framework mit dem Signal-Slot-Konzept, welches einen ereignisgesteuerten Programmfluss ermöglicht. Dabei werden als Reaktion auf Ereignisse Signale gesendet (emittiert), welche mit einem Slot verbunden werden können. Das bedeutet, dass beim Auslösen eines Signals der verbundene Slot ausgeführt wird. Ein Slot ist im Prinzip eine normale Funktion und es können mehrere Slots mit einem Signal verbunden werden. Diese werden dann nacheinander ausgeführt. Zusätzlich ist es möglich, Signale von einem Thread zu einem anderen Thread zu senden.
Ein Beispiel für eine Verbindung zwischen einem Signal und einem Slot ist:

```
1 QObject::connect(pushButton, &TouchButton::pressed, this, &TeachInModule::updateIMUdata);
```

Für diesen „connect“-Befehl müssen vier Parameter übergeben werden. Als Erstes wird ein Zeiger auf ein Objekt übergeben, welches das Signal abschickt. Da es in einer Klasse mehrere Signale geben kann, beschreibt der zweite Parameter, um welches Signal es sich handelt. Dann muss ein Zeiger auf das Objekt übergeben werden, in dem sich der Slot befindet, der als Reaktion auf das Signal ausgeführt werden soll. Zum Schluss wird angegeben, welcher Slot aufgerufen wird. Sprich in dem gezeigten Beispiel wird ein Signal gesendet, wenn eine Schaltfläche gedrückt wird und als Reaktion darauf wird der Slot „updateIMUdata“ ausgeführt.

Im Qt-Framework ist die Klasse „QObject“ das Schlüsselelement der meisten Funktionalitäten. So auch bei dem zuvor erklärten Signal-Slot-Konzept und beim Detektieren von Ereignissen. Jedes Qt-Objekt besitzt diese Klasse als Basisklasse.
Ein weiteres wichtiges Konzept ist die Strukturierung der Instanzen als Baumstruktur. Wenn eine Instanz von „QObject“ erstellt wird, dann wird im Konstruktor ein Elternobjekt übergeben. Dieses übernimmt den Besitz des Objekts und fügt es zu seiner Liste von Kindern hinzu. So ist es zum Beispiel möglich, dass beim Löschen eines Elternobjekts auch alle Kindobjekte gelöscht werden. Dadurch wird die Speicherverwaltung vereinfacht.
Im Folgenden wird ein Beispiel dargestellt, wie ein Elternobjekt und ein zugehöriges Kind erzeugt werden:

```
1 QObject *parent = new QObject();
2 QObject *child = new QObject(parent);
3 QObject *child2 = new QObject(child);
```

In der ersten Zeile wird im Konstruktor nichts übergeben. Dabei handelt es sich um das erste Elternobjekt welches alle anderen Objekte enthält. In Zeile zwei wird ein Zeiger auf das erste

Objekt übergeben. Deshalb ist das zweite Objekt ein Kind des Ersten. Weiter ist es möglich, wie in Zeile drei gezeigt, Kinder von Kindern zu erstellen.
Für weitere Details welche das Qt-Framework betreffen wird hier auf die Qt Dokumentation verwiesen [39].

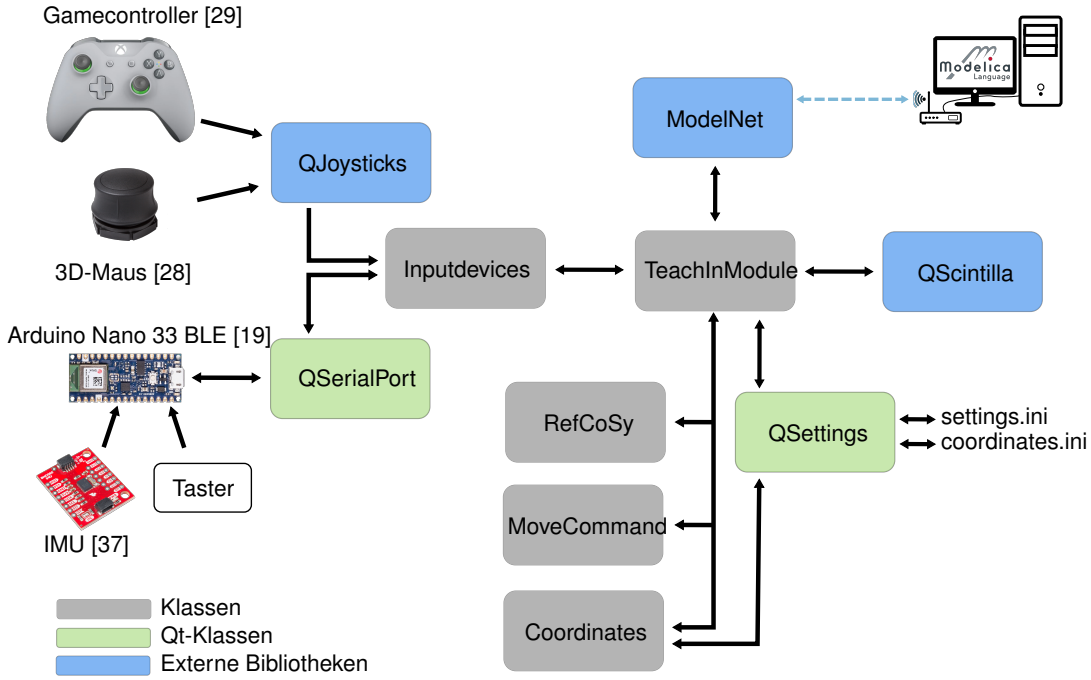


Abbildung 14 Aufbau der GUI.

In Abbildung 14 sind die wichtigsten Klassen und Bibliotheken, welche zur Programmierung der GUI benutzt werden, dargestellt. Im mittleren Bereich der Abbildung befindet sich die Klasse „TeachInModule“, von der ein Objekt in der Hauptfunktion erstellt wird. Diese ist das zentrale Element des Programms, welche alle verwendeten Klassen miteinander verknüpft. Mithilfe der Klasse „Inputdevices“ werden die Daten des Gamecontrollers, der 3D-Maus und des Arduino eingelesen und weiterverarbeitet. Um die Daten des Gamecontrollers und der 3D-Maus in die Software einzuspielen, wird eine Bibliothek mit dem Namen „QJoysticks“ [38] verwendet. Der Arduino sendet die Zustände der verschiedenen Taster und der IMU über eine serielle Schnittstelle an den Laptop, welche dann mit der vom Qt-Framework bereitgestellten Klasse „QSerialPort“ ausgelesen werden. Die Bibliothek mit dem Namen „QScintilla“ [32] ist eine Open-Source-Texteditor-Komponente, welche benutzt wird, um den Codeeditor zu erstellen. Damit ist es möglich, Features wie zum Beispiel Syntaxhervorhebung oder Zeilennummerierung umzusetzen. Im unteren Bereich von Abbildung 14 befinden sich drei Klassen, welche jeweils ein Popup-Fenster darstellen. Mithilfe von „RefCoSy“ kann das Koordinatensystem ausgewählt werden, in dem sich der TCP des simulierten Roboters bewegt. Dabei kann zwischen einem Koordinatensystem im Ursprung des Roboters und einem im TCP gewählt werden. Die Klasse „MoveCommand“ öffnet ein Fenster, in dem per Dropdown-Liste ein Bewegungsbefehl und eine gespeicherte Koordinate ausgewählt werden kann. Die-

ser Befehl wird nach Bestätigung in den Codeeditor an die Stelle des Cursors geschrieben. Weiter kann mit der Klasse „Coordinates“ ein Fenster geöffnet werden, in dem alle gespeicherten Koordinaten dargestellt sind. Zusätzlich werden die Koordinaten durch diese Klasse verwaltet und mithilfe der Klasse „QSettings“ in eine INI-Datei geschrieben. Somit sind die Koordinaten auch nach einem Neustart der GUI weiter vorhanden. Zuletzt werden mithilfe der „ModelNet“ Bibliothek die erforderlichen Daten von der Benutzeroberfläche an die Robotersimulation gesendet und Daten von der Simulation empfangen.
Im Folgenden wird im Detail auf einige Klassen aus Abbildung 14 eingegangen.

Die **„QJoysticks“**-Bibliothek bedient sich zum Einlesen der 3D-Maus und des Gamecontrollers einer Bibliothek mit dem Namen SDL (engl. Simple Directmedia Layer) [36]. Diese ermöglicht Low-Level-Zugriff auf Eingabegeräte wie zum Beispiel Tastatur, Maus oder Joysticks. Mithilfe der SDL-Bibliothek wird der Status der 3D-Maus und des Gamecontrollers im 10ms Takt abgefragt (gepollt). Um dies zu erreichen, wird die „QTimer“ Klasse des Qt-Frameworks benutzt. Damit lässt sich nach Ablauf einer vorgegebenen Zeit ein Signal senden, welches die Abfrage der Eingabegeräte auslöst. Dies kann auf zwei Arten ausgeführt werden:

```
1 QTimer *timer = new QTimer(this);
2 QObject::connect(_timer, &QTimer::timeout, this, &TeachInModule::sendData);
3 timer->setTimerType(Qt::PreciseTimer);
4 timer->start(10);
5
6 QTimer::singleShot (10, Qt::PreciseTimer, this, &MainWindow::update);
```

Die erste Möglichkeit ist in Zeile eins bis vier dargestellt. Dabei wird in der ersten Zeile ein Objekt der Klasse „QTimer“ angelegt. Daraufhin wird das Signal „timeout“ mit einem Slot in diesem Beispiel „sendData“ verbunden. Weiter gibt es die Möglichkeit, die Genauigkeit des Timers einzustellen. Mithilfe der Option „Qt::PreciseTimer“ versucht die Anwendung die Abweichung beim Auslösen des Signals unter einer Millisekunde zu halten. Abschließend wird der Timer in Zeile vier gestartet, wobei die gewünschte Time-out Zeit in Millisekunden als Parameter übergeben wird. Der Vorteil dieser Methode ist, dass der in Zeile vier gestartete Timer mit dem Befehl „timer->stop()“ wieder angehalten werden kann. So kann der Aufruf von Funktionen bei Bedarf angehalten und gestartet werden, womit der Rechenaufwand reduziert werden kann.
In der zweiten Methode, welche in Zeile sechs dargestellt ist, wird ein Timer einmalig gestartet und nach dem eingestellten Zeitintervall ein Slot ausgeführt. Um diesen wiederholt auszuführen, wird ans Ende der Funktion der gleiche Befehl geschrieben, womit sich der Slot selbst aufruft. So können gezielt Slots wiederholt ausgeführt werden oder bei Bedarf auch einmalig nach einem gewissen Zeitintervall.
In der „QJoysticks“ Bibliothek wird die zweite Variante verwendet, um die Zustände der Eingabegeräte abzufragen.
Um die Daten vom Arduino in der Qt-Anwendung auszulesen, wird die Klasse **„QSerialPort“**

verwendet. In Kapitel 4.2 wird beschrieben, wie die Nachricht aufgebaut ist. Weiter kann eine Auflistung der übertragenen Daten in Tabelle 3 gefunden werden.
Die Qt-Klasse besitzt das Signal „canReadLine“, welches ausgelöst wird, wenn eine vollständige Nachricht am Laptop angekommen ist. Daraufhin wird die Zeile ausgelesen und die Werte in die dafür vorgesehenen Datenstrukturen der Klasse „Inputdevices“ geschrieben.

