



Fachhochschule Nordwestschweiz
Hochschule für Technik

IANA – IMVS Autonomous Neural Assistant

Bachelorthesis

Studiengang: Informatik

Autoren: Benjamin Meyer, benjamin.meyer@students.fhnw.ch

Christian Scheller, christian.scheller@students.fhnw.ch

Betreuer: Prof. Dr. Christoph Stamm, Tel.: 056 202 78 32, christoph.stamm@fhnw.ch

Daniel Kröni, Tel.: 056 202 78 17, daniel.kroeni@fhnw.ch

Auftraggeber: Institut für Mobile und Verteilte Systeme (IMVS)

Projektdauer: 20.02.2017 bis 18.08.2017

Abstract

IANA ist ein Robotersystem mit folgenden Fähigkeiten: Sie kann kartographieren, Texte erkennen, unbekannte Personen nach dem Namen fragen, bekannte Personen begrüßen, sich autonom im Raum bewegen und gewünschte Ziele auf der erkannten Karte ansteuern.

Für das SLAM Problem wurden die ZED, die KinectOne und der RPLidar als Tiefensensoren ausprobiert und evaluiert. Mit dem RPLidar kann IANA kleinere Räume ($\sim 16.76m^2$) realitätsgetreu in wenigen Minuten kartographieren. Fensterfronten kann sie nicht erkennen, stören allerdings die Kartographie des restlichen Raumes nicht sonderlich. Als SLAM Algorithmus nutzt IANA die GMapping Komponente.

Die Textdetektion funktioniert mit den Türschilden des FHNW Campus und wurde speziell für diesen Anwendungsfall entworfen. Die Texterkennung funktioniert mit der Bibliothek Tesseract. Die erkannten Texte werden zurzeit von IANA nicht verwendet und müssten noch in die Karte an der richtigen Stelle eingetragen werden, um für die Benutzer einen Nutzen zu haben.

Die Erkennung von Personen wurde vom IP5 Projekt «IANA – IMVS Autonomous Neural Assistant» übernommen. Verbesserungen in der Performance der Gesichtsdetektion und der Robustheit der Gesichtserkennung wurden eingebaut. Unter anderem verfolgt IANA über eine Pan&Tilt-Vorrichtung detektierte Gesichter, was zu mehr Bildern für die Gesichtserkennung führt.

Für die autonome Bewegung wurden zwei Explorer implementiert. Ein Explorer basierend auf dem Zufall und ein Explorer basierend auf der Frontenerkennung in der aktuellen Karte.

Die Ansteuerung gewünschter Ziele wurde mit dem bereits existierenden Navigation Stack umgesetzt und evaluiert.

Danksagungen

An dieser Stelle wollen wir uns bei all denjenigen bedanken, die uns während der Arbeit mit Rat und Tatkraft zur Seite standen.

Der grösste Dank geht an Prof. Dr. Christoph Stamm und Daniel Kröni, beide betreuten und begutachteten unsere Bachelorthesis. Ihre Anregungen halfen uns bei manchen Herausforderungen dieser Arbeit und der gerichtete Fokus auf das Produkt und die Arbeit, schuf eine Atmosphäre der Produktivität. Dafür möchten wir uns herzlich bedanken.

Ebenfalls möchten wir Christoph Biel danken, der uns bei Fragen zur Elektrik und dem Befestigen der Hardware Komponenten immer tatenfreudig und hilfsbereit zur Seite stand.

Auch danken wir der FHNW für das gelernte Wissen in den letzten drei Jahre, dem Arbeitsplatz im Media-Lab für dieses Projekt und der Bereitstellung des Budgets für die benötigte Hardware Komponenten IANAs.

Ebenfalls möchten wir uns bei der Freiburger Universität bedanken, unter welcher Prof. Dr. Cyril Stachniss den SLAM Masterkurs unterrichtete und kostenfrei auf YouTube veröffentlichte. Auch wollen wir uns für die Bestätigung von Prof. Dr. Wolfram Burgard bedanken, dass der geringe Öffnungswinkel der KinectOne das Problem der schlechten Resultate des SLAM Algorithmus sein könnte.

Zuletzt möchten wir uns bei unseren Müttern Mariann Scheller und Dorothea Meyer bedanken, die uns einen angenehmen Arbeitsplatz in Basel zur Verfügung gestellt haben und uns bei der Textkorrektur unter die Arme griffen.

