



ENTWICKLUNG EINER GESTENERKENNUNG MITHILFE VON STEREOKAMERAS MIT ROS

BACHELORARBEIT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
BACHELOR OF ENGINEERING (B. ENG.)

Oliver Bosin

Betreuer:
Prof. Dr. Ferdinand Englberger

Tag der Abgabe: TT.MM.JJJJ

Universität der Bundeswehr München
Fakultät für Elektrotechnik und Technische Informatik
Institut 4

Neubiberg, Juni 2019

Erklärung

gemäß Beschluss des Prüfungsausschusses für die Fachhochschulstudiengänge der UniB-wM vom 25.03.2010

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen.

Neubiberg, den TAG MONAT JAHR

Oliver Bosin

Erklärung

gemäß Beschluss des Prüfungsausschusses für die Fachhochschulstudiengänge der UniB-wM vom 25.03.2010

Der Speicherung meiner Masterarbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere, dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

Neubiberg, den TAG MONAT JAHR

Oliver Bosin

Abstrakt

Dieses Dokument dient zum Einen dazu, eine einheitliche Vorlage für alle Abschlussarbeiten im Institut 4 bereitzustellen. Zum Anderen wird es den Studenten hiermit vereinfacht eine umfassende Abschlussarbeit anzufertigen, die gewissen Grundregeln für wissenschaftliche Arbeiten entspricht. Diese Vorlage ist nicht bindend. Es wird allerdings empfohlen, sich hieran zu halten. Auf diese Weise kann man sich auf den Inhalt und das Schreiben konzentrieren, ohne sich große Gedanken über das Layout und die technische Umsetzung Gedanken machen zu müssen.

Das erste Kapitel einer Arbeit ist das Abstrakt. Auch wenn es chronologisch an erster Stelle steht, so wird es doch eigentlich zeitlich ganz am Ende geschrieben. Denn hier wird dem Leser ein umfassender Überblick über die Arbeit ermöglicht, ohne diese komplett lesen zu müssen. Alle wichtigen Punkte der Ausarbeitung müssen dabei aufgeführt werden. Beginnend mit einer Einleitung, den Grundlagen (bzw. Methoden), die Ergebnisse und Ausblick bzw. Diskussion. Die Kunst hierbei ist, dass sich **kurz** zu fassen. Ein Abstrakt sollte maximal zwei Seiten umfassen.

Inhaltsverzeichnis

Tabellenverzeichnis	V
Abbildungsverzeichnis	VII
Listings	IX
Abkürzungsverzeichnis	XI
1 Einleitung	1
1.1 Motivation	1
1.2 Aufgabenstellung	1
1.3 Gliederung	1
2 Systemaufbau	3
2.1 Software	3
2.1.1 Betriebssysteme	3
2.1.2 Integrierte Entwicklungsumgebungen	3
2.1.3 Robot Operating System	4
2.1.4 OpenNI	4
2.1.5 SensorKinect	5
2.1.6 Ros-Kinetic-OpenNI	5
2.1.7 NITE	5
2.1.8 OpenNI Tracker	5
2.1.9 MoveIt!	5
2.1.10 Turtle Simulator	5
2.1.11 Softwarekonzept	6
2.2 Hardware	8
2.2.1 Rechner	8
2.2.2 Kinect	8
2.2.3 Roboterarm "rob_arm_small"	8
2.2.4 Mikrocontrollerboard	9
2.3 Systemaufbau	10
3 Grundlagen	11
3.1 Stereokamera	11
3.1.1 Typen	11

Inhaltsverzeichnis

3.2	OpenNI	13
3.2.1	Was ist OpenNI	13
3.2.2	Module	13
3.2.3	Kompatibilität und Verfügbarkeit	14
3.3	ROS	14
3.3.1	Was ist ROS	14
3.3.2	Konzept und Komponenten	15
3.3.3	Wichtige ROS-Befehle	17
3.4	MoveIt!	17
3.4.1	Architektur	18
3.4.2	Das move_group_interface	19
4	Implementation des Gestenerkennungssystems mit ROS	23
4.1	Vorbereitung	23
4.1.1	Catkin Workspace erstellen	23
4.1.2	Abhängigkeiten	23
4.2	Installation der Komponenten	24
4.2.1	OpenNI installieren	24
4.2.2	Ros-Kinetic-OpenNI installieren	25
4.2.3	NITE installieren	25
4.2.4	OpenNI Tracker installieren	25
4.3	Start und Test des Gestenerkennungssystems	25
4.3.1	Starten der Komponenten	25
4.3.2	Mit Rviz visualisieren und überprüfen	25
5	Implementation von Gestensteuerungen basierend auf dem Gestenerkennungssystem	27
5.1	Vorbereitung	27
5.1.1	Installation MoveIt!	27
5.1.2	Einrichtung der Pakete rob_arm_small und rob_arm_small_hw_interface	27
5.1.3	Erstellung eines ROS Paketes	27
5.2	turtlesim_gesture_control	27
5.3	roboticarm_gesture_control	27
5.4	Zitate	27
5.5	Bilder	28
5.6	Auflösung	29
5.7	Farben	30
5.8	Quelltext	30
5.9	Diagramme	32
5.10	Layout	32
5.11	Sprachliches	32
5.12	Abkürzungen	32
5.13	Tabellen	33

5.14 Formeln	33
6 Diskussion	35
6.1 Ausblick	35
7 Anhang	37
7.1 Schaltpläne	38
7.2 Projektplan mit themenbezogenem Zeitaufwand	41
Literaturverzeichnis	43
Index	45

Tabellenverzeichnis

2.1	Winkelwerte der Gelenke[27]	9
2.2	Griffweite des Greifers[27]	9
3.1	Wichtige Befehle in ROS[3, 16]	17
5.1	Diagrammarten der System Modelling Language	33

Abbildungsverzeichnis

2.1	Softwarearchitektur	6
2.2	Abstract Layered View - OpenNI	7
2.3	Softwarekonzept	7
2.4	Kinect	8
2.5	Freiheitsgrade des Roboterarms[27]	9
2.6	Aufbau Gesamtsystem	10
3.1	Kommunikationsaufbau von Knoten	16
3.2	Architektur MoveIt!	18
4.1	Ordnerstruktur Catkin-Workspace	24
5.1	Autonomes Kettenfahrzeug des Instituts 4	29
5.2	Fahrzeuge des Instituts 4.	29
5.3	Einstellung der Größe für eine Matlab Abbildung.	30
5.4	Einstellung der Abbildungs-Auflösung beim Exportieren aus Matlab heraus.	30
7.1	Schaltungsdesign für die Spannungsversorgungsplatine	39
7.2	Layout für die Spannungsversorgungsplatine.	40
7.3	Themenbezogene Stundenaufschlüsselung für die Abschlussarbeit.	41
7.4	Aufgabenübersicht für die Abschlussarbeit.	42

Listings

3.1	Planen auf ein Pose Goal	20
3.2	Planen auf ein Joint Goal	20
3.3	Planen auf ein Position Target	21
3.4	Plan ausführen	21
5.1	Wichtige Funktion für das Programm	31
5.2	Sourcecode SN74x6541	31

Abkürzungsverzeichnis

API Application Programming Interface

RGB Rot Grün Blau

NI Natural Interaction

XML Extensible Markup Language

1 Einleitung

1.1 Motivation

1.2 Aufgabenstellung

Die konkrete Aufgabestellung kann in einem Unterkapitel verdeutlicht werden. Während die Einleitung mehr eine allgemeine Hinleitung zum Thema ist, muss hier dann erläutert werden, welche Probleme bei bestehenden Systemen bestanden, welche Einschränkungen existieren oder welche Weiterentwicklungen gewünscht sind.

Ein Beispiel wäre, wenn ein Roboterfahrzeug des Instituts 4 unter Belastung ein Fehlverhalten zeigt. Daraus ergibt sich die Aufgabe zu ermitteln, woher dieses Fehlverhalten kommt und wie es behoben werden kann.

1.3 Gliederung

2 Systemaufbau

Der Aufbau, des gesamten Systems, lässt sich in den Hardwareanteil und den Softwareanteil aufgliedern. Zuerst wird auf die verschiedenen Pakete und Treiber, die den Softwareanteil ausmachen, eingegangen und deren Zusammenspiel im Gesamtsoftwarekonzept erläutert. Im zweiten Teil des Kapitels werden die Hardwarekomponenten des Systems vorgestellt. Hier wird als erstes die Aufgabe jeder relevanten Komponente beschrieben, um diese dann verständlich in einen Kontext bringen zu können. Am Ende dieses Kapitels werden der Softwareanteil und der Hardwareanteil, im Gesamtkonzept, zu einer verständlichen Darstellung des Systems zusammengefügt.

2.1 Software

In diesem Abschnitt wird die in dem Entwicklungsprozess verwendete und eingebundene Software vorgestellt. Dies soll einen ersten Überblick über den Entwicklungsprozess geben und das Softwarekonzept des Systems verständlich darstellen. Die Installation und die genauere Verwendung, der hier genannten Software, ist integraler Teil des vierten Kapitels und wird somit hier nicht behandelt.

2.1.1 Betriebssysteme

Auf dem zur Entwicklung genutzten Rechner ist WINDOWS 10 als Betriebssystem installiert. Da ROS ein Framework für Linuxdistributionen ist, ist es notwendig ein Linux-Betriebssystem in einer virtuellen Maschine zu betreiben. Hierfür wird eine virtuelle Maschine von "VM-Ware" verwendet. In der virtuellen Maschine wird ein Ubuntu in der Version 16.04 LTS, welches in der WE4 laufend gepflegt und erweitert wird, betrieben. Unter dieser Ubuntu Version der WE4 ist das ROS-Framework, in der Version ""Kinetic Kame", bereits installiert. Bei der virtuellen Maschine von "VM-Ware" ist darauf zu achten, dass der USB-Kompatibilitätsmodus auf "USB 3.0" gesetzt ist, da sonst Probleme mit USB-Geräten auftreten können.

2.1.2 Integrierte Entwicklungsumgebungen

Für die Entwicklung von Software reicht in der Regel ein Texteditor und ein Compiler aus. Um diese Entwicklung komfortabler und übersichtlicher zu gestalten werden integrierte Entwicklungsumgebungen verwendet. Hier sind normalerweise Texteditor, Compiler, Linker und weitere Programme in einer Anwendung integriert, um ein ständiges wechseln zwischen den Programmen zu vermeiden.

Eclipse

Eclipse Um Programme in der Programmiersprache C/C++ zu entwickeln, wurde die integrierte Entwicklungsumgebung "Eclipse"[2] verwendet. Die verwendete Installation von "Eclipse" wurde innerhalb der WE4 mit dem "Ubuntu" Betriebssystem zusammen gepflegt. Weiterhin waren keine Anpassungen, der Konfiguration von "Eclipse", nötig um Programme für das ROS-Framework darin zu entwickeln.

Keil

Keil Die integrierte Entwicklungsumgebung " μ -vision" von "KEIL" wurde verwendet, um ggf. Änderungen an der Software, die auf dem Mikrocontrollerboard läuft, vorzunehmen.[7] Auf das genannte Mikrocontrollerboard wird im Hardwareteil des Kapitels eingegangen.

2.1.3 Robot Operating System

Das "Robot Operating System" ist ein Quelloffenes und flexibles Framework, das darauf abzielt die Entwicklung von Software, für Robotersysteme, zu vereinfachen. Dazu stellt dieses Framework eine Sammlung von Tools, Bibliotheken und anderer Software zur Verfügung.[16, 9] Unter ROS werden zur Entwicklung von Software die Programmiersprachen C/C++ und Python verwendet. Um die Konsistenz zu anderer Software die in der WE4 verwendet wird zu wahren, wurde für die im Zuge dieser Bachelorarbeit entwickelte Software die Programmiersprache C/C++ gewählt. Auf den Aufbau und die Funktionen von ROS, wird im Grundlagenkapitel eingegangen.