Die Klasse **„Inputdevices“** dient zum Sammeln und Zwischenspeichern der Eingabegerätedaten sowie zum Aufbereiten der Signale. Hier sind für alle Eingabegeräte Datenstrukturen vorgesehen, welche die nötigen Informationen speichern. Wenn die Klassen „QSerialPort“ und „QJoysticks“ ein Signal ausgeben, dass neue Daten verfügbar sind, werden diese in die Datenstrukturen gefüllt. Folglich kann von den übergeordneten Klassen auf diese Daten zugegriffen werden.
Weiter werden Funktionen bereitgestellt, mit denen eine Totzone eingerichtet werden kann. Eine Totzone ist ein Bereich, in dem ein Analogstick oder die 3D-Maus bewegt werden kann, der Ausgabewert aber auf null bleibt. Dies ist in zwei Bereichen sinnvoll. Um den Ursprung und entlang der Achsen. Im Folgenden wird die Totzone für die Analogsticks des Gamecontrollers erläutert. Diese funktioniert auf die gleiche Weise für die Achsen der 3D-Maus.
Um den Ursprung wird eine Totzone integriert, da ein Analogstick in seiner Neutralstellung oft eine kleine Abweichung hat, welche so beseitigt werden kann. Zusätzlich wird dadurch verhindert, dass sich der Wert des Analogsticks bei kleinen ungewollten Bewegungen um den Ursprung verändert. Dies ist beispielsweise der Fall, wenn der Gamecontroller in die Hand genommen oder Vibrationen ausgesetzt wird.
Weiter wird eine Totzone entlang der Achsen der Analogsticks implementiert. Diese ist nötig, da sich bei der Bewegung eines Zwei-Achsen-Analogsticks entlang einer einzelnen Achsrichtung bei der kleinsten Abweichung von der Achse auch der zweite Achsenwert verändert. Durch die Totzone wird so die zweite Achse beschränkt. Die verwendete Totzone wurde aus einer Arbeit von Pérez Ramil [31] übernommen.

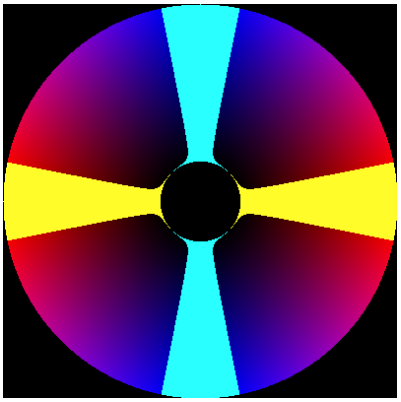


Abbildung 15 Implementierte Totzone, aus [31].

In Abbildung 15 ist die Totzone eines Zwei-Achsen-Analogsticks abgebildet. Der schwarze Kreis in der Mitte stellt einen Bereich dar, in dem der ausgegebene Wert in beiden Achsen null entspricht. Der gelbe Bereich verläuft entlang der x-Achse und der Türkise entlang der y-Achse. So lange sich der Analogstick im gelben Bereich befindet, wird der Wert der y-Achse immer auf null gesetzt. In gleicher Weise wird der Wert der x-Achse auf null gesetzt, wenn sich der Analogstick im blauen Bereich befindet. So kann der Roboter einfacher entlang einer Achse bewegt werden.

In der Klasse **„TeachInModule“** werden alle Klassen miteinander verknüpft und alle Bausteine bzw. Objekte der GUI erstellt. Weiter sind in Anhang A in den Abbildungen 24a bis 24d Screenshots der einzelnen Tabs der implementierten GUI dargestellt. Eine Erklärung der Funktionsweise der Bedieneroberfläche wurde bereits in Kapitel 3.2 gegeben.
Um die nötigen Daten, welche in Tabelle 2 dargestellt sind, von der Qt-Anwendung zur Simulation zu senden, wird die **„ModelNet“**-Bibliothek benutzt. Damit werden mithilfe der „QTimer“-Klasse jede Millisekunde die Daten aus Tabelle 1 empfangen und aus Tabelle 2 gesendet. Mit diesen Daten kann der simulierte Roboter über das Programmierhandgerät gesteuert und programmiert werden.
Im Folgenden werden die Daten aufgelistet, welche von der Simulation an die GUI gesendet werden:

Tabelle 1 Daten, die von der Simulation an die GUI gesendet werden.

Daten	Erklärung
aktuelle Roboterposition	Aktuelle Roboterposition in Achskoordinaten und kartesischen Koordinaten. Wird verwendet, um die aktuelle Position im Steuerung-Tab (Abb. 24a) anzuzeigen.
Roboter Status	Zustand, in dem sich der Roboter befindet. Wird in der Statusleiste angezeigt („Robot Command State“).
Kommando bestätigt	Variable, welche bestätigt, dass ein Kommando empfangen wurde. Wird beim Senden von Strings aus dem Programmierung-Tab (Abb. 24b) und beim Drücken der Home-Schaltfläche verwendet.
digitale Eingänge/Ausgänge	Status der digitalen Ein- und Ausgänge, welche im Eingänge/Ausgänge-Tab (Abb. 24c) angezeigt werden.
analoge Eingänge/Ausgänge	Status der analogen Ein- und Ausgänge, welche im Eingänge/Ausgänge-Tab (Abb. 24c) angezeigt werden.

Weiter werden in der nächsten Tabelle die Daten, welche von der GUI an die Simulation gesendet werden, dargestellt:

Tabelle 2 Daten, die von der GUI an die Simulation gesendet werden.

Daten	Erklärung
Modus	Variable, welche einen Wert zwischen eins und vier sendet. Dieser beschreibt in welchem Koordinatensystem der Roboter beim Teachen gesteuert wird (1 - Achskoordianten, 2 - World Koordinatensystem, 3 - TCP Koordinatensystem, 4 - IMU Orientierung).
Kommando gesendet	Variable, welche der Simulation mitteilt, dass ein Kommando gesendet wurde. Wird beim Senden von Strings aus dem Programmierung-Tab (Abb. 24b) und beim Drücken der Home-Schaltfläche verwendet.
Kommando	Variable, welche das Roboterprogramm als Zeichenfolge überträgt.
Geschwindigkeiten	Die Geschwindigkeiten in Prozent (als Wert zwischen 0 und 1) für das Teachen und den Ablauf eines Roboterprogramms.
Status Druckknöpfe	Der Status der Druckknöpfe, des Drehschalters und des Zustimmtasters, welche im Gehäuse verbaut sind.
Teachen aktiv	Versetzt die Simulation in den Teach-Modus.
Bremsen aktiv	Versetzt die Simulation in den Brems-Modus.
Teach Schaltflächen	Zustände der Teach Schaltflächen der Achskoordinaten und der kartesischen Koordinaten aus der GUI.
3D-Maus	Zustände der Achsen der 3D-Maus.
Gamecontroller	Zustände der Analogsticks und Druckknöpfe des Gamecontrollers.
IMU	Orientierung der IMU als Quaternion.
Eingabegerät Auswahl	Variable, welche der Simulation mitteilt, welches Eingabegerät benutzt werden soll.

4.1.2. Modelica

Zur Modellierung und Simulation von seriellen Robotern wurde am Institut für Systemdynamik und Regelungstechnik des DLR eine Modelica-Bibliothek mit dem Namen „DLR Robots library“ aufgebaut [2]. Darin wurde bereits das Modell einer Robotersteuerung umgesetzt, welches im Zuge dieser Arbeit erweitert wurde. In diesem Abschnitt wird beschrieben, wie die Robotersteuerung aufgebaut ist.
Für die Programmierung in Modelica wurde die grafische Entwicklungsumgebung Dymola benutzt. Darin können Simulationsmodelle mithilfe von Blöcken aufgebaut werden.
In Abbildung 16 ist der Block der Robotersteuerung dargestellt.

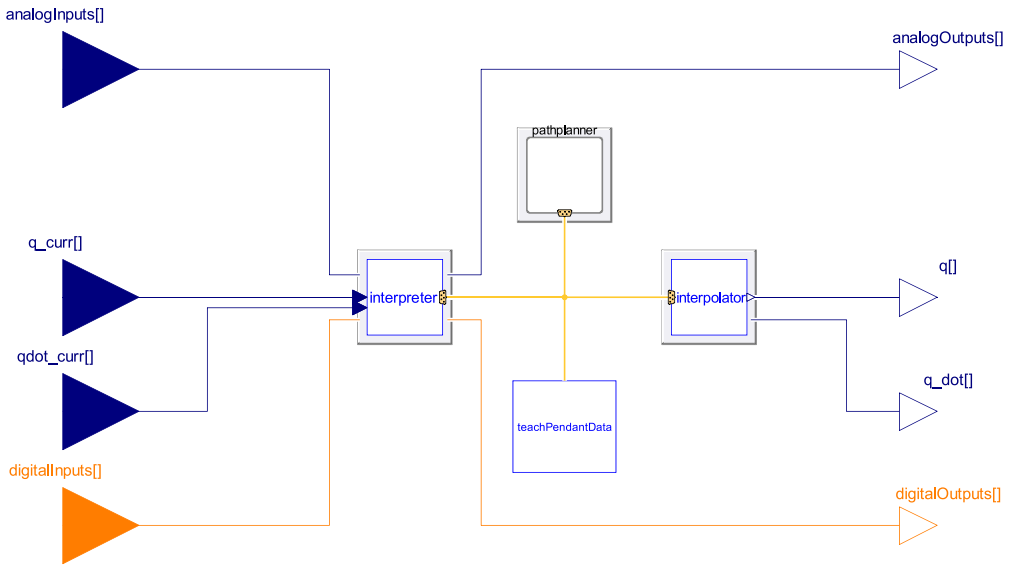


Abbildung 16 Robotersteuerungsblock aus Dymola.