Christian Scheller, Benjamin Meyer

Basel, 20.08.2017

Inhaltsverzeichnis

1	Einleitung.....	9
1.1	Ziel der Arbeit.....	9
1.2	Anforderungen an IANA	9
1.3	Was ist neu an IANA?	9
1.4	Aufbau des Dokuments	9
2	Technologien	11
2.1	Hardware.....	11
2.1.1	Kobuki.....	11
2.1.2	Jetson TX1.....	12
2.1.3	ZED.....	12
2.1.4	KinectOne	13
2.1.5	RPLidar Development Kit.....	13
2.1.6	Pan&Tilt-Vorrichtung.....	13
2.1.7	D-Link Highspeed Kamera	13
2.1.8	Turtlebot 2 Aufbau	13
2.2	Software	14
2.2.1	Jetson Jetpack 3.1.....	14
2.2.2	Kernel Modul.....	14
2.2.3	ROS	14
2.2.4	OpenCV.....	17
2.2.5	DLib.....	17
2.2.6	OpenFace.....	17
2.2.7	Torch.....	17
3	Konzept	18
3.1	Übersicht	18
3.2	Sensoren.....	19
3.2.1	ZED Sichtbereich.....	20
3.2.2	Kobuki Odometrie	21
3.3	SLAM	22
3.3.1	Einleitung.....	22
3.3.2	Faktoren	22
3.3.3	GMapping	23
3.3.4	Grid Map.....	24
3.4	Navigation	25
3.4.1	Einleitung.....	25
3.4.2	Mögliche Technologien	25

3.4.3	Navigation Stack	25
3.5	Controller	27
3.5.1	Einleitung.....	27
3.5.2	Task.....	28
3.5.3	Task System	29
3.6	IANA Navigation	30
3.6.1	Einleitung.....	30
3.7	Collision Detector	30
3.7.1	Einleitung.....	30
3.7.2	KinectOne und ZED	31
3.7.3	RPLidar.....	31
3.8	Explorer	31
3.8.1	Einleitung.....	31
3.8.2	Random Explorer	32
3.8.3	Rational Explorer	35
3.8.4	Erweiterter Rational Explorer.....	37
3.9	Person Detection	38
3.9.1	Einleitung.....	38
3.9.2	Verbesserungen gegenüber dem IP5 «IANA» Projekt.....	38
3.9.3	Persistieren der vorgestellten Personen	39
3.10	Pan&Tilt-Vorrichtung.....	40
3.10.1	Servo Motoren	40
3.10.2	Ansteuerung	40
3.11	Face Tracker	42
3.11.1	Einleitung.....	42
3.11.2	Annahmen	42
3.11.3	Berechnung der Pan- und Tilt-Winkel	42
3.12	Benutzerschnittstelle.....	43
3.12.1	Einleitung.....	43
3.13	Texterkennung	44
3.13.1	FHNW Türschilder.....	44
3.13.2	Prinzip.....	44
3.13.3	Namens-Erkenner.....	46
3.13.4	Raumnummer-Erkenner.....	47
3.13.5	Ausrichtung eines Textes.....	48
3.13.6	Ausrichten des Roboters	48
4	Umsetzung	49

4.1	Sensoren.....	49
4.1.1	D-Link Highspeed Kamera Wrapper	49
4.2	SLAM	49
4.2.1	ROS Package Informationen.....	49
4.3	Navigation	50
4.3.1	ROS Package Informationen.....	50
4.4	Controller	52
4.4.1	Ros Package Informationen.....	52
4.4.2	ROS Messages (Commands) zu Tasks	52
4.4.3	Task Interrupt Regeln	53
4.5	IANA Navigation	53
4.5.1	ROS Package Informationen.....	53
4.6	Kobuki.....	53
4.6.1	ROS Package Informationen.....	53
4.7	Collision Detector	54
4.7.1	ROS Package Informationen.....	54
4.8	Explorer	54
4.8.1	Random Explorer.....	54
4.8.2	Rational Explorer	55
4.9	Person Detection.....	56
4.9.1	ROS Package Informationen.....	56
4.10	Face Tracker	57
4.10.1	ROS Package Informationen.....	57
4.11	Pan&Tilt-Vorrichtung.....	58
4.11.1	ROS Package Informationen.....	58
4.12	Benutzerschnittstelle.....	59
4.12.1	Technologie	59
4.12.2	Protokoll	59
4.13	Text Detection	60
4.13.1	ROS Package Informationen.....	60
4.14	IANA Starten.....	61
4.14.1	Launch Dateien.....	62
4.15	Codebase	62
5	Test und Validation	65
5.1	Tiefenscan-Sensoren Evaluation	65
5.1.1	Einleitung.....	65
5.1.2	Testszenario 1.....	65