Rviz

Rviz Die Anwendung "Rviz" ist ein 3D-Visulaisierungstool unter ROS. Diese wird unter anderem für die Simulation von Robotern genutzt. Weiterhin wird mit "Rviz" visualisiert was die Sensoren "sehen" und die Aktoren "tun".[18]

2.1.4 OpenNI

Um auf 3D-Sensoren wie Stereokameras zugreifen zu können, wurde unter "Ubuntu" das quelloffene Framework "OpenNI" verwendet. Es bietet API's, um für 3D-Sensoren, RGB-Kameras, IR-Kameras und Audioeingabegeräte Anwendungen zu entwickeln.[12] Im Zuge dieser Bachelorarbeit wurde hier die Möglichkeit von "OpenNI" genutzt unkompliziert Treiber, für die genannten Gerätetypen, einzubinden. Weiterhin ist dieses Framework notwendig, um Software die auf diesem basiert ausführen zu können. Dieses Framework ist als erster Baustein in dem Gesamtsoftwarekonzept des Systems zu sehen.

2.1.5 SensorKinect

Das Paket "SensorKinect" ist ein Modul für das Framework "OpenNI". Dieses Modul ermöglicht den Zugriff, über die API's von "OpenNI", auf die Stereokamera "Kinect" von "Microsoft". [20] Somit dient "SensorKinect" als Hardwaretreiber für die Stereokamera "Kinect". Als Modul fügt sich "SensorKinect" in den Baustein "OpenNI" des Gesamtsoftwarekozeptes ein. Wird ein zum Treiber von "PrimeSense" kompatibler Sensor genutzt, ist die Installation von "SensorKinect" nicht erforderlich.[12]

2.1.6 Ros-Kinetic-OpenNI

Um die Funktionen des Framework "OpenNI" auch in Verbindung mit ROS verwenden zu können, werden zusätzlich mehrere ROS-Pakete, die im weiteren unter dem Namen "Ros-Kinetic-OpenNI" gefasst werden sollen, benötigt.

2.1.7 NITE

Das Paket "NITE" ist eine Middleware, die sich in die Infrastruktur des Framework "OpenNI" mit eingliedert. Diese Middleware erweitert die Funktionen von "OpenNI" unter anderem um das Tracking von Handbewegungen und Körperbewegungen im ganzen.[14, 12]

2.1.8 OpenNI Tracker

Das ROS-Paket "OpenNI Tracker" greift auf die durch "NITE" implementierten Funktionen zu, um die Positionen von Körpergelenken zu bestimmen und diese in ROS zur Verfügung zu stellen.[13]

2.1.9 MoveIt!

Mit "MoveIt" wurde ein weiteres Framework unter ROS genutzt. Das Framework "MoveIt" ist die "State-Of-The-Art-Software" für Navigationsplanung und Manipulationsplanung in der Robotik unter ROS.[11, 9] Dieses Framework war notwendig, da, basierend auf der zu implementierenden Gestenerkennung, eine Gestensteuerung für einen Roboterarm entwickelt wurde.

2.1.10 Turtle Simulator

Der "Turtle Simulator" ist eine Anwendung in der Schildkröten auf einem 2D-Spielfeld gesteuert werden können. In der ROS-Community wird der "Turtle Simulator" für Tutorials, unter anderem zum erlernen des Umganges mit den TF-Paketen, genutzt.[24]

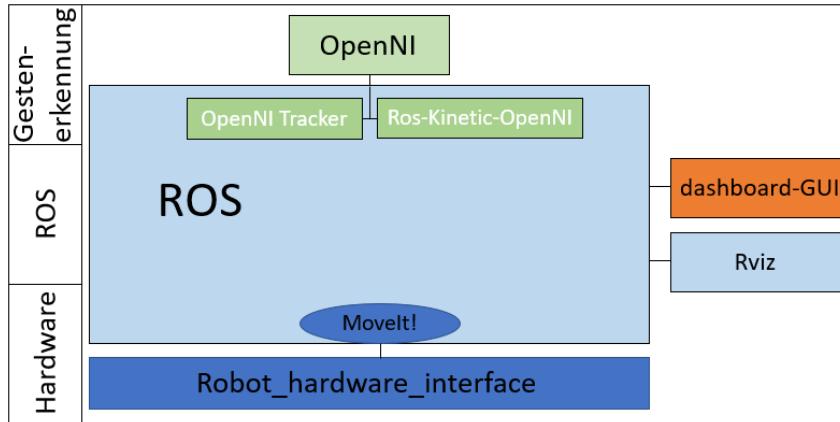


Abbildung 2.1: Softwarearchitektur

2.1.11 Softwarekonzept

In diesem Abschnitt wird das Softwarekonzept des Gesamtsystems dargestellt und erklärt. In der Abbildung 2.1 ist die Softwarearchitektur des Gestenerkennungssystems, mit Anteilen der Steuerung eines Roboterarmes, zu sehen. Die Softwarearchitektur lässt sich in drei Schichten aufteilen. Die ROS-Schicht in der Mitte bildet die Basis und stellt die Umgebung zur Verfügung, in der die Anwendungen laufen werden, welche die Gestenerkennung nutzen. Zu der Gestenerkennungsschicht gehören die Bausteine "OpenNI", "OpenNI Tracker" und "Ros-Kinetic-OpenNI". In der Abbildung 2.2, einer "Abstract Layered View", ist der schematische Aufbau des Bausteins "OpenNI" zu sehen. Wie zu sehen ist, fügen sich die Module "SensorKinect" und "NITE" in diesen ein und werden im Weiteren nicht mehr eigenständig behandelt. Die Hardwareschicht setzt sich aus den Bausteinen "MoveIt!" und "Robot_hw_interface" zusammen. Unter dem Baustein "Robot_hw_interface" sind alle Bestandteile, die zur Kommunikation von "MoveIt!" bis zum Roboterarm "rob_arm_small" und zur Steuerung des Roboterarmes dienen, zusammengefasst. Auf die Anteile der Hardwareschicht und den Roboterarm "rob_arm_small" wird im Hardwareteil dieses Kapitels sowie im Grundlagenkapitel näher eingegangen. Mit dem Baustein Rviz können 3D-Punktfolgen, basierend auf den von "Ros-Kinetic-OpenNI" bereitgestellten Daten, dargestellt werden. Parallel zur Punktfolge kann das von "OpenNI Tracker" erkannte Skelett über TF's visualisiert werden. Der Baustein "dashboard-GUI" bietet die Möglichkeit unkompliziert GUI's zu erstellen. Dies ist z.B. für das einfache und übersichtliche Debugging von Software und Hardware interessant. Die Abbildung 2.3 zeigt das Softwarekonzept des Gesamtsystems. Den äußeren Rahmen bildet hier das Hostbetriebssystem "Windows 10". Auf diesem Hostbetriebssystem läuft im "VmWare Player", einer virtuellen Maschine, ein "Ubuntu" der Version 16.04 LTS. Unter diesem Betriebssystem fügt sich dann die Softwarearchitektur des Gestenerkennungssystems ein.

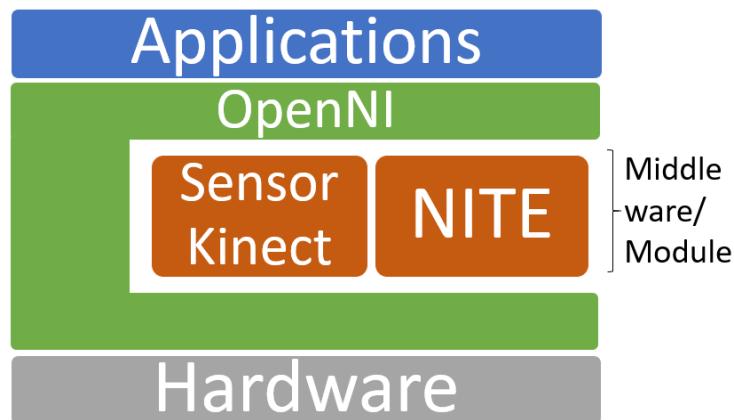


Abbildung 2.2: Abstract Layered View - OpenNI

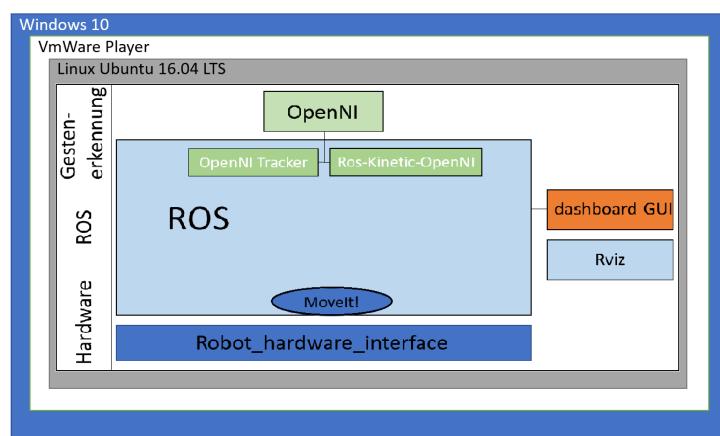


Abbildung 2.3: Softwarekonzept



Abbildung 2.4: Kinect

2.2 Hardware

In diesem Abschnitt wird die Hardware des Gesamtsystems kurz vorgestellt. Auf die Funktionsweise und die Grundlagen zu elementarer Hardware , wird erst im Grundlagenkapitel näher eingegangen.

2.2.1 Rechner

Der verwendete Rechner, auf dem das Hostbetriebssystem lief, hatte folgende Systemspezifikation:

Prozessor: AMD Ryzen Threadripper 1900X

Arbeitspeicher: 64 GB

Grafikkarte: AMD Radeon RX580

Bei dem verwendeten Rechner ist darauf zu achten, dass der verbaute Prozessor die Befehlsätze SSE3, SSE4 und SSSE3 unterstützt. Bei nicht unterstützen der genannten Befehlssätze, können Fehler zur Laufzeit der verwendeten Software auftreten. Weiterhin ist darauf zu achten das mindestens ein vollwertiger USB 3.X Port belegbar ist.

2.2.2 Kinect

Für das Gestenerkennungssystem wurde die "Kinect for Windows"[8] Modell 1517 von "Microsoft", zu sehen in Abbildung 2.4, als Stereokamera genutzt. Es ist hier darauf zu achten, dass die "Kinect" nur an einem vollwertigen USB 3.X Port betrieben wird. Wenn der genutzte USB Port nicht die volle elektrische Leistung bringt kann es zu Problemen während dem Betrieb kommen.

2.2.3 Roboterarm "rob_arm_small"

Basierend auf dem implementierten Gestenerkennungssystem wurde eine Gesteuerung für den Roboterarm "rob_arm_small" entwickelt. Dieser Roboterarm wurde von der Firma "CrustCrawler" unter der Bezeichnung "SG6-UT" vertrieben. Wie der Abbildung 2.5 zu entnehmen ist, ist der

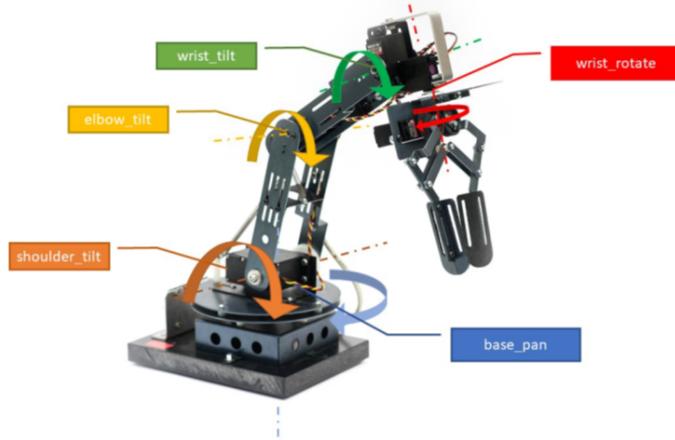


Abbildung 2.5: Freiheitsgrade des Roboterarms[27]

Joint	Min. Winkel			Max. Winkel		
	[rad]	[°]	[μs]	[rad]	[°]	[μs]
base_pan	-0.785	-45	1950	+0.785	+45	1050
shoulder_tilt	-0.261	-15	1854	+1.57	+90	1067
elbow_tilt	0.0	0	1906	+1.57	+90	1027
wrist_tilt	0.0	0	1934	+1.57	+90	1042
wrist_rotate	-0.785	-45	1959	+0.785	+45	1032

Tabelle 2.1: Winkelwerte der Gelenke[27]

”rob_arm_small“ ein Roboterarm mit sechs Freiheitsgraden.[27] In der Tabelle 2.1 sind die Winkelwerte, mit zugehörigen Pulsweiten, festgehalten. Die Griffweite des Grippers, mit zugehöriger Pulsweite, ist in der Tabelle 2.2 festgehalten. Diese Werte wurden bereits in einer vorangegangenen Projektarbeit[28] durch Christian Waldner ermittelt.