Es gibt vier Eingänge und vier Ausgänge für den Robotersteuerungsblock. Als Eingänge werden die analogen Eingänge (analogInputs[]), die aktuellen Achspositionen des Roboters (q_curr[]), die aktuellen Achsgeschwindigkeiten des Roboters (qdot_curr[]) und die digitalen Eingänge (digitalInputs[]) übergeben. Die Robotersteuerung wird in vier Blöcke unterteilt. Der Block mit dem Namen „teachPendantData“ dient zum Austausch der Daten mit der GUI. Über den „controlBus“ (gelbe Linien) gibt „teachPendantData“ diese an die anderen Blöcke weiter. Der „interpreter“ ist für die Ausführung von Roboterprogrammen zuständig. Darin befindet sich ein Lua Interpreter. Lua ist die Programmiersprache, in der die Roboterprogramme geschrieben sind. Hinter dem „interpolator“ verbirgt sich ein einfaches PT2-Glied mit einer sehr kleinen Zeitkonstante (50 ms). Weiter ist der „pathplanner“ dafür zuständig beim Abfahren von Roboterprogrammen sowie beim Teachen des Roboters die Achswinkel vorzugeben.

Als Ausgänge besitzt der Robotersteuerungsblock analoge Ausgänge (analogOutputs[]), die Soll-Position des Roboters (q[]), die Soll-Geschwindigkeit (q_dot[]) und digitale Ausgänge (digitalOutputs[]).
Weiter wird der „pathplanner“ im Detail beschrieben. Der Aufbau ist in Abbildung 17 dargestellt.

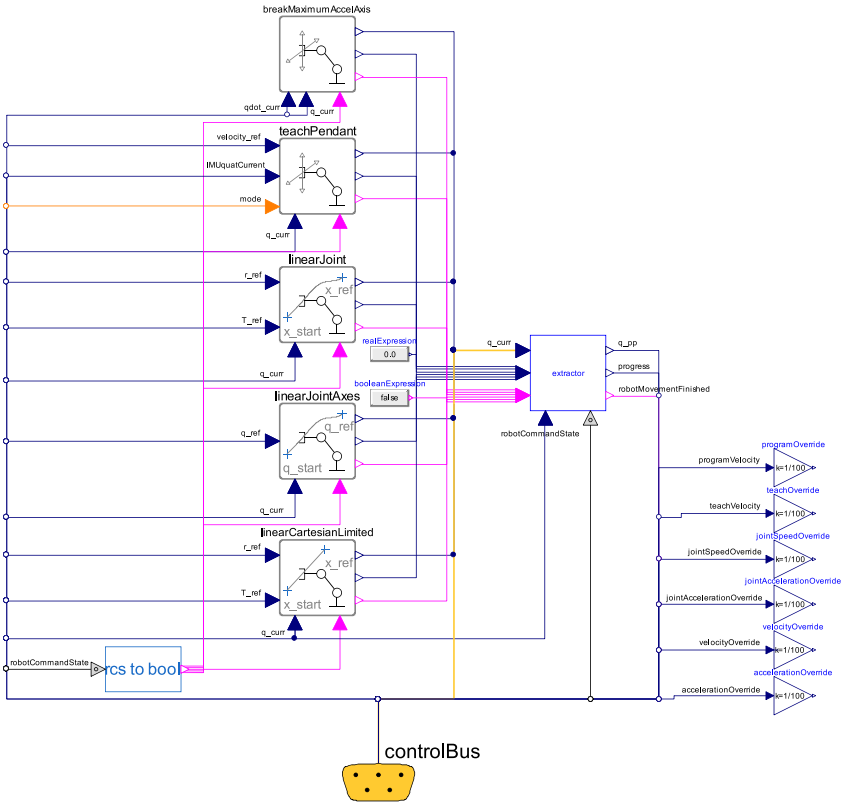


Abbildung 17 Pfadplanungs-Block aus Dymola.

Der „pathplanner“ besitzt fünf Blöcke, um für verschiedene Zustände des Roboters die Achswinkel vorzugeben. Diese sind in der Mitte des Bildes dargestellt. Daraus wird mit dem „extractor“ einer der Achswinkel ausgewählt und an den Roboter weitergeleitet. Welcher Ausgang weitergeleitet wird, wird durch den „robot command state“ bestimmt. Dieser kann mit dem Programmierhandgerät oder durch ein Roboterprogramm verändert werden.
In dieser Arbeit wurden die oberen zwei Blöcke in der Mitte von Abbildung 17 hinzugefügt. Der Block ganz oben mit dem Namen „breakMaximumAcceleration“ dient zum Abbremsen des Roboters nach Abbruch eines Roboterprogramms. Ohne diesen würde der Roboter beim Abbruch eine Vollbremsung durchführen, wobei die maximalen Beschleunigungen der einzelnen Achsen überschritten werden können und ein realer Roboter beschädigt werden kann. Das soll aber nur beim Aktivieren des Not-Halts geschehen. Durch diesen Block werden die Achsen des Roboters beim Abbruch eines Programms mit einer definierten Beschleunigung verzögert. Das bedeutet, die Achsen des Roboters bewegen sich noch ein kleines Stück weiter als bei einer Vollbremsung.

Der zweite Block mit dem Namen „teachPendant“ ermöglicht das manuelle Verfahren des Roboters. Es wird mit den verschiedenen Eingabegeräten eine Geschwindigkeit (velocity_ref) vorgegeben, welche den Roboter in Bewegung versetzt. Eine Ausnahme gibt es bei der Steuerung mit der IMU. Dabei wird direkt die Orientierung der IMU (IMUquatCurrent) als Quaternion übergeben. Nach Aktivierung der IMU-Steuerung wird die Änderung der Orientierung des Programmierhandgeräts bestimmt und diese auf den TCP des Roboters übertragen. Mithilfe der Variable „mode“, welche einen Eingang des „teachPendant“-Blocks darstellt, wird festgelegt, ob der Roboter in Achskoordinaten, in kartesischen Koordinaten oder durch die IMU gesteuert wird.
Die restlichen drei Blöcke dienen zur Bahnplanung, wenn ein Roboterprogramm abgefahren wird. Auf diese wird hier nicht näher eingegangen, da diese schon vorhanden waren und nicht als Teil dieser Arbeit implementiert wurden.

4.2. Hardware

Ein Überblick der im Gehäuse verbauten Komponenten wurde bereits in Abschnitt 3.1 gegeben. Weiter wurde in Kapitel 4.1.1 erläutert, wie die Daten des Gamecontrollers, der 3D-Maus und des Arduino in die GUI eingelesen werden. In diesem Abschnitt wird näher erläutert, wie die Hardwarekomponenten mit dem Arduino verbunden werden und wie das Gehäuse aufgebaut ist.

4.2.1. Arduino

Die Verkabelung der am Arduino angeschlossenen Bauteile wird in Abbildung 18 dargestellt. Um die Zustände der Drucktaster, des Zustimmtasters und des Drehschalters im Mikrocontroller einzulesen, werden diese jeweils an einen digitalen I/O Pin (D4 - D8) und an die Masse (GND) angeschlossen. Die Bauteile haben die Funktion eines Schalters, welcher einen Stromkreis öffnen oder schließen kann. Wenn ein digitaler Eingang eines Arduino mit einem Schalter verbunden wird, muss immer dafür gesorgt werden, dass ein definierter Zustand anliegt (LOW oder HIGH). Daher wird jeder Eingang zusätzlich mit einem internen Pull-Up-Widerstand des Arduino verbunden. Es handelt sich dabei um hochohmige Widerstände, welche die Eingänge des Mikrocontrollers mit der positiven Versorgungsspannung verbinden. Bei geöffnetem Stromkreis liegt somit immer der Zustand HIGH an. Wird ein Schalter geschlossen, wird die positive Versorgungsspannung mit der Masse verbunden, wodurch die Spannung über den Widerstand abfällt. In diesem Fall liegt am Eingang des Arduinos der Zustand LOW an.
Die IMU ist mit dem Arduino über vier Kabel verbunden. Zwei davon werden für die Stromversorgung (3V3, GND) genutzt und zwei für den I²C-Datenbus (A4/SDA, A5/SCL). Die Informationen, welche der Mikrocontroller über den Datenbus ausließt, werden von einem Mikroprozessor auf der IMU bereitgestellt. Auf diesem laufen Signalverarbeitungsalgorithmen, mit denen die Signale der Sensoren auf der IMU fusioniert werden. Dabei gibt es nicht nur Algorithmen, um Orientierungen oder Beschleunigungen auszugeben, sondern auch welche, um

beispielsweise Schritte zu zählen oder Bewegungen zu klassifizieren. Um die Daten von der IMU auslesen zu können, wird von Sparkfun eine Arduino Bibliothek [30] bereitgestellt. Für diese Arbeit ist nur die Orientierung der IMU interessant. Es gibt die Möglichkeit verschiedene Koordinatensysteme auszulesen, welche sich in der Referenzierung, der Anzahl der Sensoren die fusioniert werden und im Algorithmus unterscheiden. Dies hat einen Einfluss auf die Genauigkeit der Orientierung und auf die Abtastfrequenz, mit der die Daten ausgelesen werden können.
In dieser Arbeit wird der „Rotation Vector“ verwendet. Dieser bietet die genaueste Orientierung mit einem statischen Rotationsfehler von zwei Grad, einem dynamischen Rotationsfehler von 3.5 Grad und einer maximalen Abtastfrequenz von 400 Hz [21, S. 51]. Die Orientierung wird bezüglich eines Koordinatensystems ausgegeben, welches in Richtung der Gravitationskraft und der magnetischen Nordrichtung ausgerichtet ist. Für den „Rotation Vector“ werden die Daten des Beschleunigungssensors, des Gyroskops und des Magnetometers fusioniert.
Mithilfe der Arduino Bibliothek kann eingestellt werden, mit welcher Frequenz die Werte aus der IMU ausgelesen werden. Dafür wird die maximale Frequenz von 400 Hz gewählt. Weiter wird die dauerhafte Kalibrierung aktiviert, um eine möglichst genaue Ausgabe der Orientierung zu erreichen. Die nötigen Schritte zur Kalibration können im Datenblatt der IMU unter Kapitel 3.2 nachgelesen werden [21, S. 40].