5.1.3	Testszenario 2.....	67
5.1.4	Fazit	67
5.2	Testräume	67
5.3	SLAM	69
5.3.1	Einleitung.....	69
5.3.2	Kinect.....	70
5.3.3	RPLidar.....	71
5.3.4	Entwicklung der Karte	73
5.3.5	Vergleich vom RPLidar und der KinectOne.....	73
5.3.6	Fazit	73
5.4	Navigation	74
5.4.1	Einleitung.....	74
5.4.2	Testszenario.....	74
5.4.3	Ergebnis	75
5.4.4	Fazit	75
5.5	Explorer	75
5.5.1	Einleitung.....	75
5.5.2	Random Explorer	76
5.5.3	Rational Explorer	77
5.5.4	Fazit	77
5.6	Person Detection.....	78
5.6.1	Gesichtsdetektion.....	78
5.6.2	Fazit	78
5.7	Text Recognition.....	79
5.7.1	Testdaten.....	79
5.7.2	Text Detection	79
5.7.3	Text Recognition.....	81
5.7.4	Fazit	85
5.8	Ressourcen Verbrauch	85
5.8.1	Einleitung.....	85
5.8.2	Testszenario.....	85
5.8.3	Ergebnis	86
5.8.4	Fazit	86
6	Ergebnisse und Fazit.....	87
6.1	Source Code	87
6.2	Roboter.....	87
6.3	Roboter durch Bürolandschaft bewegen	87

6.4	Interaktion mit Personen	87
6.5	Tests	88
7	Weitere Schritte	89
7.1	Validierung der Erkennung bereits bekannter Personen über längere Zeit.....	89
7.2	Texterkennung in Karte einbauen	89
7.3	Frontier-Auswahl verbessern	89
7.4	Erkennung von Türschilder und Ausrichtung des Roboters	89
8	Anhang	91
8.1	Definition Rationalität	91
8.2	Grenzen strapazieren	91
9	Literaturverzeichnis.....	93
10	Abbildungsverzeichnis.....	98

1 Einleitung

1.1 Ziel der Arbeit

Aufbauend auf den IP5 Projekten «IANA» (Separates Dokument «IANA – IMVS Autonomous Neural Assistant») und «ARMR» (Separaten Dokument «ARMR») soll die Autonome Assistentin IANA erweitert werden: IANA soll sich autonom im Raum fortbewegen und sich diesen zugleich als zweidimensionale Karte merken. Zudem soll IANA bekannte Passanten begrüßen und unbekannte nach dem Namen fragen. Anhand der FHNW Türschilder soll IANA die erkannten Raumnummern und Namen von Mitarbeiter zuordnen. So kann IANA Kommandos ausführen, z.B. zu welcher Raumnummer oder zu welchem Arbeitsplatz sie gehen soll.

1.2 Anforderungen an IANA

Aus dem Ziel der Arbeit entstehen folgende Anforderungen an IANA:

Anforderungen	Begründung
Autonomes Erkunden	IANA muss eigenständig eine ihr unbekannte Umgebung erkunden und zugleich eine Karte (für die spätere Navigation) der Umgebung erstellen. Zudem muss IANA die Türschilder der FHNW erkennen und ablesen können, sowie diese Informationen (Nummer und Namen) in die Karte einbauen.
Navigation in einer Umgebung	Mit einer Karte der Umgebung muss IANA vorgegebene Ziele falls physikalisch möglich erreichen.
Begrüßen von Passanten	Während IANA die beiden oben aufgezählten Anforderungen realisiert, muss IANA bekannte Passanten begrüßen und unbekannte nach dem Namen fragen.

1.3 Was ist neu an IANA?

Das Erstellen einer Karte, sowie das spätere Navigieren auf einer Karte sind bekannte Probleme der Robotik mit vorhandenen Lösungen [1] [2] und Teil der aktiven Forschung. Die Texterkennung und die Gesichtserkennung sind bekannte Probleme der Bildverarbeitung oder des Machine Learnings mit vorhandenen Lösungen. Neu an IANA ist die Kombination dieser einzelnen Bauteile zum grossen Ganzen.