2.2.4 Mikrocontrollerboard

Das Mikrocontrollerboard, ein ”STM32F746ZG“ von ”STMicroelectronics“, wird zur Erzeugung der PWM-Signale für die Servomotoren des Roboterarmes ”rob_arm_small“ verwendet. Die auf dem Mikrocontrollerboard laufende Software wurde durch die WE4 zur Verfügung gestellt. Die Verbindung zum Rechner, auf dem das Gestenerkennungssystem implementiert ist, wird über

Joint	Min. Winkel			Max. Winkel		
	[rad]	[μs]	Öffnung [cm]	[rad]	[μs]	Öffnung [cm]
z_gripper	0.0	1555	6,4	1.0	1156	3,3

Tabelle 2.2: Griffweite des Greifers[27]

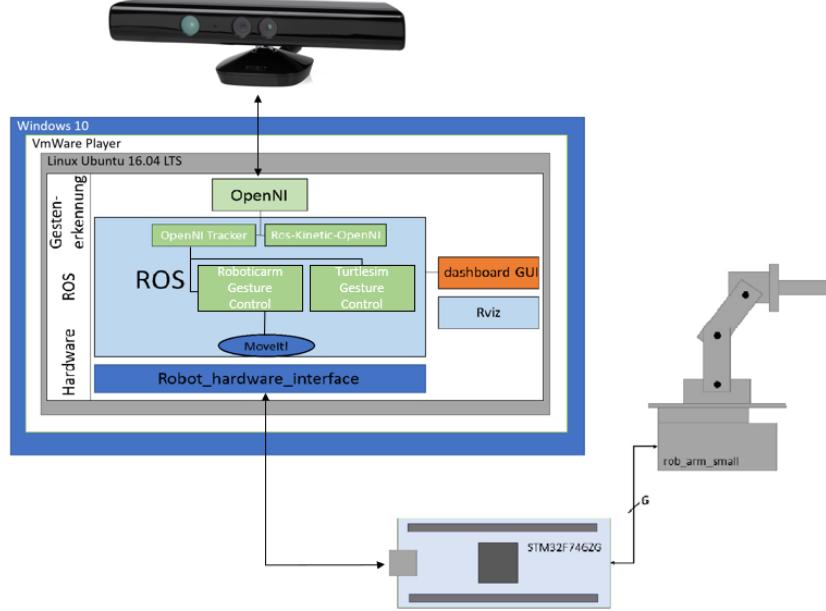


Abbildung 2.6: Aufbau Gesamtsystem

Ethernet hergestellt.[27] Abgesehen von kleinen Anpassungen, die im vierten Kapitel erläutert werden, wurde die bereitgestellte Software nicht in der Funktion verändert.

2.3 Systemaufbau

Im Systemaufbau werden das Softwarekonzept und die verwendete Hardware zusammengeführt. In der Abbildung 2.6 ist der Aufbau des Gesamtsystems dargestellt. In diesem Aufbau ist, mit dem Mikrocontrollerboard und dem Roboterarm "rob_arm_small", der Hardwareanteil der Gestensteuerung für den Roboterarm enthalten. Weiterhin wurden die Pakete "Roboticarm Gesture Control" und "Turtlesim Gesture Control" in den Aufbau eingefügt. Als integraler Teil dieser Bachelorarbeit wird auf die Entwicklung, Implementation und die Funktionsweise der beiden Pakete im vierten Kapitel eingegangen. Der Baustein "OpenNI" hat, wie in Abbildung 2.6 zu sehen, Zugriff auf die Hardware der "Kinect". Über die API's von "OpenNI" greifen die Pakete "OpenNI Tracker" und "Ros-Kinetic-OpenNI" auf die verarbeiteten Bilddaten der "Kinect" zu. Das Paket "OpenNI Tracker" nutzt zusätzlich das durch NITE implementierte Skeletttracking. Das Framework "MoveIt!" stellt über das "Robot_hardware_interface" eine Verbindung zum Mikrocontrollerboard her. Den Anwendungen in der ROS-Schicht bietet "MoveIt!" Funktionen zur generierung von Steueranweisungen für Roboter. In diesem Systemaufbau werden die Steueranweisungen an das Mikrocontrollerboard gesendet und von diesem in PWM-Signale, für die Servomotoren des "rob_arm_small", umgewandelt.

3 Grundlagen

In diesem Kapitel wird auf die Grundlagen zu verwendeter Hard- und Software eingegangen. Dies soll Hintergrundinformationen zur gesamten Arbeit und speziell für die Ausführungen im vierten Kapitel bereitstellen. Hierbei wird sich nur auf die wesentlichen Komponenten, für das Gesamtsystem, beschränkt.

3.1 Stereokamera

Eine Stereokamera ist ein Kamerasystem, dass zur Gewinnung von Tiefenbildern genutzt wird. Diese Tiefenbilder enthalten, im Gegensatz zu normalen Farbbildern, den Abstand einzelner Punkte zum Sensor der Kamera. Diese Kamerasysteme haben immer zwei optische Sensoren zur Bilderzeugung. Es gibt unter anderem Systeme mit zwei RGB-Sensoren sowie Systeme mit einem RGB-Sensor und einem Infrarotsensor.

3.1.1 Typen

Unter den Stereokameras gibt es verschiedene Typen, die sich in zwei Gruppen einteilen lassen: Diese Gruppen sind, Kameras mit passiven Verfahren und Kameras mit aktiven Verfahren. Für diese Arbeit wurden die Beschreibungen auf die gängigsten Typen, von Stereokameras, begrenzt.

Embedded Stereo

Die "Embedded Stereo" Kameras nutzen das passive Stereoverfahren, um Tiefenbilder zu erzeugen. Diese Kamerasysteme haben meist zwei RGB-Sensoren um Bilder aufzunehmen, sowie einen eingebetteten Mikroprozessor oder FPGA, für die Berechnung von Tiefenbildern, integriert. Das passive Stereoverfahren ist an das menschliche Sehen angelehnt, bei dem aus zwei 2D-Bildern ein 3D-Bild gemacht wird. Das passive Verfahren, zur Erzeugung der Tiefenbilder, lässt sich in mehrere Schritte aufteilen. Zu erst wird mit beiden RGB-Sensoren gleichzeitig ein Bild aufgenommen. Anschließend wird an beiden Bildern eine Merkmalsextraktion durchgeführt. Diese Bildmerkmale, auch als "Keypoints" bezeichnet, lassen Unterschiede zu Ihrer Umgebung erkennen. Die gefundenen Bildmerkmale, beider Bilder, werden bei der Korrespondenzsuche miteinander verglichen. Nachdem die Korrespondenzsuche abgeschlossen ist, ist es noch notwendig falsche Korrespondenzen aus den Ergebnissen herauszufiltern. Mit den endgültigen korrespondierenden Bildmerkmalen kann nun, mittels Triangulation, die Entfernung zu diesem Merkmal im Bild errechnet werden.[19, 10]

Um den Prozessor des nutzenden Hauptsystems zu entlasten, werden die Berechnungen, für

3 Grundlagen

das beschriebene Verfahren, auf dem eingebetteten Mikroprozessor bzw. FPGA durchgeführt. Da der hohe Rechenaufwand auf eine erhöhte Leistungsaufnahme schließen lässt, werden die "Embedded Stereo" Kameras nicht für mobile Systeme empfohlen.

Time-Of-Flight

Eine "Time-Of-Flight" Kamera gehört zu den Kamerasystemen die aktive Verfahren, zur Generierung von Tiefenbildern, nutzen. Das von "Time-Of-Flight" Kameras genutzte Verfahren ist das Laufzeitverfahren. Bei dem Laufzeitverfahren wird ein Signal von der Kamera ausgesendet und die Zeit ermittelt wie lange das Reflektierte Signal gebraucht hat, um wieder an der Kamera aufzutreffen. Aus der Laufzeit t und Geschwindigkeit v des Signals kann, über die Formel $S = vt$, direkt auf die Strecke S zum reflektierenden Objekt geschlossen werden. Als typische Emitter-Sensor Systeme, in dem Laufzeitverfahren, zählen Radar-, Ultraschall- und Infrarotsysteme.[19]

Die "Time-Of-Flight" Kameras nutzen in der Regel Infrarotsysteme. Die gängigen "Time-Of-Flight" Kameras haben meist einen RGB-Sensor, einen oder mehrere Infrarotemitter und einen Infrarotsensor verbaut. Aufgrund der hohen Ausbreitungsgeschwindigkeit von Licht, werden hohe Anforderungen an die Elektronik, zur Zeitmessung, gestellt. Unter anderem legt die kleinste messbare Zeitdifferenz Δt die maximale Tiefenauflösung ΔR , mit der Formel $\Delta R = \frac{v\Delta t}{2}$, fest.[6] Durch diese hohen Anforderungen an die Elektronik, sind hochauflösende "Time-Of-Flight" Kameras kostenintensiv. Da hier Licht, im Infrarotbereich, detektiert wird, sind "Time-Of-Flight" Kameras nicht für Bereiche mit direkter Sonneneinstrahlung geeignet.[1] In Bereichen mit direkter Sonneneinstrahlung empfehlen sich stattdessen Kameras die auf passive Verfahren zur Tiefengewinnung zurückgreifen.

Infrarotmuster

Wie die "Time-Of-Flight" Kameras nutzen auch "Infrarotmuster" Kameras ein aktives Verfahren, zur Gewinnung von Tiefenbildern. Das Triangulationsverfahren, mit dem Ansatz des codierten Lichts, wird von den "Infrarotmuster" Kameras verwendet. Bei diesem Ansatz wird ein Infrarotmuster, mit bekanntem Code, in den Raum projiziert und mit einer Infrarotkamera aufgenommen. Somit sind, über den bekannten Code, alle vorher definierten Messpunkte im Bild eindeutig identifizierbar. Aufgrund der Kenntnis über die Position von Infrarotemitter und Infrarotkamera, kann für jeden Messpunkt der Abstand zur Kamera, über die Triangulation, errechnet werden.[6, 10] Wie die "Time-Of-Flight" Kameras sind auch die "Infrarotmuster" Kameras, aufgrund der Verwendung von Infrarotlicht, nicht für Bereiche mit direkter Sonneneinstrahlung geeignet. Die für die Umsetzung des Gestenerkennungssystems dieser Arbeit genutzte Stereokamera, die "Kinect", ist eine "Infrarotmuster" Kamera. Somit ist das, in dieser Bachelorarbeit, entwickelte Gestenerkennungssystem nur für Innenbereiche geeignet.

3.2 OpenNI

Das Framework "OpenNI" soll in diesem Abschnitt, in seiner Funktion und seinem Aufbau, beschrieben werden.

3.2.1 Was ist OpenNI

"OpenNI" ist ein Framework, dass verschiedene Programmiersprachen erlaubt und für verschiedene Plattformen verfügbar ist. Weiterhin definiert es API's zur Entwicklung von Anwendungen, die natürliche Interaktion, wie Sprache und Gesten, verwenden. Die API's von OpenNI bestehen aus mehreren Schnittstellen, zur Entwicklung von NI-Anwendungen. Der Hauptzweck von "OpenNI" ist es, eine Standard-API zur Verfügung zu stellen, die eine Kommunikation mit folgenden Komponenten ermöglicht:

- optischen und akustischen Sensoren
- Middleware, die Funktionen zur Verarbeitung von optischen und akustischen Sensordaten implementiert

"OpenNI" liefert hierzu Schnittstellen die von den Sensorgeräten implementiert werden müssen sowie Schnittstellen die von der Middleware implementiert werden müssen. Durch diesen Ansatz werden Abhängigkeiten, zwischen Sensorgeräten und Middleware, vermieden. Dies ermöglicht es Anwendungen, ohne den Aufwand der Portierung, auf der Basis von verschiedenen Middleware's zu arbeiten. Des Weiteren bietet "OpenNI" die Möglichkeit, Anwendungen zu entwickeln die Sensorrohdaten verwenden, unabhängig vom Sensor der diese liefert.[12]

3.2.2 Module

Als Module werden Gerätetreiber und Middleware bezeichnet, die im "OpenNI" Framework registriert wurden. Die folgenden Module werden von "OpenNI" unterstützt:[12]

Sensormodule

- 3D Sensor
- RGB-Kamera
- Infrarotkamera
- Mikrofon

Middleware-Module

- Middleware zur Ganzkörperanalyse
- Middleware zur Analyse der Handposition
- Middleware zur Gestenerkennung
- Middleware zur Analyse von Szenen in Bildern

3.2.3 Kompatibilität und Verfügbarkeit

Die Entwickler von "OpenNI" garantieren volle Rückwärtskompatibilität[12]. Dies bedeutet das Anwendungen, die auf Basis irgendeiner Version von "OpenNI" entwickelt wurden, mit neueren Versionen von "OpenNI" arbeiten können.