Tabelle 3 Daten, die vom Arduino an die GUI gesendet werden.

Daten	Erklärung
IMU Orientierung	Orientierung der IMU ausgegeben als Quaternion.
Kardanwinkel	Orientierung der IMU ausgegeben in Roll, Nick und Gierwinkel.
Kalibriergenauigkeit	Variablen, welche die Genauigkeiten der verschiedenen Sensoren auf der IMU angeben. Diese werden von der IMU bereitgestellt und haben folgende Bedeutung: 0 - Unzuverlässig, 1 - Niedrig, 2 - Mittel, 3 - Hoch. Die Variablen werden zum Kalibrieren der IMU im Einstellungen-Tab (Abb. 24d) benutzt.
Zustimmtaster	Zustand des Zustimmtasters im Gehäuse.
Drehschalter	Zustand des Drehschalters im Gehäuse.
Drucktaster	Zustände der drei Drucktaster im Gehäuse.

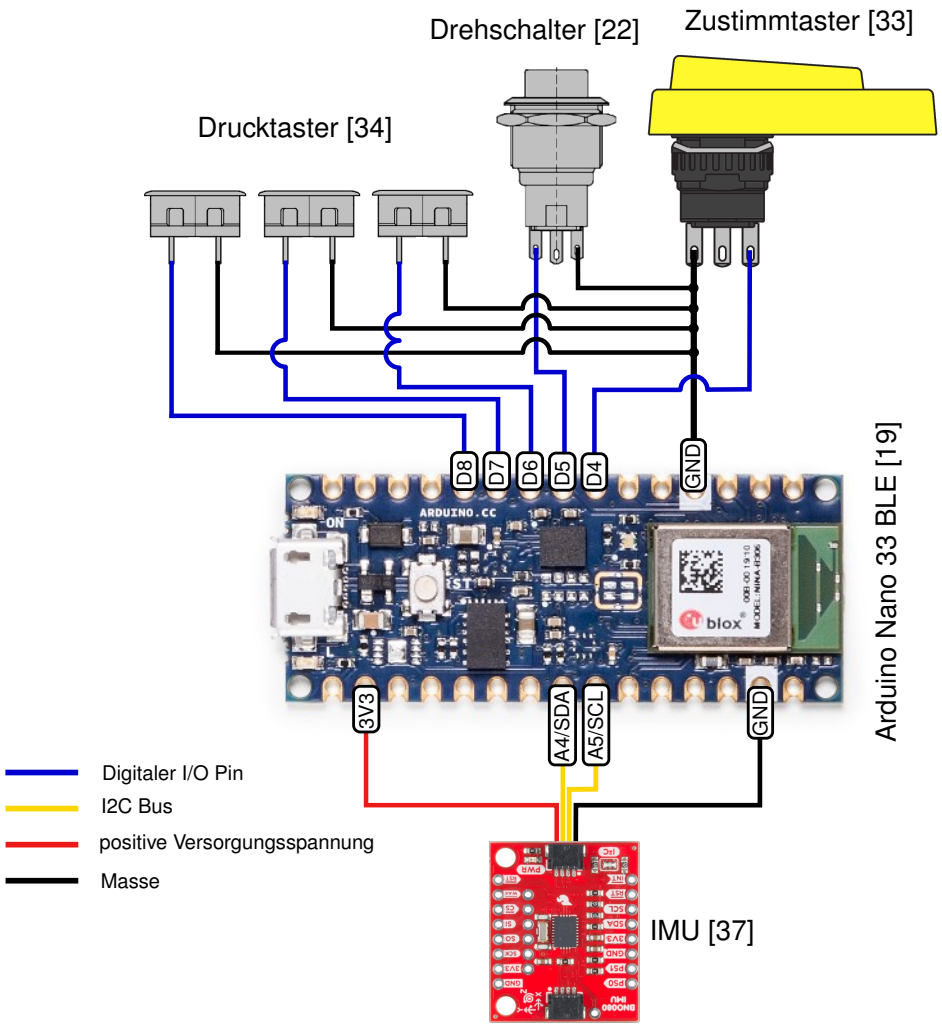


Abbildung 18 Arduino Schaltplan.

Um die Daten vom Arduino an den Laptop weiterzuleiten, wird eine serielle Datenübertragung verwendet, welche über ein USB-Kabel hergestellt wird. Zum Senden der Daten vom Mikrocontroller wird der Befehl „Serial.println()“ benutzt. Dieser sendet immer eine Zeile, die mit einem Carriage-Return (\r) und einem Zeilenvorschubzeichen (\n) endet. Das ist wichtig, damit in der GUI das Ende einer Nachricht detektiert werden kann. In der gesendeten Zeile werden die Zustände und Werte der angeschlossenen Geräte hintereinandergeschrieben und durch ein Semikolon getrennt. Welche Daten übertragen werden, ist in Tabelle 3 dargestellt.

Die Ausgabe der Kardanwinkel und Kalibriergenauigkeit kann bei Bedarf ausgeschaltet werden, um die Geschwindigkeit der seriellen Datenübertragung zu erhöhen. Sind Kardanwinkel und Kalibriergenauigkeit deaktiviert, können die Daten vom seriellen Bus im Durchschnitt alle $1{,}8\,\mathrm{ms}$ ausgelesen werden. Die maximale Abweichung betrug bei dieser Messung $35\,\mathrm{ms}$. Werden die Daten aktiviert, dauert die Übertragung durchschnittlich $3{,}4\,\mathrm{ms}$, mit einer maximalen Abweichung von $28\,\mathrm{ms}$. Für die durchschnittliche Übertragungszeit wurden die Zeiten

vom Empfang einer Nachricht bis zum Empfang der nächsten Nachricht in der GUI über einen Zeitraum von einer Stunde gemessen. Daraus wurde schließlich der Mittelwert berechnet. Die Programmierung des Mikrocontrollers wurde mithilfe der Arduino IDE [20] durchgeführt.

4.2.2. Gehäuse

Ein weiterer Bestandteil dieser Arbeit ist die Konstruktion und der Aufbau des Gehäuses, das auf dem Laptop befestigt werden kann. Darin sind alle Bauteile, die in Abbildung 10 enthalten sind, verbaut.
Im Folgenden sind die wesentlichen Anforderungen an das Design des Gehäuses zusammengefasst:

- Gehäuse vom Laptop abnehmbar,
- Klappfuß des Laptops weiter funktionsfähig,
- abnehmbare Tastatur weiter benutzbar,
- Gewicht möglichst gering,
- ergonomisches Design.

Abbildung 19 und 20 zeigen eine Fotoaufnahme des Programmierhandgeräts. Zusätzlich sind in Anhang B in den Abbildungen 25 und 26 weitere Details des Gerätes dargestellt.

Abbildung 19 Vorderseite des Programmierhandgeräts.

Um das Gehäuse (1) abnehmbar zu gestalten, wird eine Schutzhülle (2) verwendet, auf der zwei Führungsschienen (3) angebracht sind. Über diese Schienen lässt sich das Gehäuse am Laptop befestigen. Weiter rasten am Ende der Führungsschiene zwei Clips (4) ein, wel-

che das Gehäuse in Position halten. Zur Fixierung befindet sich zusätzlich in der Hülle ein Magnet (14). Dieser wird von einem anderen Magnet angezogen, welcher im Laptop verbaut ist. Ursprünglich ist der Magnet im Laptop als Halterung für einen Eingabestift vorgesehen. Beim Aufschieben des Gehäuses wird ein USB-Stecker (13) am Laptop angeschlossen. Die verbaute Hardware wird dadurch verbunden und kann Daten an den Laptop senden. Das Gehäuse (1) selbst besteht aus sechs 3D gedruckten Einzelteilen, welche mit dem „HP Multi Jet Fusion“-Verfahren aus dem Material PA12 gedruckt wurden.
Um die Benutzung des Klappfußes (5) weiter zu ermöglichen, wurde das Handstück (6) am Gehäuse um 15 Grad geneigt. Dies ist in Anhang B in Abbildung 26 dargestellt. Das Bedienmodul kann somit aufrecht abgestellt werden.
Für die Bedienung des Roboters, befindet sich an der Hinterseite des Programmierhandgeräts ein um 45 Grad gedrehtes und um 15 Grad geneigtes Handstück (6). Wird dieses in die linke Hand genommen, kann der Zustimmtaster (7) und ein Druckknopf (8) betätigt werden. Der Druckknopf (8) ist dafür vorgesehen, ein Roboterprogramm zu starten. Mit der rechten Hand können die restlichen Elemente des Programmierhandgeräts bedient werden. Beispielsweise die 3D-Maus (9), welche zur Steuerung des Roboters benutzt wird. Weiter gibt es einen Druckknopf (10) an der Hinterseite des Gehäuses, welcher die Steuerung der Orientierung des TCP über die IMU aktiviert. Auf der Oberseite des Gehäuses befindet sich schließlich ein Drehschalter (11), welcher zwischen dem Manuellen-Modus und dem Automatik-Modus umschaltet und ein frei konfigurierbarer Druckknopf (12).