1.4 Aufbau des Dokuments

In diesem Dokument gibt es fünf Hauptteile: Einleitung, Technologie, Konzept, Umsetzung sowie Validation und Ergebnisse. Im Anhang befinden sich wichtige Quellen dieses Berichts.

Die Einleitung gibt eine Einführung in die Problemstellung.

Der Hauptteil Technologie gibt eine Übersicht über die Hardware IANAs und Informationen zu den verwendeten Softwares.

Im Konzept erklären wir unsere Architektur und Komponenten.

In der Umsetzung werden die im Konzept beschriebenen Komponenten und die wichtigsten Entscheidungen zu deren Umsetzung beschrieben, wie beispielsweise Parameter und deren zugehörige Werte.

Um die Verbindungen der Komponente von Konzept und Umsetzung darzustellen, haben wir ein einheitliches Farbkonzept verwendet: Jede Komponente bekommt in der Übersicht eine Farbe, welche in den Abbildungen im Dokument konsequent für diese Komponente genutzt wird.

Im letzten Teil sind Validation und Ergebnisse der von uns geleisteten Arbeit zu finden.

2 Technologien

2.1 Hardware

2.1.1 Kobuki

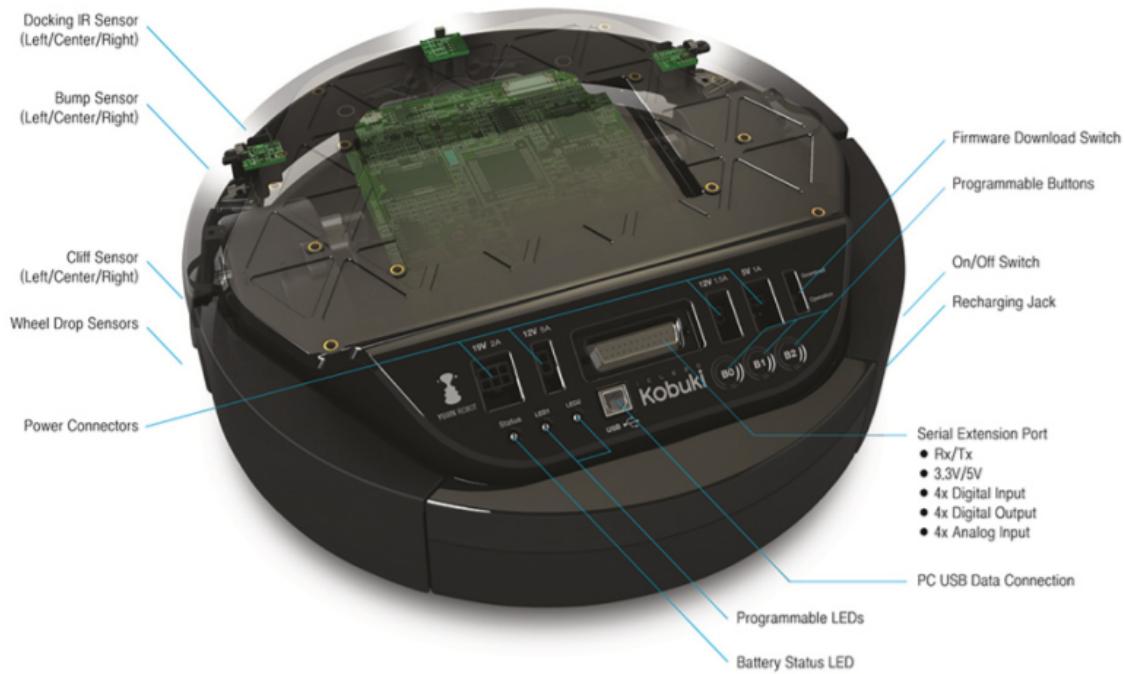


Abbildung 1: Kobuki Detailansicht

Kobuki wurde insbesondere für Ausbildungs- und Forschungszwecke entwickelt [3] und ist ein kostengünstiger 2-Rad-Roboter. Er ist IANAs Non-Holonomic Antrieb [4].