"OpenNI" ist für folgende Betriebssysteme verfügbar[12]:

- Windows XP und aktueller
- Linux Ubuntu 10.10 und aktueller

3.3 ROS

Dieser Abschnitt soll eine Einführung in das Konzept, die Komponenten und das Vokabular von ROS geben. Diese Informationen sind, zum Verständnis des vierten Kapitels, elementar. Die folgenden Punkte werden in dieser Einführung behandelt:

- Was ist ROS
- Konzept und Komponenten
- Wichtige ROS-Befehle

3.3.1 Was ist ROS

Da ROS eine Vielzahl von Aufgaben eines Betriebssystems übernimmt, aber als Umgebung ein Linux Betriebssystem benötigt, wird es oft als "Meta-Betriebssystem" bezeichnet. Die elementare Aufgabe von ROS ist es, eine Kommunikation zwischen Nutzer, Betriebssystem und externer Hardware bereitzustellen. Zur externen Hardware, mit der kommuniziert werden kann, zählen beispielsweise Sensoren, Kameras und Roboter. In dem ROS sich wie ein Betriebssystem verhält, bringt es den Vorteil der Hardwareabstraktion. Somit kann ein Nutzer einen Roboter steuern, ohne große Kenntnis über dessen Hardware- und Softwaredetails zu haben. Als quelloffenes Framework, ist ROS für jeden frei verfügbar. Dieser Ansatz wird auch bei den meisten ROS-Paketen verfolgt und diese unter der BSD-3-Lizenz veröffentlicht. Aufgrund dieser Voraussetzungen, werden neue Entwicklungen und neues Wissen, in der ROS-Community, schnell

verbreitet. Aber auch für Unternehmen ist ROS interessant, da auch kommerzielle Produkte mit ROS entwickelt werden können. Hierfür ist die einzige Bedingung, dass der Copyright-Vermerk der ursprünglichen Software zitiert wird. [3, 16]

3.3.2 Konzept und Komponenten

Das Konzept von ROS teilt sich in drei Konzeptschichten auf[16]:

- ROS Dateisystem
- ROS Computation Graph
- ROS Community

ROS Dateisystem

Das ROS-Dateisystem setzt sich aus folgenden Komponenten zusammen:

- **Packages:** In Paketen wird die Software in ROS organisiert.
- **Manifests:** Im Manifest sind Metadaten über das Paket enthalten. Diese Manifests sind in XML geschrieben.
- **Stacks:** Pakete, die gemeinsam eine Funktion bereitstellen, werden in Stacks zusammengefasst.
- **Stack Manifest:** Im Stack Manifest sind Metadaten über das Stack enthalten.[3, 16]

ROS Computation Graph

Das Netzwerk von Prozessen, die eine gemeinsame Aufgabe erfüllen, in ROS bildet den "Computation Graph". Die folgenden Komponenten versorgen den Graph mit Daten:

- **Nodes:** Als Knoten werden Prozesse in ROS bezeichnet, welche eine Aktion ausführen. Diese Knoten haben die Möglichkeit sich bei dem Master zu registrieren und untereinander zu kommunizieren. Die Verbindungsinformationen, zur Kommunikation untereinander, erhalten die Knoten vom Master. Hierfür melden die Knoten dem Master welche Topics abonniert werden und auf welchen Topics Daten veröffentlicht werden. Knoten können auf verschiedene Arten gestartet werden. Zum einen über ein Terminalfenster durch ausgeführte Befehle, zum anderen als Teil eines Programms, geschrieben in Python oder C++. [3, 16]
- **Master:** Durch den Master wird die Kommunikation zwischen Knoten aufgebaut. Der Master bietet dafür Namensdienste und Dienste zum registrieren an. In Abbildung 3.1 ist ein Kommunikationsaufbau beispielhaft dargestellt. Im ersten Schritt meldet sich der Knoten "Camera" beim Master an und informiert diesen darüber, dass, unter dem

3 Grundlagen

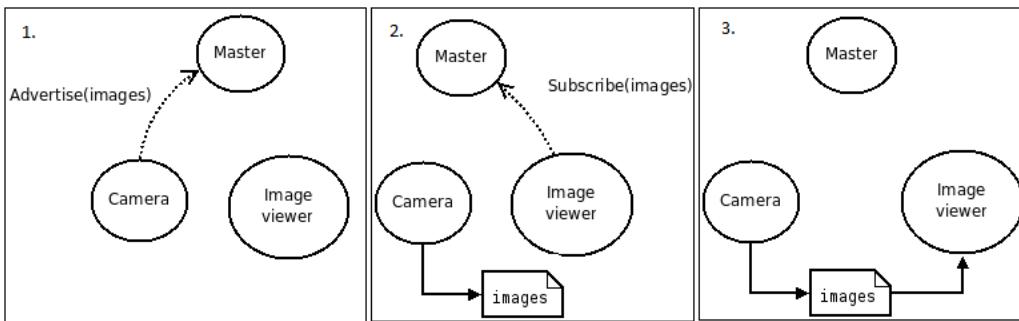


Abbildung 3.1: Kommunikationsaufbau von Knoten

Topic "images", Bilder veröffentlicht werden. Im zweiten Schritt meldet sich der Knoten "Imageviewer" beim Master an und abonniert über diesen das Topic "images". Im dritten Schritt beginnt der Knoten "Imageviewer" die Bilder direkt über das Topic "images" zu empfangen.[3, 16]

- **Parameter Server:** Der "Parameter Server" stellt eine geteilte Bibliothek von Parametern dar, welche innerhalb des Masters ausgeführt wird. Knoten können zur Laufzeit Parameter in dieser Bibliothek speichern und abrufen.[3, 16]
- **Messages:** Die Kommunikation der Knoten untereinander findet durch Nachrichten statt. Diese Nachrichten sind Datenstrukturen, welche die auszutauschenden Informationen in typisierten Feldern beinhalten. Von diesen Nachrichten gibt es verschiedene Typen, welche unterschiedliche Aufgaben haben. Es können auch neue Nachrichtentypen definiert werden.[3, 16]
- **Topics:** Die Verteilung der Nachrichten, zwischen den Knoten, wird über einen "Publisher-Subscriber" Mechanismus durchgeführt. Um diesen Mechanismus umzusetzen werden Themen verwendet. Die Knoten können Nachrichten zu einem Thema veröffentlichen, welches über einem Namen identifiziert wird. Ein anderer Knoten, der an den Nachrichten interessiert ist, kann dieses Thema abonnieren, um die veröffentlichten Nachrichten zu erhalten. Knoten können unter mehreren Themen veröffentlichen und sie können mehrere Themen abonnieren.[3, 16]
- **Services:** Um, neben dem "Publisher-Subscriber" Mechanismus, Knoten die Möglichkeit zu geben Anfragen von anderen Knoten zu erhalten und zu beantworten werden Dienste verwendet.[3, 16]

ROS Community

- **Distributions:** Neue Versionen von ROS werden über Distributionen verteilt. Diese Distributionen setzen sich aus verschiedenen "Stacks" zusammen, die jeweils auch eine Versionsnummer tragen. Durch die Verteilung über Distributionen, wird die Installation von ROS erleichtert.[16]

Befehl	Aktion	Befehlsbeispiele und Beispiele für Unterbefehle
roscore	Durch diesen Befehl wird der ROS Master gestartet	\$ roscore
rosrun	Führt ein Programm aus und erstellt einen Knoten	\$ rosrun [Paketname] [Ausführbare Datei]
rosnode	Zeigt Informationen zu Knoten und listet aktive Knoten auf	\$ rosnode info [Name des Knoten] \$ rosnode <Unterbefehl> Unterbefehle: list
rostopic	Zeigt Informationen über Topics an	\$ rostopic <Unterbefehl> <Name vom Topic> Unterbefehle: echo, type, info
rosmsg	Zeigt Informationen über Message-Typen an	\$ rosmsg <Unterbefehl> [Paketname] / [Message Typ] Unterbefehle: show, type, list
rosservice	Zeigt Informationen zu Services an	\$ rosservice <Unterbefehl> [Servicename] Unterbefehle: args, call, find, info, list, type
rosparam	Liefert und setzt Werte von Parametern	\$ rosparam <Unterbefehl> [Parameter] Unterbefehle: get, set, list, delete

Tabelle 3.1: Wichtige Befehle in ROS[3, 16]

- **Repositories:** In der ROS Community werden Pakete und Quellcode über Repositories, wie "github"¹, verteilt.[16]
- **The ROS Wiki:** Im ROS Wiki² sind Dokumentation und Tutorials rund um ROS veröffentlicht. Nach einer Anmeldung können hier eigene Dokumentationen, Tutorials, Korrekturen und andere Informationen veröffentlicht werden.[16]
- **ROS Answers:** "ROS Answers"³ ist ein weiteres Portal für Informationen über ROS. Hier können Fragen gestellt und Antworten geteilt werden.

Durch diese organisierte Community werden Lösungen für Probleme schneller gefunden und diese Lösungen an zentraler Stelle bereitgestellt.

3.3.3 Wichtige ROS-Befehle

In der Tabelle 3.1 ist eine Auswahl von wichtigen ROS-Befehlen aufgeführt:

3.4 MoveIt!

"MoveIt!" ist ein Framework das Fähigkeiten für Manipulation, Bewegungsplanung, Steuerung und mobile Manipulation bietet. Zusätzlich bietet "MoveIt" verschiedene Tools, welche bei der Entwicklung von Anwendungen unterstützen. Weiterhin steht auch hinter "MoveIt" eine Community, die gemeinsam "MoveIt!" erweitert und pflegt. Ein weiterer Vorteil ist, dass für viele gängige Robotermodelle bereits Dokumentationen, in Verbindung mit "MoveIt!", vorhanden

¹<https://github.com/>

²<http://wiki.ros.org/de>

³<https://answers.ros.org/>

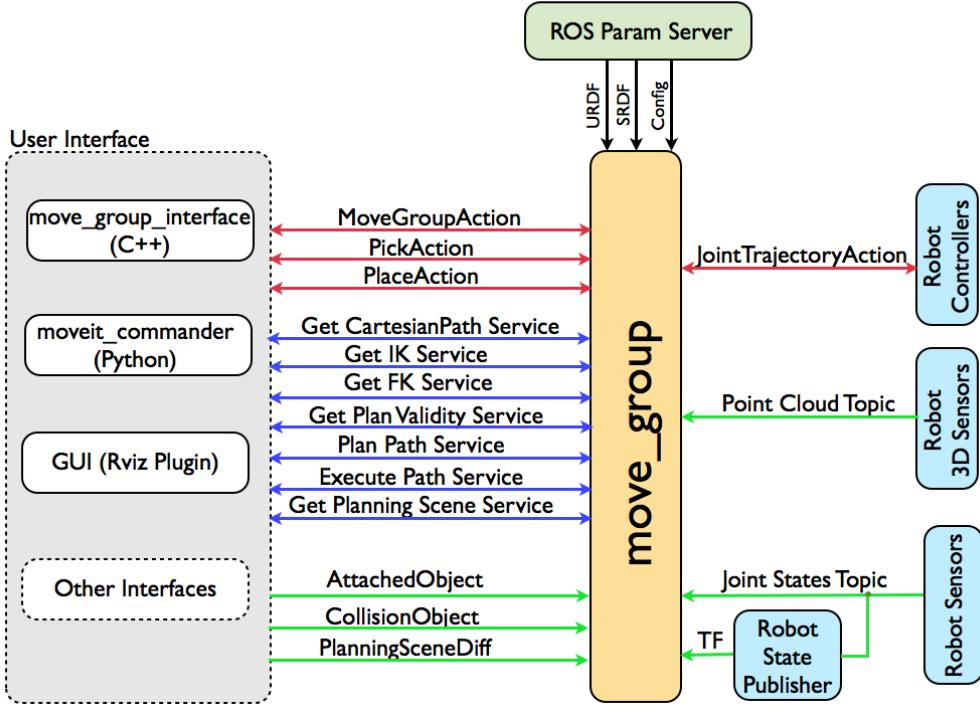


Abbildung 3.2: Architektur MoveIt!⁴

sind.[9] Dies erleichtert den Einstieg in die Entwicklung von Anwendungen mit "MoveIt!". Um die Erstellung eines Pakets mit "MoveIt!" weiter zu erleichtern, wird ein Setup-Assistent bereitgestellt. Dieser Setup-Assistent bietet die Möglichkeit neue Roboter zu importieren und für diese alle benötigten Komponenten eines "MoveIt"-Paketes zu generieren[9]. Erläutert werden in diesem Abschnitt nur Informationen zu "MoveIt!", die essentiell für diese Arbeit sind. Für weitergehende Information wird auf die folgenden Quellen verwiesen:

- Die MoveIt! Webseite⁵
- Die MoveIt! Tutorials⁶
- Den MoveIt! Quellcode⁷

3.4.1 Architektur

In der Abbildung 3.2 ist die Architektur von "MoveIt!" dargestellt.