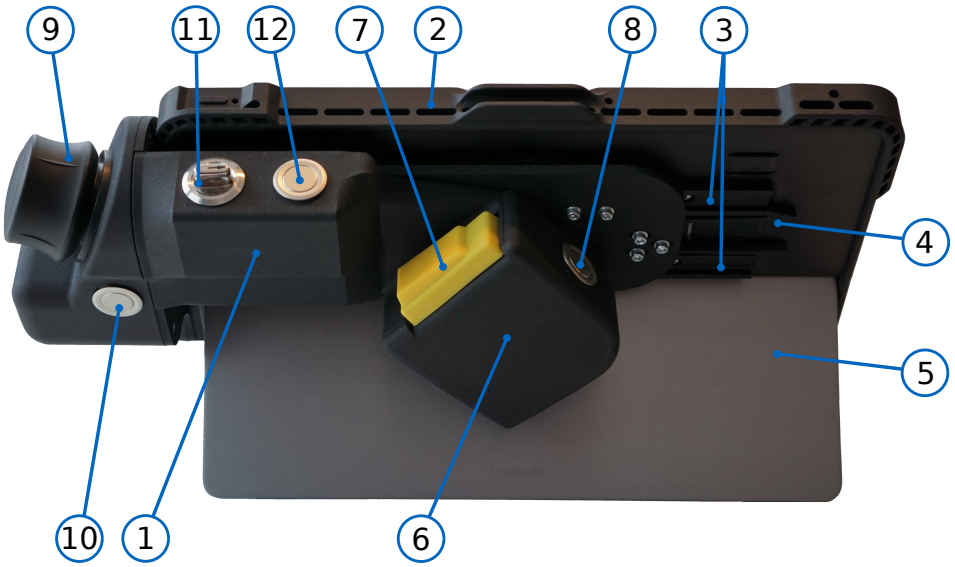


Abbildung 20 Rückseite des Programmierhandgeräts.

5. Ergebnisse

In diesem Kapitel wird auf die Ergebnisse der Arbeit eingegangen. Zuerst wird der gesamte Aufbau der Arbeit dargestellt. Dann wird in Abschnitt 5.1 die Bedienung mit den verschiedenen Eingabegeräten erklärt und in Abschnitt 5.2 werden die Ergebnisse diskutiert.

In Abbildung 21 ist ein Übersichtsbild dargestellt, in dem gezeigt wird, wie der simulierte Industrieroboter mit dem Programmierhandgerät bedient wird. Im unteren Bereich der Abbildung befindet sich der Prototyp des Programmierhandgeräts. Darauf läuft die implementierte GUI, welche Daten mit der Simulation austauscht. Im oberen Bereich des Bildes ist ein Bildschirm dargestellt, welcher an den Simulationscomputer angeschlossen ist. Darauf läuft das Simulationsmodell der Robotersteuerung auf der rechten Seite des Bildschirms und eine Visualisierung eines Industrieroboters auf der linken Seite. Mit dem umgesetzten Programmierhandgerät ist es möglich, den dargestellten Roboter zu steuern und zu programmieren.

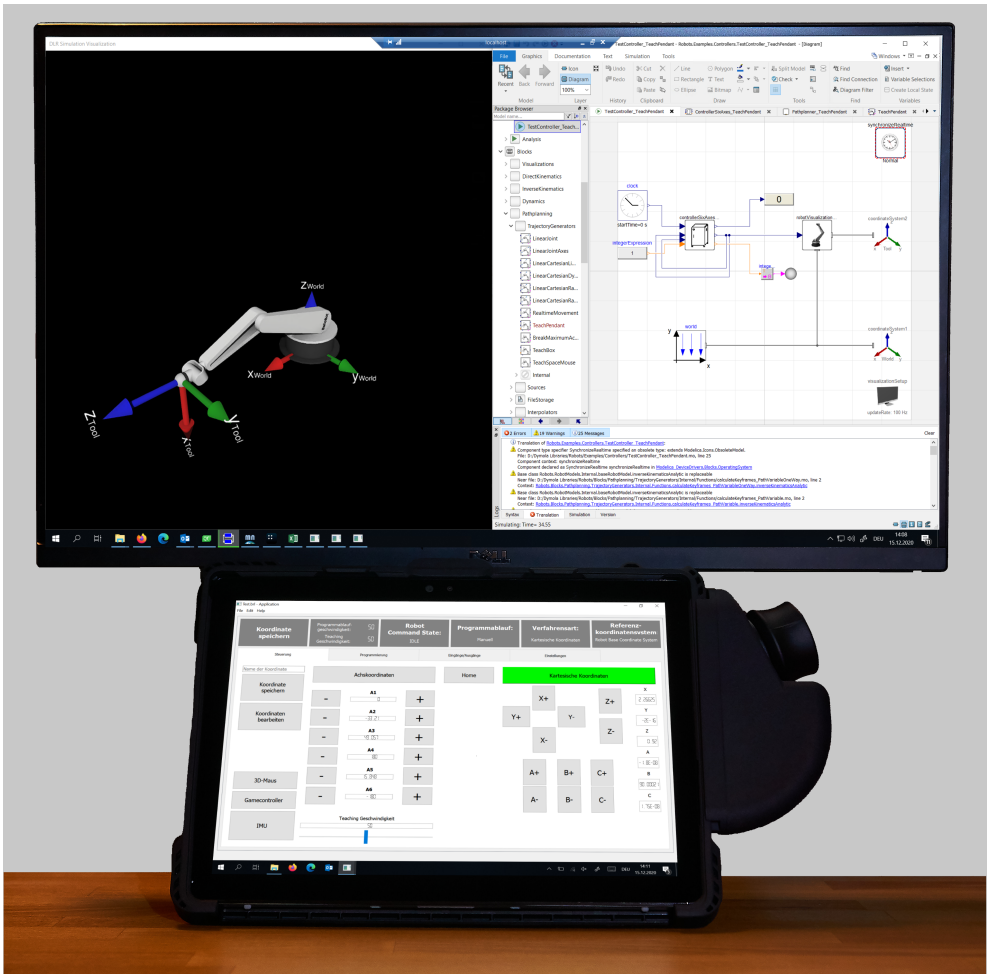


Abbildung 21 Gesamtübersicht der Arbeit.

5.1. Bedienung

Hier wird im Detail erklärt, wie der Roboter mit den verschiedenen Eingabegeräten bedient wird. Mit dem Gamecontroller und der 3D-Maus ist es möglich, den Roboter in Achskoordinaten sowie in kartesischen Koordinaten zu steuern. Zusätzlich kann im kartesischen Modus ausgewählt werden, ob die Bewegungen bezüglich eines Koordinatensystems im Ursprung des Roboters oder einem Koordinatensystem im TCP ausgeführt werden.

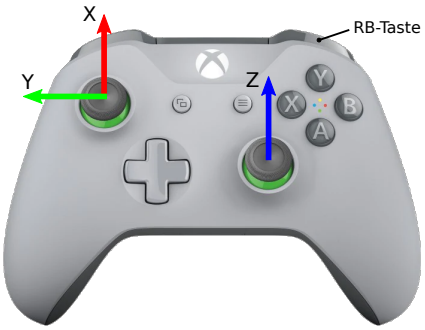


Abbildung 22 Bedienung mit Xbox Controller.

Mit dem linken Analogstick des Xbox Controllers, welcher in Abbildung 22 dargestellt ist, kann der Roboter im kartesischen Modus entlang der x-Achse und der y-Achse bewegt werden. Weiter kann dieser mit dem rechten Analogstick entlang der z-Achse gesteuert werden. Durch gedrückt halten der RB-Taste wird vom translatorischen Modus auf den rotatorischen Modus umgeschaltet. Dabei wird der Roboter, bei gleichen Analogstickeingaben, nicht entlang der Achsen bewegt, sondern der TCP wird um diese Achsen rotiert. Das bedeutet zum Beispiel, bei einer Eingabe mit dem rechten Analogstick dreht sich der TCP um die z-Achse. Bei einer Steuerung des Roboters in Achskoordinaten können die ersten drei Achsen des simulierten Sechsachsroboters bewegt werden, wenn die RB-Taste nicht gedrückt ist. Hier steuert der linke Analogstick die erste (y-Richtung) und die zweite (x-Richtung) Roboterachse. Mit Hilfe des rechten Analogsticks kann Achse drei (z-Richtung) bewegt werden. Durch gedrückt halten der RB-Taste können die Achsen vier bis sechs gesteuert werden. Dabei steuert der linke Analogstick Achse vier (x-Richtung) und Achse fünf (y-Richtung) und der rechte Analogstick Achse sechs (z-Richtung). Zusätzlich besteht die Möglichkeit, durch Drücken der A-Taste die aktuellen Koordinaten des Roboters zu speichern. Die restlichen Tasten sind ohne Funktion.

Die 3D-Maus bietet eine weitere Bedienmöglichkeit, die sechs Freiheitsgrade des Roboters zu steuern. Durch Verschieben der 3D-Maus in Richtung der drei Koordinatenachsen, welche in Abbildung 23 eingezeichnet sind, bewegt sich der Roboter im kartesischen Modus entlang dieser Achsen. Es ist auch möglich, die 3D-Maus um diese Achsen zu drehen. Dabei verän-

dert der TCP des Roboters die Orientierung in der jeweiligen Achse.
Weiter kann der Roboter mit der 3D-Maus in Achskoordinaten gesteuert werden. Wird diese in Richtung, der in Abbildung 23 eingezeichneten y-Achse geschoben, wird die erste Achse bewegt. Durch Verschieben der 3D-Maus in x-Richtung, wird die zweite Achse gesteuert und in z-Richtung die dritte Achse. Die vierte Achse kann durch eine Drehung der 3D-Maus um die y-Achse bewegt werden. Die fünfte und sechste Roboterachse wird durch eine Drehung um die x-Achse und die z-Achse gesteuert.
Zusätzlich ist es in den Einstellungen möglich, die drei Freiheitsgrade der Translation oder die drei Freiheitsgrade der Rotation zu sperren. So können unbeabsichtigte Bewegungen vermieden werden.

Abbildung 23 Bedienung mit 3D-Maus und IMU.

Eine weitere Möglichkeit zur Steuerung des Roboters bietet die IMU. Hier kann nur die Orientierung des TCP verändert werden, translatorische Bewegung können nicht durchgeführt werden. Als Referenzkoordinatensytem wurde bei der Steuerung mit der IMU nur das TCP Koordinatensystem implementiert.
Aktiviert wird die IMU-Steuerung durch einen Druckknopf am Gehäuse oder durch eine Schaltfläche in der GUI. Diese müssen während der Zeit, in der der TCP bewegt werden soll, gedrückt bleiben. Nach Aktivierung der IMU-Steuerung folgt der TCP der Bewegung, die mit dem Programmierhandgerät vorgegeben wird.
In Abbildung 23 ist ein IMU Koordinatensystem eingezeichnet. Wird das Programmierhandgerät um diese Achsen gedreht, bewegt sich auch der TCP um diese Achsen. Dafür wird beim Aktivieren der IMU-Steuerung die Ausgangsorientierung abgespeichert. Diese wird benötigt, um während der Bewegung die Relativorientierung zwischen der aktuellen Position des Programmierhandgeräts und der Ausgangslage zu berechnen und diese auf den TCP des Roboters zu übertragen.