⁴(besucht am 03.06.2019): <https://moveit.ros.org/documentation/concepts/>

⁵<https://moveit.ros.org/>

⁶https://ros-planning.github.io/moveit_tutorials/

⁷<https://github.com/ros-planning>

Kinematik

Für die direkte Kinematik bietet "MoveIt!" eine native Implementation, andererseits bietet es für die inverse Kinematik eine Plugin basierte Architektur. Dies bedeutet das die Berechnung der inversen Kinematik jederzeit, durch Austausch des Plugin, an die eigenen Bedürfnisse angepasst werden kann.[9]

Bewegungsplanung

Die Bewegungsplanung wird durch eine Plugin-Schnittstelle implementiert. Dies ermöglicht es "MoveIt!" mit verschiedenen Bewegungsplanern, aus einer Vielzahl von Bibliotheken, zu kommunizieren. Dieser Ansatz betont noch einmal die weitgehende Erweiterbarkeit und Anpassungsfähigkeit von "MoveIt!".[9]

Planning Scene

Die "Planning Scene" repräsentiert die Welt um den Roboter herum und beinhaltet auch den aktuellen Zustand vom Roboter selbst. Die "Planning Scene"-Schnittstelle bietet den primären Zugriff für Nutzer, um den Zustand der Welt, in der der Roboter operiert, zu verändern.[9]

Trajectory Processing

Im Gegensatz zu reinen Bewegungsplanern bietet "MoveIt!" zusätzlich ein "Trajectory Processing". Dies bedeutet das neben dem Weg auch die Beschleunigung, an bestimmten Punkten, berechnet wird. Somit kann der Weg zeitparametrisiert werden. Notwendige Limits für die Beschleunigung der einzelnen Gelenke, werden aus der Datei `joint_limits.yaml` ausgewertet.[9]

3.4.2 Das move_group_interface

Das "move_group_interface" ist eine Benutzerschnittstelle, welche API's für den "move_group"-Knoten bereitstellt. Die Schnittstelle abstrahiert die ROS-API auf "MoveIt!" und vereinfacht deren Nutzung dadurch. [9]

Pose Goal planen

Bei einem "Pose Goal" wird die Koordinate im Raum und die einzunehmende Orientierung des Endeffektors angegeben. Der folgende C++ Code zeigt wie auf ein "Pose Goal" geplant werden kann:

3 Grundlagen

```
1 moveit::planning_interface::MoveGroup group("rob_arm_group");  
2  
3 geometry_msgs::Pose pose;  
4 pose.orientation.w = 1.0;  
5 pose.position.x = 0.22;  
6 pose.position.y = -0.5;  
7 pose.position.z = 0.0;  
8 group.setPoseTarget(pose);  
9  
10 moveit::planning_interface::MoveGroup::Plan my_plan;  
11 bool success = group.plan(my_plan);
```

Listing 3.1: Planen auf ein Pose Goal

In Zeile eins wird die "MoveGroup", mit der gearbeitet werden soll, definiert. Als nächstes werden der Message "pose" die benötigten Werte zugewiesen. Die Message "pose" wird der Funktion "setPoseTarget()" übergeben und mit der Funktion "plan()" ein Plan berechnet.

Joint Goal planen

Bei einem "Joint Goal" werden für jedes Gelenk die Winkelwerte angegeben, welche in der Endposition eingenommen werden sollen. Wie auf ein "Joint Goal" geplant wird zeigt der folgende C++ Code:

```
1 std::vector<double> group_joint_values;  
2 group.getCurrentState() -> copyJointGroupPositions(group.  
3     getCurrentState()  
4 -> getRobotModel() -> getJointModelGroup(group.getName()),  
5     group_joint_values);  
6 group_joint_values[0] = -0.700;  
7 group.setJointValueTarget(group_joint_values);  
8 moveit::planning_interface::MoveGroup::Plan my_plan;  
success = group.plan(my_plan);
```

Listing 3.2: Planen auf ein Joint Goal

In diesem Beispiel werden in Zeile zwei bis Zeile 4 die aktuellen Winkelwerte der Gelenke kopiert. Darauf folgend wird ein Winkelwert verändert und mit der Funktion "setJointValueTarget()" als neues Ziel gesetzt.

Position Target planen

Wenn die Orientierung des Endeffektors nicht von Bedeutung ist, kann auf ein "Position Target" geplant werden. Bei dem "Position Target" wird nur die Koordinate im Raum angegeben, zu der sich der Endeffektor bewegen soll. Im folgenden C++ Code wird beispielhaft auf ein "Position Target" geplant:

```

1  tf2 :: Vector3 distance;
2  distance[0] = conversion_factor * (transformStamped_left_hand_1 .
3      transform . translation .x - 0.15);
4  distance[1] = conversion_factor * (transformStamped_left_hand_1 .
5      transform . translation .z*(-1.0));
6  distance[2] = conversion_factor * (transformStamped_left_hand_1 .
7      transform . translation .y - 0.15);
8
9  group . setPositionTarget(distance [0], distance [1], distance [2]);
moveit :: planning_interface :: MoveGroup :: Plan my_plan;
success = group . plan(my_plan);

```

Listing 3.3: Planen auf ein Position Target

In dem Beispiel werden der Funktion "setPositionTarget()" die drei Komponenten, der Koordinate im Raum, als Gleitkommazahlen übergeben und diese Koordinate als neues Ziel gesetzt.

Plan ausführen

Um einen Plan auszuführen, werden die Funktionen "move()" und "execute()" bereitgestellt. Um einen vorher durch die Funktion "plan()" berechneten Plan auszuführen wird die Funktion "execute()" verwendet. Die Funktion "move()" berechnet erst einen Plan und führt diesen direkt aus.

```

1  moveit :: planning_interface :: MoveGroup :: Plan my_plan;
2
3
4 // entweder
5 success = group . plan(my_plan);
6 group . execute(my_plan);
7
8 // oder
9 group . move(my_plan);

```

Listing 3.4: Plan ausführen

4 Implementation des Gestenerkennungssystems mit ROS

In diesem Kapitel wird die gesamte Implementation, des Gestenerkennungssystems, beschrieben. Das Kapitel ist so gegliedert, sodass die Abfolge der Schritte im Text auch die einzuhaltende Implementationsreihenfolge darstellt. Bei nicht einhalten der Implementationsreihenfolge sind Konflikte zwischen Paketen nicht auszuschließen.

4.1 Vorbereitung

Die Vorbereitenden Maßnahmen werden in diesem Abschnitt beschrieben.

4.1.1 Catkin Workspace erstellen

Um veränderte und eigene Pakete einfach zu compilieren und zu bauen, ist es notwendig einen Catkin-Workspace zu erstellen. Die folgenden Befehle sind in einem Terminal einzugeben, um einen Catkin-Workspace zu erstellen und initial zu bauen:

```
$ mkdir -p ~/catkin_ws/src  
$ cd ~/<name des Workspace>/  
$ catkin_make
```

Nachdem der Catkin-Workspace erstellt und gebaut wurde, sollte die Ordnerstruktur wie in Abbildung 4.1 dargestellt aussehen. Im Ordner "src" wurde eine Datei mit dem Namen "CMake-Lists.txt" generiert. Im Ordner "devel" wurden verschiedene Setup-Dateien generiert. Um den Pfad des Catkin-Workspace in die Umgebungsvariablen aufzunehmen, muss der folgende Befehl ausgeführt werden:

```
$ source devel/setup.bash
```

Weitergehende Informationen zu "Catkin" sind im ROS Wiki⁸ zu finden.

4.1.2 Abhängigkeiten

Die Komponenten des Gestenerkennungssystems haben Abhängigkeiten zu anderen Paketen. Diese Abhängigkeiten müssen vor der Implementation, des Gestenerkennungssystems, erfüllt

⁸<http://wiki.ros.org/catkin>

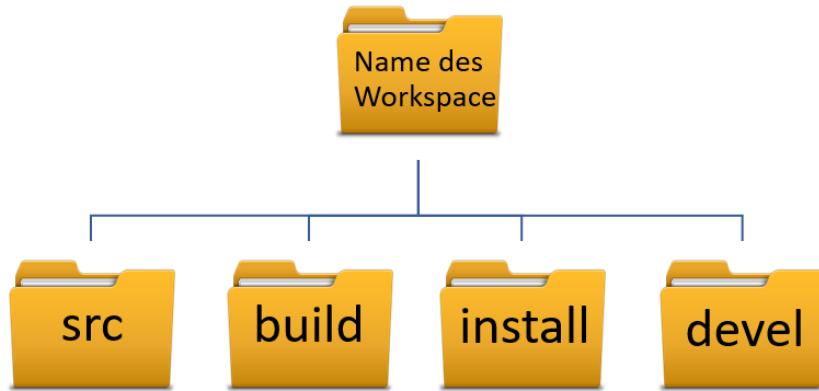


Abbildung 4.1: Ordnerstruktur Catkin-Workspace

sein. Um diese Abhängigkeiten zu erfüllen, werden mit den folgenden Befehlen die benötigten Pakete installiert:

```
$ sudo apt-get install git build-essential python libusb-1.0-0-dev freeglut3-dev openjdk-8-jdk  
$ apt-get install doxygen graphviz mono-complete
```

4.2 Installation der Komponenten

In diesem Abschnitt werden die Komponenten des Gestenerkennungssystems installiert. Um die Installation von "OpenNI" und der dazu gehörigen Module von den Paketen getrennt zu halten, wird ein neuer Ordner erstellt. Der folgende Befehl erstellt den neuen Ordner:

```
$ mkdir -p ~/kinect
```

4.2.1 OpenNI installieren

Um "OpenNI" zu installieren wird erst ein Repository, in den Ordner "kinect", geklont und "OpenNI" aus dieser Kopie heraus installiert. Die folgenden Befehle führen dies aus:

```
$ cd ~/kinect  
$ git clone https://github.com/OpenNI/OpenNI.git  
$ cd OpenNI  
$ git checkout Unstable-1.5.4.0  
$ cd Platform/Linux/CreateRedist  
$ chmod +x RedistMaker  
$ ./RedistMaker  
$ cd ../Redist/OpenNI-Bin-Dev-Linux-x64-v1.5.4.0  
$ sudo ./install.sh
```

4.2.2 Ros-Kinetic-OpenNI installieren

4.2.3 NITE installieren

4.2.4 OpenNI Tracker installieren

4.3 Start und Test des Gestenerkennungssystems

4.3.1 Starten der Komponenten

4.3.2 Mit Rviz visualisieren und überprüfen

5 Implementation von Gestensteuerungen basierend auf dem Gestenerkennungssystem

5.1 Vorbereitung

5.1.1 Installation Movelt!

5.1.2 Einrichtung der Pakete rob_arm_small und rob_arm_small_hw_interface

5.1.3 Erstellung eines ROS Paketes

5.2 turtlesim_gesture_control

5.3 roboticarm_gesture_control

Dieses Kapitel ist ganz der eigenen Arbeit gewidmet. Hier werden alle erzielten Ergebnisse und eigenen Arbeiten dokumentiert. An erster Stelle sind hier natürlich erstellte Platinen oder auch eigens erstellte Software zu nennen. Darüber hinaus werden hier aber auch durchgeführte Messreihen dokumentiert. Ergebnisse von Fahrtest, erstellte Umgebungskarten oder auch Screenshots der eigenen Software können ebenfalls hier aufgeführt werden. Allerdings werden hier noch keine Interpretationen hinsichtlich der Aufgabenerfüllung vorgenommen. Lediglich Beobachtungen und reine Fakten sind hier von Bedeutung.