5.2. Diskussion

In diesem Abschnitt werden die Ergebnisse kurz zusammengefasst und diskutiert. Dafür wird im ersten Absatz eine kurze Bewertung der Arbeit in Hinblick auf die Aufgabenstellung gegeben. Im zweiten Absatz wird eine Bewertung und Diskussion des entwickelten Prototyps im Vergleich zum Stand der Technik durchgeführt und im weiteren Verlauf werden mögliche Verbesserungen des Prototypen vorgestellt.

Im Rahmen dieser Arbeit wurde ein Prototyp aufgebaut, in dem die grundlegenden Funktionen eines Programmierhandgeräts implementiert sind.
Dafür wurde als Erstes in Kapitel 2 der Stand der Technik für Programmierhandgeräte und Teach-in-Verfahren ermittelt, wie in der Aufgabenstellung gefordert. Auf Basis davon wurden im nächsten Kapitel Konzepte für das Programmierhandgerät erstellt. Diese beinhalten ein Konzept für die Hardware und ein Konzept für die GUI, welche zum Auslesen der Hardware, zum Bedienen des Roboters und als Kommunikationsschnittstelle zwischen Bedienmodul und Simulationsumgebung dient. Die Überprüfung der Übertragbarkeit der Roboterprogrammierung mit dem Programmierhandgerät von der Simulation auf den realen Roboter konnte im Rahmen dieser Arbeit nicht durchgeführt werden, da die Robotersteuerung noch nicht fertiggestellt ist und deshalb das Testen am realen Roboter nicht möglich ist. Das stellt den einzigen Punkt der Aufgabenstellung dar, welcher nicht umgesetzt wurde. Ein weiterer Punkt der Aufgabenstellung ist der prototypische Aufbau eines Programmierhandgeräts und die Integration in das Bahnplanungsmodul der „DLR Robots library“ inklusive der Definition und Umsetzung der Kommunikationsschnittstelle zwischen Eingabegerät und Simulationsumgebung. Dieser Punkt konnte weitestgehend umgesetzt werden und ist zum größten Teil in Kapitel 4 beschrieben. Schlussendlich sollte noch eine Bewertung und Diskussion des entwickelten Prototyps im Vergleich zum Stand der Technik durchgeführt werden. Dies kann im nächsten Absatz gefunden werden.

Mit dem umgesetzten Prototyp ist es möglich, den Roboter durch vier verschiedene Eingabegeräte zu steuern. Das stellt im Vergleich zum Stand der Technik eine Verbesserung dar, da in modernen Programmierhandgeräten meist nur zwei Eingabegeräte zum Einsatz kommen. Mehrere Eingabegeräte bieten den Vorteil, dass die Anwenderin bzw. der Anwender für bestimmte Anwendungsfälle ein Eingabegerät wählen kann, welches Vorteile mit sich bringt oder welches ihr/ihm am besten liegt. Zusätzlich kann auch eine Kombination von Eingabegeräten benutzt werden. Zum Beispiel stellt meines Erachtens die Kombination aus IMU und 3D-Maus eine gute Möglichkeit dar, einen Roboter zu steuern, wenn dabei die rotatorischen Freiheitsgrade der 3D-Maus gesperrt werden. So können mit der IMU die rotatorischen Freiheitsgrade verändert werden und mit der 3D-Maus die translatorischen. Grund dafür ist, dass die Bedienung nur mit der 3D-Maus bei wenig Übung schwierig sein kann. Das Problem dabei ist, dass bei einer gewollten translatorischen Bewegung unbeabsichtigt die rotatorische

Bewegung ausgelöst werden kann und umgekehrt.
Ein weiterer Vorteil des umgesetzten Programmierhandgeräts ist die Integration in die modular erweiterbare Modellierungs- und Simulationsumgebung Dymola, welche auf der Programmiersprache Modelica aufbaut. Dadurch ist es sehr schnell und einfach möglich, neue Funktionen umzusetzen und zu testen.
Durch die Bauweise des abnehmbaren Gehäuses ist es möglich, den Tablet-Computer weiterhin als solchen zu benutzen. Das bedeutet, dass das Programmierhandgerät durch den Standfuß auf der Rückseite aufrecht auf den Tisch gestellt werden kann und eine Tastatur angedockt werden kann. Dadurch ist das Schreiben von Roboterprogrammen direkt am Programmierhandgerät sehr einfach. Das stellt einen weiteren Vorteil im Vergleich zum Stand der Technik dar, da für aktuelle Programmierhandgeräte am Markt meist eine Halterung benötigt wird, um diese aufrecht abzustellen.
Durch das abnehmbare Gehäuse ist es auch möglich, den Laptop für weitere Aufgaben zu benutzen, bei dem keine externe Hardware benötigt wird, oder um in Zukunft mehrere Gehäuse für verschiedene Anwendungsfälle zu erstellen. So wäre der Prototyp sehr flexibel einsetzbar.
Ein weiterer positiver Punkt ist, dass das Programmierhandgerät kabellos ist und das bis auf das 3D gedruckte Gehäuse nur Komponenten von der Stange benutzt wurden.
Mit einem Gewicht von $2\,\mathrm{kg}$ ist der umgesetzte Prototyp etwas schwerer als die Bedienmodule von Kuka oder Yaskawa, welche ein Gewicht von $1{,}1\,\mathrm{kg}$ aufweisen.

Im weiteren Verlauf dieses Kapitels wird auf mögliche Verbesserungen eingegangen.
Mithilfe eines Codeeditors können im Programmierung-Tab Roboterprogramme erstellt und bearbeitet werden. Diese werden über die Tastatur eingetippt. In Zukunft kann dieser Tab so erweitert werden, dass zum Schreiben des Roboterprogramms keine Tastatur mehr benötigt wird, sondern die Eingabe vollständig über die GUI stattfindet. Dafür sind bereits Schaltflächen vorgesehen, in die vorerst nur das Ausführen und Bearbeiten eines Zeilenbefehls implementiert wurde. Weitere Funktionen wurden im Konzept der GUI präsentiert (siehe Kapitel 3.2).
Das geschriebene Roboterprogramm kann an den simulierten Roboter gesendet und ausgeführt werden. Dabei wird das Programm als Zeichenfolge über ModelNet verschickt. Da ModelNet nicht optimal für den Austausch von langen Strings vorgesehen ist, wäre es besser, eine andere Methode zum Übermitteln des Roboterprogramms zu implementieren. Dies könnte potenziell als einer der nächsten Schritte umgesetzt werden. Im Zuge der Entwicklung eines Prototypen wurde einfachheitshalber ModelNet benutzt, da es zum Testen der Funktionsweise ausreicht und auch zum Senden der restlichen Daten verwendet wird.
Im Einstellungen-Tab können derzeit die Eingänge und Ausgänge benannt und deren Anzahl eingestellt werden. Weiter ist es möglich, den Status der Hardware im Gehäuse zu überprüfen und die Translation und Rotation der 3D-Maus zu aktivieren oder zu deaktivieren. In der Weiterentwicklung sollten hier weitere Einstellmöglichkeiten hinzugefügt werden, wie beispielsweise die Einstellung von Achsenlimits oder die Konfiguration des TCP.

Um die Hardware mit dem Tablet-Computer zu verbinden, wurde ein Gehäuse konstruiert und aufgebaut. Dabei wurde zur Verbindung der Hardware mit dem Laptop ein Arduino gewählt. Dieser stellt eine sehr einfache und schnell umsetzbare Methode dar, um die Hardware einzulesen. Der Arduino kommuniziert mit dem Tablet-Computer über einen Serial-Port. Es ist anzumerken, dass der Datenaustausch mit serieller Kommunikation nicht die beste Lösung für ein Programmierhandgerät darstellt, da dabei die Daten asynchron übertragen werden. Eine bessere Möglichkeit zur Datenübertragung zwischen dem Arduino und dem Tablet-Computer stellt ein I²C oder SPI Bus dar. Dafür gibt es einen I²C/SPI zu Ethernet Adapter. Zusätzlich müsste dann auch auf dem Tablet-Computer ein echtzeitfähiges Betriebssystem laufen, welches die Daten mit einem echtzeitfähigen Thread ausließt und an die Robotersteuerung weiterleitet. Da diese Umsetzung den Umfang der Arbeit überschreitet und zur Demonstration der Funktionsweise des Programmierhandgeräts eine serielle Datenübertragung ausreicht, wird diese hier genutzt.
Ein Problem stellt die gewählte Datenübertragung zum Beispiel für den Datenaustausch des Zustimmtasters dar. Da dieser ein sicherheitsrelevantes Bauteil ist, muss garantiert werden, dass die Daten zyklisch an der Robotersteuerung ankommen. Das gleiche Problem besteht für den Not-Halt-Schalter, weshalb auf diesen im Gehäuse verzichtet wurde. Es wurde aber an der Oberseite des Programmierhandgeräts anstelle des Drucktasters (12) neben dem Drehschalter (11) bereits ein Platz für den späteren Einbau vorgesehen (siehe Abbildung 20).

6. Zusammenfassung und Ausblick

Gegenstand der vorliegenden Arbeit ist der Aufbau des Prototypen eines Programmierhandgeräts, mit dem Ziel einen simulierten Industrieroboter zu bedienen.
Zu Beginn der Arbeit wird in Kapitel 2 mithilfe einer Literaturrecherche ein kurzer Überblick über den Stand der Technik gegeben. Auf Basis dieser wurden im dritten Kapitel Konzepte zur Implementierung der Software und Hardware für den Prototypen erstellt. Im Anschluss wurde in Kapitel 4 die Umsetzung der Implementierung vorgestellt. Diese lässt sich im Wesentlichen in drei Punkte gliedern:

- Die Implementierung einer GUI, welche es ermöglicht, den Roboter zu steuern und zu programmieren.
- Die Konstruktion und der Aufbau eines Gehäuses, in dem die nötige Hardware verbaut ist, um einem Tablet-Computer die Funktionen eines Programmierhandgeräts zu geben.
- Die Integration des umgesetzten Programmierhandgeräts in das Bahnplanungsmodul der „DLR Robots library“.