Zusätzlich zu diesen allgemeinen Informationen zu dem Ergebnis-Kapitel werden in dieser Vorlage noch ein paar Informationen zu dem handwerklichen Vorgehen beim Erstellen einer schriftlichen Ausarbeitung aufgeführt.

5.4 Zitate

Grundlegend gilt: Alle Informationen, Aussagen, Bilder, Tabellen etc. die NICHT SELBER erstellt wurden, müssen über ein Zitat kenntlich gemacht werden [Kornmeier2012, 25]. Bei uns wird in aller Regel nicht wörtlich zitiert, also nicht 1-zu-1 abgeschrieben und in Hochkommata gesetzt. Vielmehr werden Informationen aus Artikeln, Webseiten, Datenblättern oder Ähnlichem

übernommen und sinngemäß wiedergegeben. Wird eine Quelle nicht deutlich gemacht, ist dies in der Regel ein Plagiat [Anderson1998, 15].

Es gibt sehr unterschiedliche Arten zu zitieren. Grundlegend wird in Geisteswissenschaften häufig in Form von Fußnoten zitiert. In der Naturwissenschaft ist ein nachgestelltes Literaturverzeichnis die Regel, so wie es auch in diesen Guidelines der Fall ist. Ein Zitat wird meist am Ende eines Satzes durch ein in eckige Klammern eingefasstes Kürzel angegeben. Dieses Kürzel kann numerisch durchnummeriert werden, IEEE Zitat Stil, oder aber Namens-Kürzel und Jahresangabe verwenden (bsp. [ENG2012]). Diese Vorlage nutzt im Wesentlichen den IEEE Zitierungs Stil.

Wenn Informationen aus dem Internet oder anderen Quellen verwendet werden, sind einige Punkte zu beachten. Nicht alles wird als zitierungswürdig betrachtet. So ist es beispielsweise unüblich Artikel aus Tageszeitungen oder Ähnliches zu zitieren. Und auch Artikel aus Wikipedia sind nicht unbedingt guter Stil. Besser ist es sich Bücher oder wissenschaftliche Artikel zu dem Thema zu besorgen und diese zu referenzieren. Über das Uni-Netz hat man Zugang zu mehreren online Quellen von wissenschaftlichen Artikeln. Wichtige Beispiele sind der IEEE Xplorer [4], SpringerLink [23] oder auch die Unibibliothek [26]. Um ein paar Beispiele an die Hand zu geben, werden an dieser Stelle eine Bachelor-, eine Masterarbeit, ein Datenblatt sowie ein Skript referenziert [LM2731, EmbSys, 17, 21].

5.5 Bilder

Wenn die Bilder nicht selber erstellt wurden, dann müssen sie auf alle Fälle zitiert werden. Schwierig hierbei ist das Copyright. Um hier Problemen aus dem Weg zu gehen, wird empfohlen, keine Bilder aus dem Internet zu verwenden. Besser und unproblematischer ist es, selber einfache Zeichnungen zu erstellen. Bilder die im Institut 4 erstellt wurden, dürfen verwendet werden, müssen aber mit Zitat (die Intranet Seite) gekennzeichnet werden.

Bilder aus wissenschaftlichen Artikeln dürfen unter anderem abgedruckt werden. Hier ist es ratsam, sich im Vorfeld eine „Reprint Permission“ vom Autor einzuholen. Viele Verlage haben eigene Vorgehen, um eine „Reprint Permissions“ zu erteilen. Hier muss man sich vorher auf alle Fälle erkundigen ob man ein verwendetes Bild wirklich verwenden darf [IEEEReprint, 22].

Auch die Platzierung von Bildern ist immer wieder missverständlich. Generell sollten diese entweder ganz oben oder ganz unten auf der Seite zu finden und mit einer Bildunterschrift versehen sein. Bei der Verwendung von Latex wird dies automatisch erfolgen. Ganz wichtig ist, dass ein Bild immer im Text erwähnt beziehungsweise erläutert werden muss. Keine Bilder aufführen, die nicht im Text Erwähnung finden. Gerade Diagramme o.ä. müssen auch erläutert werden. Ein SysML-Blockdiagramm, ein Ablaufdiagramm oder ein Zustandsdiagramm (auch wenn sie nur im Anhang gezeigt werden) müssen textuell erläutert und beschrieben werden. Bei Bildern von Robotern oder Sensoren reicht auch eine Erwähnung in einem Halbsatz.

Als Beispiel zeigt die Abbildung 5.1 das Kettenfahrzeug FTW-TV5 Rumbler welches im Institut 4 für autonome Fahrversuche verwendet wird.

Sollen zwei Abbildungen nebeneinander platziert werden, so wird eine *Subfigure* Umgebung verwendet. Auch andere Konstellationen mit mehr Bildern sind realisierbar sollen hier aber nicht



Abbildung 5.1: Autonomes Kettenfahrzeug des Instituts 4



(a) Panda

(b) Bobby Car

Abbildung 5.2: Fahrzeuge des Instituts 4.

weiter vertieft werden. Hierbei lässt sich die komplette Abbildung referenzieren 5.2 oder aber auch einzelne Unterabbildungen 5.2(a).

5.6 Auflösung

Werden eigene Diagramme und Skizzen erstellt, so ist auf eine gute Auflösung für den Druck zu achten. Die meisten Textsatzsysteme unterstützen leider keine Vektorgrafiken. Aber es ist möglich die Abbildungen erst als Vektorgrafik zu erstellen (bsp. Mittel dem Open Source Tool Inkscape[5] und dann in einer hohen Auflösung in eine Rastergrafik zu exportieren).

Auch das Erstellen von Abbildung aus Matlab heraus führt immer wieder zu Problemen. Häufigster Fehler ist eine zu niedrige Auflösung und zu kleine Schrift oder zu dünne Linien. Hierbei sollte zunächst einmal die Größe der Abbildung festgelegt werden. Dies kann unter Eigenschaften der Abbildung erfolgen, wie in Abbildung 5.3 gezeigt. Danach können die

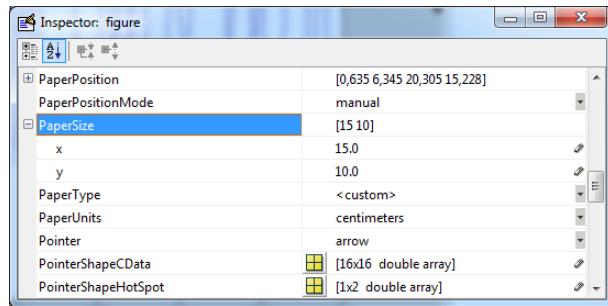


Abbildung 5.3: Einstellung der Größe für eine Matlab Abbildung.

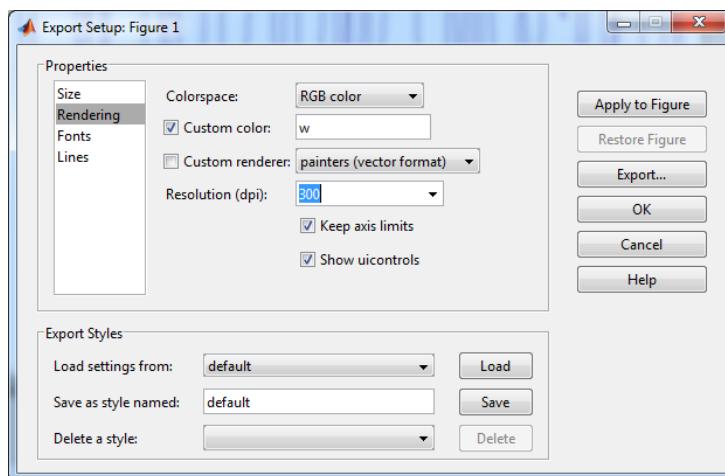


Abbildung 5.4: Einstellung der Abbildungs-Auflösung beim Exportieren aus Matlab heraus.

Schriftarten und die Liniendicken entsprechend der schriftlichen Ausarbeitung angepasst werden. Danach sollte die Abbildung nicht einfach als Bild gespeichert, sondern Exportiert werden. Hierbei kann dann die Auflösung eingestellt werden, siehe Abbildung 5.4.

5.7 Farben

Hierbei gilt: Weniger ist mehr. Nach Möglichkeit sind nur wenige oder gar keine Farben zu verwenden. Ausnahmen hierbei sind natürlich Fotografien. Für alle selber erstellten Abbildungen gilt, dass diese im Zweifel besser und professioneller wirken, wenn man diese nur in Graustufen erstellt. Ein positiver Nebeneffekt ist, dass weniger Farbseiten beim Ausdruck anfallen.

5.8 Quelltext

Hier im Institut 4 ist es die Regel, dass der Quelltext mittels dem Tool Doxygen dokumentiert wird. Insofern ist es nicht notwendig, den kompletten Quelltext im Text oder auch im Anhang

aufzuführen. Trotzdem sollten **zentrale** Abschnitte im Text erläutert werden. Beispielsweise ein komplizierterer Algorithmus oder Ähnliches muss hier erwähnt und beschrieben werden. Dies ist schließlich ein zentraler Teil der eigenen Arbeit. Dies dient zum einen der Dokumentation, da der Ablauf unter Umständen nicht aus den Dokumentierungen im Quelltext deutlich wird. Außerdem muss die Arbeit auch für Außenstehende verständlich sein. Diese haben vielleicht keinen Zugang zur beigelegten CD.

Wird Quelltext in den Text eingebunden, so erfolgt dies über ein Listing. Ein Beispiel für eine sinnlose Funktion ist im Listing 5.1 gegeben. Ein Beispiel dafür, wenn eine externe Quelldatei eingebunden wird, ist im Listing 5.2 zu finden.

```

1 void do_something(int i , int y)
2 {
3     for (int index = 0; i < 100; i++)
4     {
5         i = i + index * y;
6         if (mod(i , 2))
7         {
8             i = 1;
9         }
10        else
11        {
12            i = i *3;
13        }
14    }
15 }
```

Listing 5.1: Wichtige Funktion für das Programm

```

1 — VHDL Beispiel
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4
5 ENTITY SN74x6541 IS
6 PORT(
7     OE1a_n ,OE1b_n : IN STD_LOGIC;
8     OE2a_n ,OE2b_n : IN STD_LOGIC;
9     A1,A2:           IN STD_LOGIC_VECTOR(7 DOWNTO 0);
10    Y1,Y2:           OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
11 );
12 END SN74x6541;
13
14 ARCHITECTURE arch OF SN74x6541 IS
15 BEGIN
16     Y1 <= A1 WHEN OE1a_n = '0' AND OE1b_n='0' ELSE
17         (OTHERS=>'Z');
18     Y2 <= A1 WHEN OE2a_n = '0' AND OE2b_n='0' ELSE
19         (OTHERS=>'Z');
20 END arch;
```

Listing 5.2: Sourcecode SN74x6541

Nicht ausreichend ist es, nur den Quelltext in der Arbeit aufzunehmen. Es ist sehr wichtig,

dass auch die Funktion des Programmes erläutert wird. So kann beispielsweise die Aufteilung in mehrere Tasks über ein Sequenzendiagramm dargestellt werden. Generell bieten sich Grafiken und Diagramme hier mehr an, als reiner Text. Ganz auf einen erklärenden Text kann aber nicht verzichtet werden.

5.9 Diagramme

Bei großen Diagrammen (SysML) stellt sich häufig die Frage, ob diese im Anhang oder im Text aufgeführt werden sollen. Wird ein bestehendes Diagramm lediglich wiedergegeben, so sollte dies in den Anhang. Ist das Diagramm aber ein zentraler Punkt der eigenen Arbeit, so sollte es auch im Text auftauchen. In beiden Fällen muss der Text den Diagramm-Inhalt erläutern. Sehr große Diagramme können auch in mehrere Abbildungen aufgesplittet werden.

5.10 Layout

Auch wenn es dem eigenen Design-Empfinden besser passt, die Arbeit einseitig zu drucken: Es empfiehlt sich, einen doppelseitigen Druck zu verwenden. Die Arbeit wird nicht nach Gewicht bewertet und so kann an Druckkosten gespart werden. Auch hat man somit mehr Informationen auf einen Blick. Bei der Verwendung dieser Vorlage muss doppelseitig gedruckt werden, weil ansonsten die Seitennummerierung mal auf der linken und mal auf der rechten Seite des Blattes erscheint.

5.11 Sprachliches

Nicht jedem liegt das Schreiben von spannenden Texten. Lassen Sie den Text von jemandem lesen, der von der Thematik keine Ahnung hat. Im Zweifel eher kürzere Sätze verwenden. Nicht zu lange Schachtelsätze erzeugen. Lieber Aufzählungen verwenden.

Ein häufig wiederkehrender Punkt sind sogenannte „Weichmacher“, die in einer wissenschaftlichen Arbeit nach Möglichkeit vermieden werden: „sollen, sollte, könnte, wäre möglich, etc.“. Definitiv schreiben. Der Roboter muss folgende Aufgaben erfüllen. Dies wiederstrebt einem in der Regel, wenn man weiß, dass vielleicht noch nicht alles hundertprozentig funktioniert. Diese Weichmacher hinterlassen aber den Eindruck, dass der Autor sich seiner Arbeit nicht so sicher ist oder auch nicht davon überzeugt ist.

5.12 Abkürzungen

Ein Abkürzungsverzeichnis ist nicht verpflichtend. Aber jede eingesetzte Abkürzung, muss beim ersten Auftauchen erläutert werden. Dies kann als Fußnote, in Klammern oder auch im Text erfolgen. Als Beispiel sei hier SysML¹ aufgeführt.

¹System Modelling Language

Diagramm	Anmerkung
System Context Diagram	Gibt einen Überblick über das System und die Interaktion mit der Umwelt.
Use Case Diagram	Anwendungsfälle, also das was mit dem System gemacht werden kann.
Block Definition Diagram	Aufteilung in System und Subsysteme.
Internal Block Definition Diagram	Schnittstellen von Subsystemen zueinander.
Requirements Diagram	Anforderungen an das System.
Package Diagram	Wird hier nicht benötigt.
State Machine	Zustände die das System annehmen kann.
Allocations	Wird hier nicht benötigt.
Parametric Diagram	Wird hier nicht benötigt.
Sequence Diagram	Ablauf von Funktionen und Nachrichtenflüssen.
Activity Diagram	Abfolge von Aktivitäten im System.

Tabelle 5.1: Diagrammarten der System Modelling Language

5.13 Tabellen

Zu Tabellen gelten die gleichen Regeln wie auch für Abbildungen. Jede Tabelle muss im Text erläutert oder zumindest erwähnt werden. Auch sollte darauf geachtet werden, dass die Tabellen übersichtlich bleiben und nicht überladen werden. Als Beispiel listet Tabelle die unterschiedlichen Diagramme der *System Modelling Language* auf.

5.14 Formeln

Formeln können zum einen direkt inline im Text oder abgesetzt eingefügt werden. Wird sich für eine abgesetzte Formel entschieden, dann sind diese zu nummerieren, um sie im Text referenzieren zu können. Alle etwas komplizierteren Formeln sollten abgesetzt werden, um das Schriftbild übersichtlich zu halten. Einfache Zusammenhänge, wie beispielsweise $x_i = \cos(i - 1)$, stören das Schriftbild aber nicht weiter. Als Beispiel für eine abgesetzte Formel sei auf Formel 5.1 oder 5.2 verwiesen.

$$f(x_{k,i}, x_{k+1,i}) = \begin{cases} 1 & \text{falls } (-x_{k,i} \cdot x_{k+1,i}) > 0 \wedge |x_{k,i} - x_{k+1,i}| > \text{Schwellwert} \\ 0 & \text{sonst} \end{cases} \quad (5.1)$$

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \quad (5.2)$$

6 Diskussion

Was lässt sich aus den Ergebnissen ableiten. Sind die Anforderungen an die neue Entwicklung erfüllt? Bestehen Probleme im entwickelten Programm? Hat der Sensor Einschränkungen für die Benutzung. Hier wird nicht nur das wiedergegeben, was man selber gemacht hat, sondern versucht die Ergebnisse zu interpretieren.

6.1 Ausblick

Es hat sich bewährt, einen Ausblick zu geben, was weiterhin verbessert werden kann. Welche Änderungen an der Software sind noch notwendig, welche Hardware-Umbauten wären hilfreich etc. Das ist gerade für uns wichtig, wenn wir weiter Arbeiten für dieses Fahrzeug o.ä. anbieten.

7 Anhang

In den Anhang kommen alle entworfenen und umgesetzten Schaltpläne, Layouts etc., die wichtig für die Arbeit sind. Alle Anhänge sollten aus dem Text heraus referenziert werden. Auch hier im Anhang können Unterkapitel eingefügt werden.

7.1 Schaltpläne

Wenn Schaltpläne bzw. Layouts eingefügt werden, dann sind immer die Kenndaten mit aufzuführen. Hierzu gehören der Platinen-Name, die Platinen-Nummer, die Version, der Status und ob noch Fehler und Änderungen offen sind.

7.1 Schaltpläne

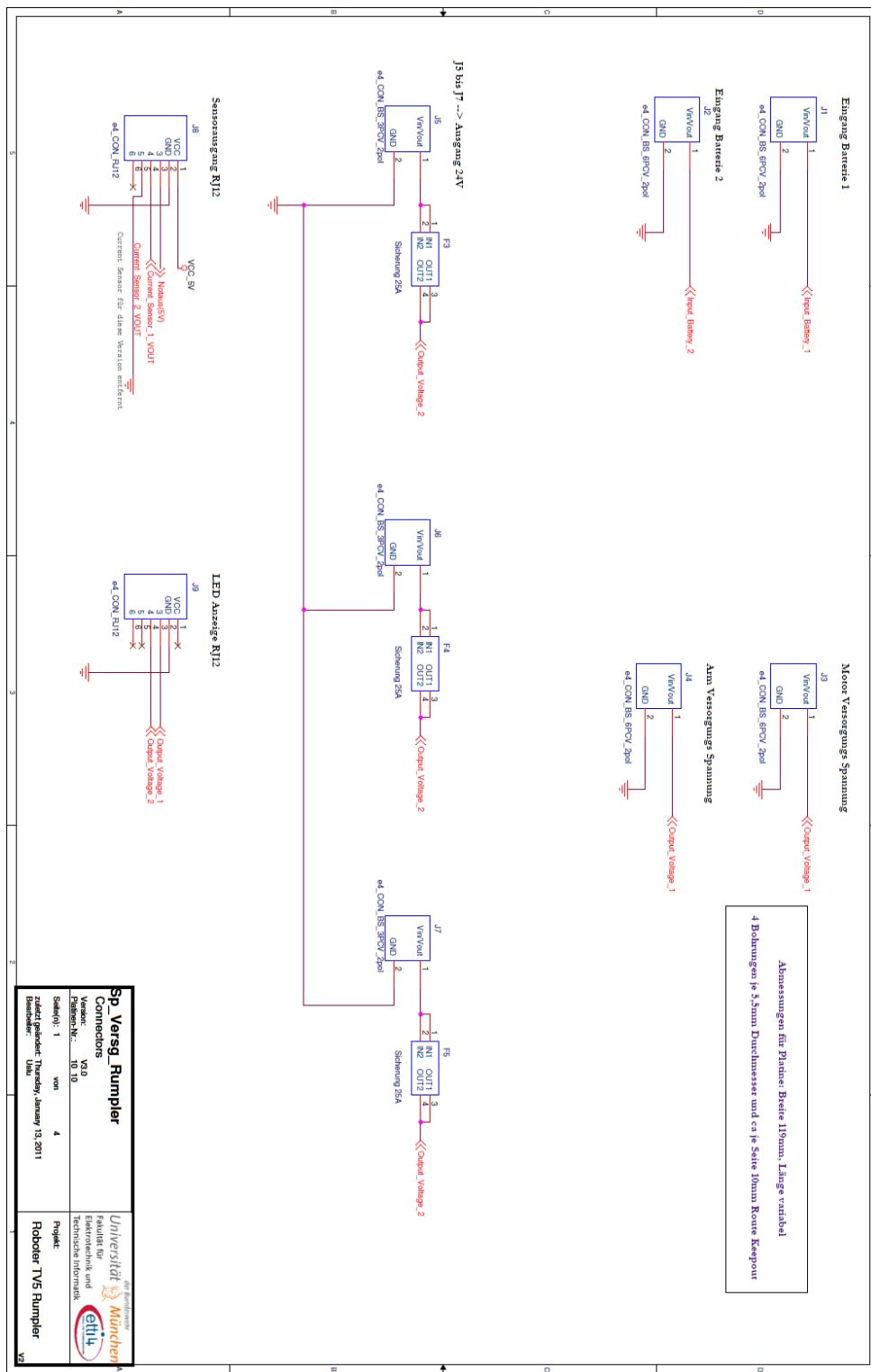


Abbildung 7.1: Schaltungsdesign für die Spannungsversorgungsplatine

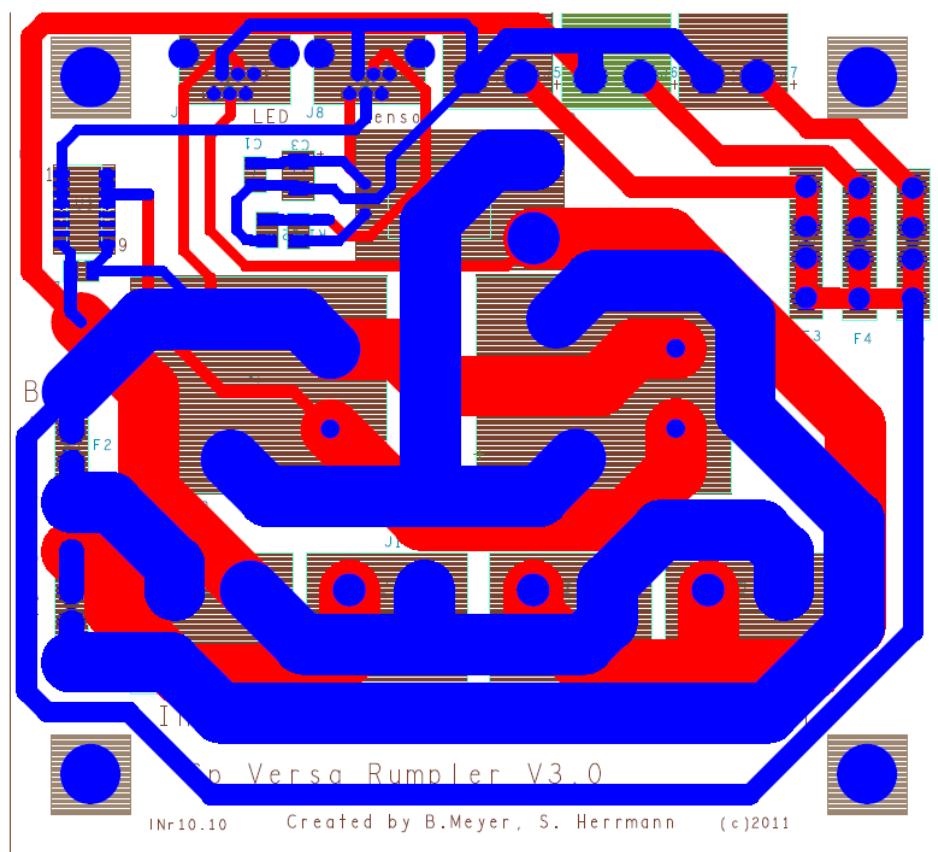


Abbildung 7.2: Layout für die Spannungsversorgungsplatine.