Im Folgenden werden die Punkte detaillierter beschrieben.
Die GUI, welche in Anhang A abgebildet ist, wurde mithilfe des Qt-Frameworks erstellt. Aus Gründen der Übersichtlichkeit wurde die GUI in vier Tabs aufgeteilt, welche jeweils zum Ausführen einer speziellen Aufgabe vorgesehen sind. Mit dem ersten Tab ist es möglich, den Roboter zu steuern. Das bedeutet, ihn mit verschiedenen Eingabegeräten an eine gewünschte Position zu bewegen. Es gibt die folgenden vier verschiedenen Möglichkeiten den Roboter zu steuern: Eingabetasten für jede Koordinatenrichtung im Steuerungs-Tab der GUI, eine 3D-Maus, einen Xbox Controller und eine IMU. Mit dem nächsten Tab können Roboterprogramme erstellt und bearbeitet werden. Weiter gibt es noch einen Tab, mit dem die Eingänge und Ausgänge des Roboters angezeigt werden bzw. gesteuert werden können. Im letzten Tab können verschiedene Einstellungen vorgenommen werden, wie beispielsweise die Namen der Ein- und Ausgänge zu verändern. Um den Status des Roboters immer im Blick zu haben, wurde am oberen Rand der GUI eine Statusleiste implementiert, welche immer sichtbar ist und die wichtigsten Informationen des Roboters enthält. Zusätzlich können mithilfe der implementierten Benutzeroberfläche die Daten der verschiedenen Eingabegeräte eingelesen und an die Simulation des Roboters weitergeleitet werden.
Weiter wird die Konstruktion des Gehäuses näher beschrieben. Über Schienen kann das 3D gedruckte Gehäuse am Laptop angebracht und entfernt werden. Im Gehäuse wurden mehrere Drucktaster, ein Drehschalter und ein Zustimmtaster verbaut. Die Taster und der Drehschalter sind gemeinsam mit einer IMU an einen Arduino angeschlossen. Dieser ist mit dem Laptop verbunden und sendet die Zustände und Daten der angeschlossenen Geräte an die GUI. Es wurde außerdem eine 3D-Maus in dem Gehäuse verbaut.
Zur Simulation der Robotersteuerung wurde das Bahnplanungsmodul der „DLR Robots libra-

ry“ so erweitert, dass es die Daten aus der GUI empfängt und die Kommandos umsetzt.
In Kapitel 5 wird die Bedienung des Roboters mit den Eingabegeräten erklärt und die Ergebnisse der Arbeit diskutiert.
Die Aufgabenstellung konnte bis auf eine Ausnahme vollständig umgesetzt werden. Bei der Ausnahme handelt es sich um die Überprüfung der Übertragbarkeit der Roboterprogrammierung mit dem Programmierhandgerät von der Simulation auf den realen Roboter. Dies konnte nicht durchgeführt werden, da die Robotersteuerung noch nicht einsatzbereit ist und somit kein realer Roboter zum Testen zur Verfügung stand.

Die Möglichkeit der Steuerung des Roboters mit vier verschiedenen Eingabegeräten bietet einen Vorteil gegenüber dem Stand der Technik, da dabei meist nur zwei Eingabegeräte zum Einsatz kommen. Der Vorteil dabei ist, dass dadurch ein bevorzugtes Eingabegerät benutzt werden kann und sich auch eine Kombination von verschiedenen Eingabegeräten vorteilhaft auswirken kann.
Einen weiteren Vorteil bietet die Integration in die „DLR Robots library“. Dadurch lassen sich neue Funktionen sehr schnell umsetzten und testen.
Durch die Bauweise des Gehäuses kann der Tablet-Computer sehr flexibel eingesetzt werden und auch wie ein normaler Laptop bedient werden, was für das Erstellen von Roboterprogrammen über die Tastatur Vorteile bringt.
Weiter ist der umgesetzte Prototyp kabellos und die Hardwarekomponenten sind bis auf das 3D gedruckte Gehäuse ausschließlich von der Stange. Das Gewicht des Programmierhandgeräts ist etwas höher als bei aktuellen Geräten auf dem Markt.

Der aufgebaute Prototyp kann als Ausgangsplattform zur Entwicklung eines Programmierhandgeräts für den Einsatz am realen Roboter genutzt werden. Außerdem ist es möglich, das umgesetzte Bedienmodul in Zukunft bei der Entwicklung der Robotersteuerung zu benutzen, um verschiedene Funktionen zu testen.
Als nächster Schritt wäre es sinnvoll, eine Studie durchzuführen, in der Probanden mehrere Aufgaben mithilfe der verschiedenen Eingabegeräte lösen müssen. Durch die Ergebnisse der Studie können Vor- und Nachteile der entsprechenden Eingabegeräte analysiert werden. Diese stellen eine aussagekräftige Grundlage zur Bewertung der verschiedenen Eingabegeräte dar.

Literatur

[1] Sushilkumar Ambhore. “A Comprehensive Study on Robot Learning from Demonstration”. In: *2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA 2020)*. Piscataway, NJ: IEEE, 2020, S. 291–299. ISBN: 978-1-7281-4167-1. DOI: 10.1109/ICIMIA48430.2020.9074946.

[2] Tobias Bellmann, Andreas Seefried und Bernhard Thiele. “The DLR Robots library – Using replaceable packages to simulate various serial robots”. In: *Proceedings of Asian Modelica Conference 2020, Tokyo, Japan, October 08-09, 2020*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2020, S. 153–161. DOI: 10.3384/ecp2020174153.

[3] Riccardo Caccavale, Matteo Saveriano, Alberto Finzi und Dongheui Lee. “Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction”. In: *Autonomous Robots* 6 (2019), S. 1291–1307. ISSN: 0929-5593. DOI: 10.1007/s10514-018-9706-9.

[4] Davide Colombo, D. Dellafrate und Lorenzo Molinari Tosatti. “PC based control systems for compliance control and intuitive programming of industrial robots”. In: *Proceedings of the Joint Conference on Robotics*. VDI-Berichte. Düsseldorf: VDI-Wissensforum-IWB-GmbH, 2006. ISBN: 3180919566.

[5] Wilson F. Cueva C., S. Hugo M. Torres und M. John Kern. “7 DOF industrial robot controlled by hand gestures using microsoft kinect v2”. In: *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*. Hrsg. von Diego Patiño und Eugenio Yime. Piscataway, NJ: IEEE, 2017, S. 1–6. ISBN: 978-1-5386-0398-7. DOI: 10.1109/CCAC.2017.8276455.

[6] Staffan Ekvall und Danica Kragic. “Robot Learning from Demonstration: A Task-level Planning Approach”. In: *International Journal of Advanced Robotic Systems* 5.3 (2008), S. 33. ISSN: 1729-8814. DOI: 10.5772/5611.

[7] Hilding Elmqvist, Sven Erik Mattsson und Martin Otter. “Modelica - a language for physical system modeling, visualization and interaction”. In: *Computer-Aided Design and Test for High-Speed Electronics 1999*. Hrsg. von IEEE. Piscataway: IEEE, Sept. 1999, S. 630–639. ISBN: 0-7803-5500-8. DOI: 10.1109/CACSD.1999.808720.

[8] Björn Hein, Martin Hensel und Heinz Worn. “Intuitive and model-based on-line programming of industrial robots: A modular on-line programming environment”. In: *2008 IEEE International Conference on Robotics and Automation*. 2008, S. 3952–3957.

[9] Lin Hsien-I und Lin Yu-Hsiang. “A novel teaching system for industrial robots”. In: *Sensors (Basel, Switzerland)* 14.4 (2014), S. 6012–6031. DOI: 10.3390/s140406012.

[11] Pedro Neto, Dario Pereira, Norberto Pires und Paulo Moreira. “Real-time and continuous hand gesture spotting: An approach based on artificial neural networks”. In: *IEEE International Conference on Robotics and Automation (ICRA), 2013*. Piscataway, NJ: IEEE, 2013, S. 178–183. ISBN: 978-1-4673-5643-5. DOI: 10.1109/ICRA.2013.6630573.

[10] Pedro Neto, Norberto Pires und Paulo Moreira. “High–level programming and control for industrial robotics: using a hand–held accelerometer–based input device for gesture and posture recognition”. In: *Industrial Robot: An International Journal* 37.2 (2010), S. 137–147. ISSN: 0143-991X.

[12] Andreas Seefried, Alexander Pollok, Richard Kuchar, Matthias Hellerer, Martin Leitner, Daniel Milz, Christian Schallert, Thiemo Kier, Gertjan Looye und Tobias Bellmann. “Multi-domain Flight Simulation with the DLR Robotic Motion Simulator”. In: *Annual Simulation Symposium (ANSS 2019)*. Society for Modeling and Simulation International (SCS), 2019. ISBN: 9781510892156. DOI: 10.22360/SpringSim.2019.ANSS.011.

[13] Franz Steinmetz, Verena Nitsch und Freek Stulp. “Intuitive Task-Level Programming by Demonstration Through Semantic Skill Recognition: IEEE Robotics and Automation Letters, 4(4), 3742-3749”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), S. 3742–3749. DOI: 10.1109/LRA.2019.2928782.

[14] Franz Steinmetz, Annika Wollschlager und Roman Weitschat. “RAZER—A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), S. 1362–1369. DOI: 10.1109/LRA.2018.2798300.

[15] Gilbert Tang und Phil Webb. “The Design and Evaluation of an Ergonomic Contactless Gesture Control System for Industrial Robots”. In: *Journal of Robotics* 2018 (2018), S. 1–10. ISSN: 1687-9600. DOI: 10.1155/2018/9791286.

[16] Maximilian Wagner, Dennis Avdic und Peter Heß. “Gamepad Control for Industrial Robots - New Ideas for the Improvement of Existing Control Devices”. In: *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics - (Volume 1)*. SciTePress, 2016, S. 368–373. ISBN: 978-989-758-198-4. DOI: 10.5220/0005982703680373.

[17] Wolfgang Weber. *Industrieroboter: Methoden der Steuerung und Regelung*. 2., neu bearbeitete Auflage. München: Hanser Verlag, 2019. ISBN: 9783446419964. DOI: 10.3139/9783446419964. URL: http://www.hanser-elibrary.com/isbn/9783446410312.

[18] Shaolin Zhang, Shuo Wang, Fengshui Jing und Min Tan. “A Sensorless Hand Guiding Scheme Based on Model Identification and Control for Industrial Robot”. In: *IEEE Transactions on Industrial Informatics* 15.9 (2019), S. 5204–5213. ISSN: 1551-3203.