7.2 Projektplan mit themenbezogenem Zeitaufwand

Am Anfang der Arbeit sollte man sich Gedanken darüber machen, welche Arbeiten notwendig sind und wie viel Zeit man hierfür benötigt. Außerdem sollte eine Übersicht über die tatsächlich aufgewendeten Stunden erstellt werden. Die Planung der Arbeit kann über ein GANTT-Diagramm erfolgen. Um den Aufwand für bestimmte Tätigkeiten zu bestimmen, können die Mitarbeiter des Instituts 4 befragt werden.

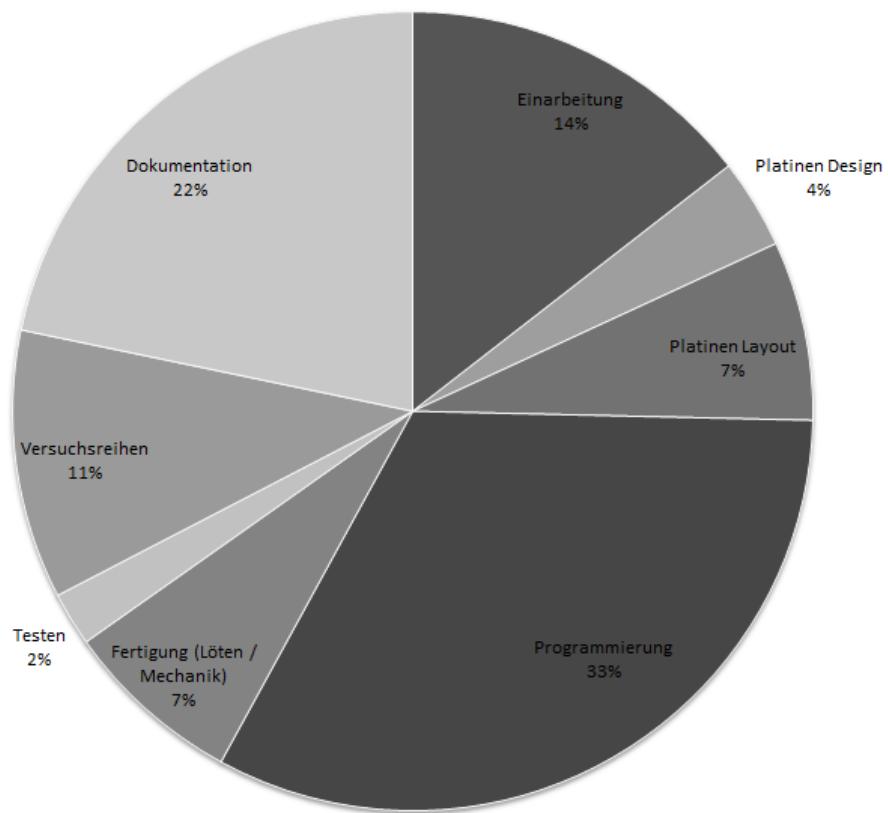


Abbildung 7.3: Themenbezogene Stundenaufschlüsselung für die Abschlussarbeit.

7 Anhang

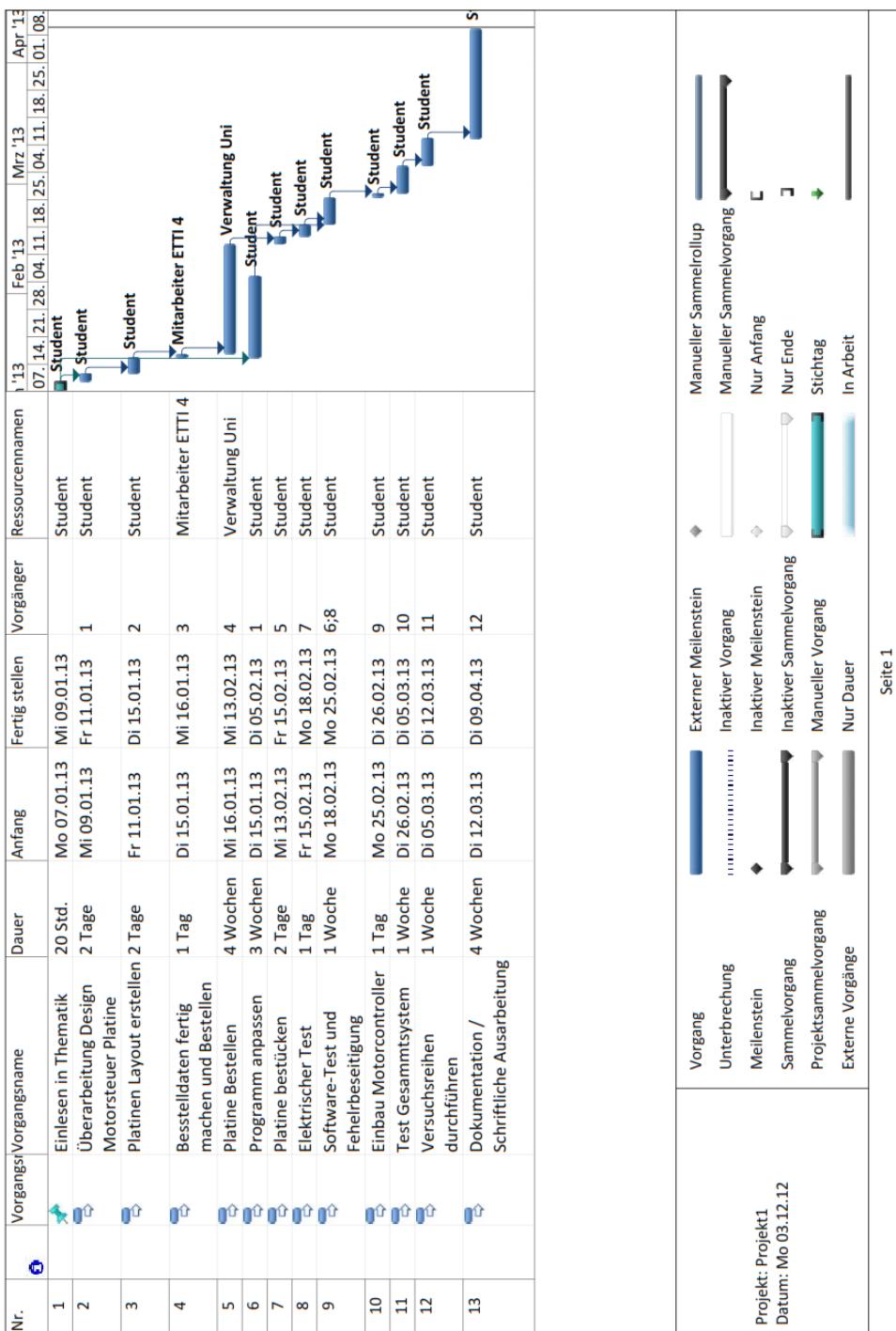


Abbildung 7.4: Aufgabenübersicht für die Abschlussarbeit.

Literaturverzeichnis

- [1] Christian Blank. „Generierung von Tiefenbildern mittels Stereoskopie“. Bachelorarbeit. Hochschule für Angewandte Wissenschaften Hamburg, 2013.
- [2] *Eclipse*. 15. Mai 2019. URL: <https://www.eclipse.org/>.
- [3] Carol Fairchild und Dr. Thomas Harman. *ROS Robotics By Example*. Packt Publishing, 2016.
- [4] *IEEE Xplorer Digital Library*. <http://ieeexplore.ieee.org/Xplore/guesthome.jsp>. Dez. 2012. URL: <http://ieeexplore.ieee.org/Xplore/guesthome.jsp>.
- [5] *Inkscape*. <http://inkscape.org/?lang=de>. Dez. 2012.
- [6] Xiaoyi Jiang und Horst Bunke. *Dreidimensionales Computersehen*. Springer, 1997.
- [7] *KEIL*. 15. Mai 2019. URL: <http://www.keil.com/>.
- [8] *Kinect*. 20. Mai 2019. URL: <https://developer.microsoft.com/de-de/windows/kinect>.
- [9] Anis Koubaa. *Robot Operating System(ROS)*. Springer International Publishing Switzerland, 2016.
- [10] Daniel Modrow. „Echtzeitfähige aktive Stereoskopie für technische und biometrische Anwendungen“. Dissertation. Technische Universität München, 2008.
- [11] *MoveIt!* 17. Mai 2019. URL: <https://moveit.ros.org/>.
- [12] *OpenNI*. 15. Mai 2019. URL: https://github.com/OpenNI/OpenNI/blob/master/Documentation/OpenNI_UserGuide.pdf.
- [13] *OpenNI Tracker*. 16. Mai 2019. URL: http://wiki.ros.org/openni_tracker.
- [14] *PrimeSense NITE Algorithms 1.5*. PrimeSense. 16. Mai 2019. URL: http://cvrlcode.ics.forth.gr/web_share/OpenNI/NITE_SDK/NITE_1.x/NITE-Algorithms.pdf.
- [15] Tim S. Robert. *Student Plagiarism in an Online Eorls: Problems and Solutions (Premier Reference Source)*. Idea Group Reference, 2007.
- [16] *Robot Operating System*. 15. Mai 2019. URL: <https://www.ros.org/about-ros/>.
- [17] Ralf Rüther. „Weiterentwicklung der Roboterplattform Rumbler (Kettenfahrzeug“. Bachelorarbeit. Universität der Bundeswehr München, 2011.
- [18] *Rviz*. 19. Mai 2019. URL: <http://wiki.ros.org/rviz>.

Literaturverzeichnis

- [19] Christoph Schmiedecke. „Tiefenbilderzeugung mit Hilfe von skalierungsinvarianten Merkmalen für ein Stereokameramodul“. Bachelorarbeit. Hochschule für Angewandte Wissenschaften Hamburg, 2009.
- [20] *SensorKinect*. 15. Mai 2019. URL: <https://github.com/avin2/SensorKinect>.
- [21] Thomas Solzbacher. „Entwicklung und Implementierung des SLAM Algorithmus (Synchronous Localization and Mapping) zur Steuerung von autonomen Robotern“. Masterarbeit. Universität der Bundeswehr München, 2011.
- [22] Springer Verlag New York / Heidelberg. *Springer Rights Platform*. <http://www.springer.com/rights?SGWID=122-0-0-0>. Juni 2012.
- [23] *SpringerLink*. <http://link.springer.com/>. Dez. 2012. URL: <http://link.springer.com/>.
- [24] *Turtle Simulator*. 17. Mai 2019. URL: <http://wiki.ros.org/turtlesim>.
- [25] Sylwia Ufnalska. *EASE Guidelines for Authors and Translators of Scientific Articles to be Published in English*. Technischer Bericht. European Association of Science Editors, 2011.
- [26] *Universität der Bundeswehr München - Universitätsbibliothek*. <http://link.springer.com/>. Dez. 2012. URL: <http://www.unibw.de/unibib/digibib>.
- [27] Christian Waldner. „Entwicklung der Steuerung eines Roboterarms mit ROS“. Bachelorarbeit. WE4 ETTI Universität der Bundeswehr München, 2018.
- [28] Christian Waldner. „Konzept für die Steuerung eines Roboterarms mit ROS“. Projektarbeit. WE4 ETTI Universität der Bundeswehr München, 2018.

Index

- Überblick, 5
- Abkürzungen, 24
- Ablaufdiagramm, 20
- Abschlussarbeit, 5
- Abstrakt, 5
- Anhang, 29
- Auflösung, 21
- Aufwand, 33
- Ausblick, 27
- Bilder, 20
- Blockdiagramm, 20
- Diagramm, 20
- Diagramme, 24
- Diskussion, 27
- Einleitung, 13
- Ergebnisse, 19
- Farben, 21
- Formeln, 25
- GANTT-Diagramm, 33
- Hardware, 18
- IEEE Xplorer, 19
- Inkscape, 21
- Layout, 24
- Matlab, 21
- Methoden, 15
- Miktex, 15
- Plagiat, 19
- Platine, 30
- Platinen-Nummer, 30
- Projektplan, 33
- Quelltext, 22
- Rastergrafik, 21
- Roboter, 13
- Rumbler, 20
- Schaltpläne, 30
- Schaltplan, 29
- Sprache, 24
- SpringerLink, 19
- SysML, 20, 24
- Tabellen, 24
- TexnicCenter, 18
- Vektorgrafik, 21
- Version, 30
- Versuchsaufbau, 18
- Zeitaufwand, 33
- Zitate, 19
- Zustandsdiagramm, 20