Internetquellen

[19] Arduino, Hrsg. *Arduino Nano 33 BLE*. URL: https://store.arduino.cc/arduino-nano-33-ble (besucht am 02. 11. 2020).

[20] Arduino, Hrsg. *Arduino Software*. URL: https://www.arduino.cc/en/software (besucht am 19. 11. 2020).

[21] CEVA, Inc., Hrsg. *BNO080/BNO085 Datenblatt*. 2020. URL: https://www.ceva-dsp.com/wp-content/uploads/2019/10/BNO080_085-Datasheet.pdf (besucht am 18. 11. 2020).

[22] Conrad Electronic SE, Hrsg. *TRU COMPONENTS TC-GQ22-A-11X/21/B/12V/S203 Drehschalter 250 V 3 A Schaltpositionen 2 1 x 90 ° IP40 1 St.IP40*. URL: https://www.conrad.de/de/p/tru-components-tc-gq22-a-11x-21-b-12v-s203-drehschalter-250-v-3-a-schaltpositionen-2-1-x-90-ip40-1-st-1602507.html (besucht am 02. 11. 2020).

[23] Dell GmbH, Hrsg. *Latitude 7200 2-in-1 Laptop*. URL: https://www.dell.com/de-de/work/shop/dell-notebooks/latitude-7200-2-in-1-laptop/spd/latitude-12-7200-2-in-1-laptop#features_section (besucht am 02. 11. 2020).

[24] Franka Emika GmbH, Hrsg. *Capability*. 2018. URL: https://www.franka.de/capability (besucht am 10. 11. 2020).

[25] International Federation of Robotics, Hrsg. *IFR presents World Robotics Report 2020*. Frankfurt, 2020. URL: https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe (besucht am 07. 12. 2020).

[26] Kuka AG, Hrsg. *KUKA ready2_pilot*. URL: https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/ready2_use/kuka-ready2_pilot (besucht am 15. 10. 2020).

[27] Kuka AG, Hrsg. *Kuka smartPAD 2 Broschüre*. 2018. URL: https://www.kuka.com/de-at/produkte-leistungen/robotersysteme/robotersteuerungen/smartpad (besucht am 02. 11. 2020).

[28] MEGATRON Elektronik GmbH & Co. KG, Hrsg. *Fingerjoystick Serie SpaceMouse Module*. URL: https://www.megatron.de/produkte/fingerjoysticks/fingerjoystick-serie-spacemouser-module.html (besucht am 02. 11. 2020).

[29] Microsoft Ireland Operations Limited, Hrsg. *Xbox Wireless Controller*. URL: https://www.microsoft.com/de-de/p/xbox-wireless-controller-grau-gr-n/8rvfzqnvwt2c?activetab=pivot%3aoverviewtab (besucht am 01. 11. 2020).

[30] Nathan Seidle. *SparkFun BNO080 Arduino Library*. Hrsg. von SparkFun. 1.12.2020. URL: https://github.com/sparkfun/SparkFun_BNO080_Arduino_Library (besucht am 16. 12. 2020).

[31] Carlos Pérez Ramil. *thumbstick-deadzones*. 2017. URL: https://github.com/Minimuino/thumbstick-deadzones (besucht am 28. 11. 2020).

[32] Riverbank Computing, Hrsg. *QScintilla Download*. URL: https://riverbankcomputing.com/software/qscintilla/download (besucht am 25. 11. 2020).

[33] RS Components GmbH, Hrsg. *Idec Taster Rastend Wechselschalter, 2-polig, Ø 16.2mm*. URL: https://de.rs-online.com/web/p/drucktaster-schalter/8077102/ (besucht am 02. 11. 2020).

[34] RS Components GmbH, Hrsg. *Schurter Druckschalter Tastend EIN-AUS Schalter, 1-polig, Ø 19mm*. URL: https://de.rs-online.com/web/p/drucktaster-schalter/4412895 (besucht am 02. 11. 2020).

[35] Siemens AG, Hrsg. *Ansteuerung eines KUKA Industrieroboters über eine SIMATIC S7-1500: SIMATIC S7-1500/ TIA Portal V13 SP1/ KUKA.PLC mxAutomation: Anwendungsbeispiel 01/2016*. 2016. URL: https://docplayer.org/54585216-Ansteuerung-eines-kuka-industrieroboters-ueber-eine-simatic-s7-1500.html (besucht am 02. 11. 2020).

[36] *Simple DirectMedia Layer - Homepage*. URL: https://www.libsdl.org/ (besucht am 06. 11. 2020).

[37] SparkFun Electronics, Hrsg. *SparkFun VR IMU Breakout - BNO080 (Qwiic)*. URL: https://www.sparkfun.com/products/14686 (besucht am 02. 11. 2020).

[38] Alex Spataru. *QJoysticks: Introduction*. 14.05.2018. URL: https://frc-utilities.github.io/documentation/qjoysticks/index.html (besucht am 20. 11. 2020).

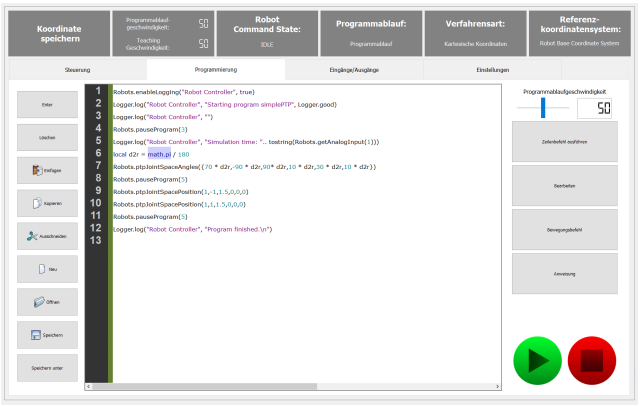
[39] The Qt Company Ltd., Hrsg. *Qt 5.15*. 2020. URL: https://doc.qt.io/qt-5/index.html (besucht am 29. 11. 2020).

[40] Wandelbots GmbH, Hrsg. *TracePen*. URL: https://wandelbots.com/tracepen/ (besucht am 15. 10. 2020).

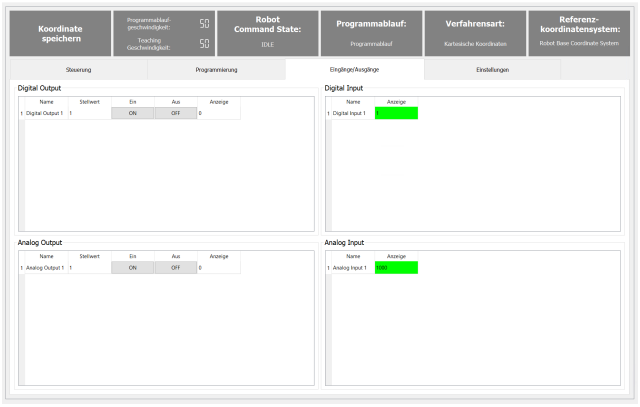
[41] Yaskawa America, Inc., Hrsg. *Smart Pendant Datenblatt*. 2019. URL: https://www.motoman.com/getmedia/52e61c70-b36a-46ae-a3a0-7014d412a9f7/SmartPendant.pdf.aspx?_ga=2.6486967.425154820.1609618105-1363967682.1607449004 (besucht am 02. 11. 2020).

A. GUI

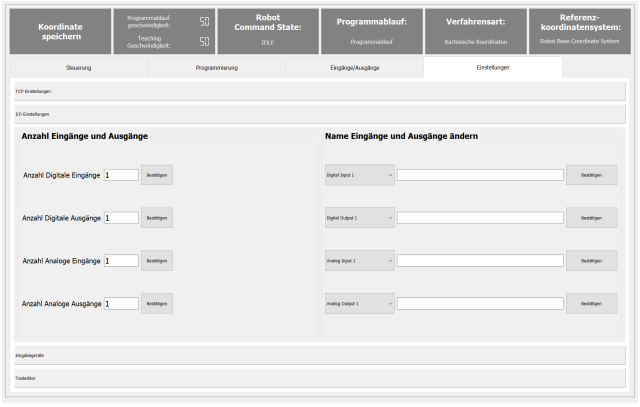
(a) Steuerung-Tab.



(b) Programmierung-Tab.



(c) Eingänge/Ausgänge-Tab.



(d) Einstellungen-Tab.

Abbildung 24 Tabs der GUI.

B. Gehäuse

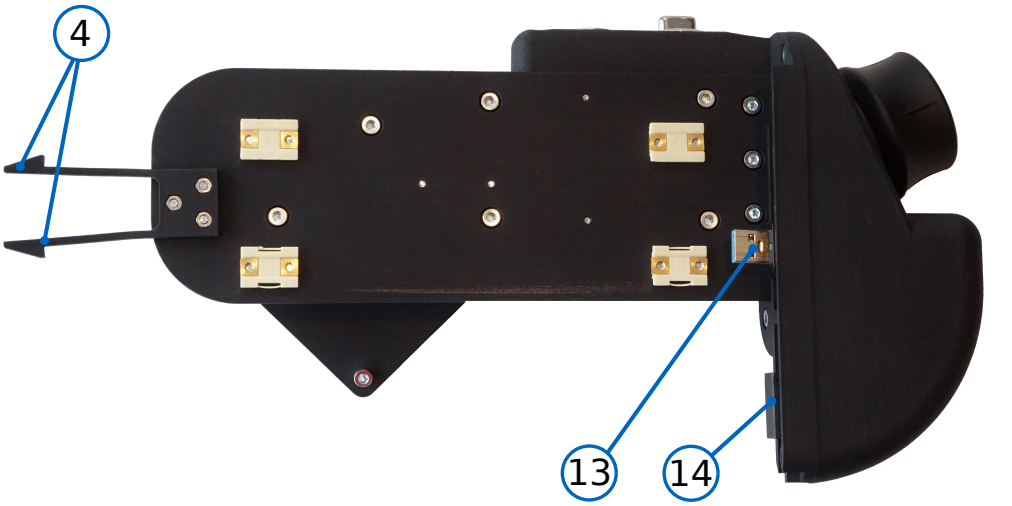


Abbildung 25 Gehäuse des Programmierhandgeräts.

Abbildung 26 Programmierhandgerät von der Seite.

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum, Unterschrift