Neben einem Gyro-Sensor verfügt Kobuki über drei Bumper-Sensoren an seiner Front sowie drei Cliff- und Wheel-Drop-Sensoren zur Erkennung von Abgründen. Als Stromversorgung dient eine Batterie mit 3 (klein) oder 7 (gross) Stunden Betriebszeit. Die verwendete Batterie speist den Roboter mit Strom und kann zudem via externe Stromanschlüsse zum Betrieb weiterer Geräte genutzt werden. Zur Kommunikation mit der Steuerung von Kobuki kann neben einem USB-Anschluss auch der Serial Port verwendet werden.

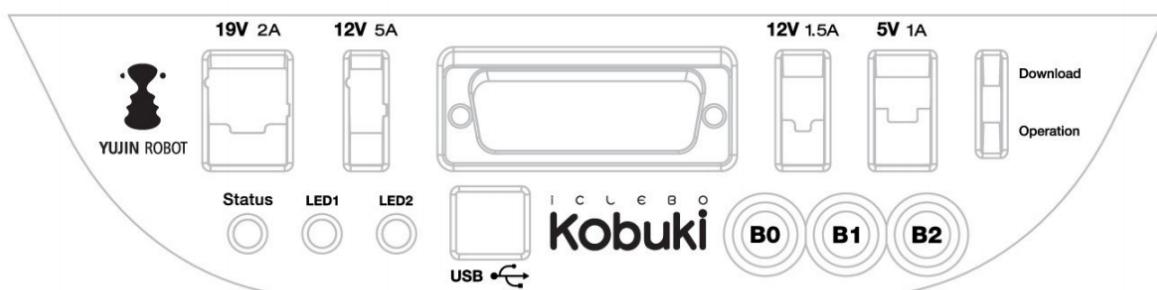


Abbildung 2: Control Panel von Kobuki mit den 4 Stromanschlüssen, dem USB- und Serial-Port

2.1.2 Jetson TX1

Das Jetson TX1 Board ist ein AI Supercomputer [5] mit den nötigen Spezifikationen, um unseren Roboter zu betreiben. Das Jetson TX1 ist die zentrale Einheit von IANA, es steuert die anderen Hardware-Komponenten und betreibt mit Ubuntu als Betriebssystem die Software Komponenten.

Die Spezifikation der Jetson TX1 [5]:

Hardware	Jetson TX1
GPU	NVIDIA Maxwell™, 256 CUDA cores
CPU	Quad ARM® A57/2 MB L2
Video	4K x 2K 30 Hz Encode (HEVC) 4K x 2K 60 Hz Decode (10-Bit Support)
Memory	4 GB 64 bit LPDDR4 25.6 GB/s
Display	2x DSI, 1x eDP 1.4 / DP 1.2 / HDMI
CSI	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.1 (1.5 Gbps/Lane)
PCIE	Gen 2 1x4 + 1x1
Data Storage	16 GB eMMC, SDIO, SATA
Other	UART, SPI, I2C, I2S, GPIOs
USB	USB 3.0 + USB 2.0
Connectivity	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
Mechanical	50 mm x 87 mm (400-Pin Compatible Board-to-Board Connector)

2.1.3 ZED

Die ZED [6] ist eine Stereokamera: Links und rechts hat sie jeweils eine RGB-Kamera und kann damit, ähnlich den menschlichen Augen, mit Algorithmen die Tiefe berechnen.

Wir haben zu Beginn des Projektes die ZED vom «ARMR» Projekt übernommen, mussten aber feststellen, dass das Tiefenbild für den Anwendungsfall SLAM ungenügend ist (Siehe 5.1 Tiefenscan-Sensoren Evaluation). Daher verwenden wir die ZED ausschliesslich als RGB Kamera, womit wir die Gesichtserkennung vom IP5-Projekt «IANA» speisen. Für die Texterkennung der FHNW Türschilder sind die aufgenommenen Bilder zu wenig detailliert, da der Text selbst von menschlichem Auge nicht entzifferbar ist (Siehe Abbildung 3). Daher hat IANA noch eine zusätzliche Kamera, die D-Link-Highspeed-Kamera.



Abbildung 3: Aufnahme eines FHNW Türschilds mit der ZED

Die ZED hat verschiedene Modi, von welchen wir den 720p Modus verwenden:

Modus	Frame Rate	Auflösung in Pixel
2.2K	15	4416 x 1242
1080p	30	3840 x 1080
720p	60	2560 x 720
WVGA	100	1344 x 376

2.1.4 KinectOne

Der Bewegungssensor KinectOne mit Tiefenbild, RGB Kamera, Mikrofon und Boxen ist schlussendlich nicht mehr Teil von IANA und wurde eine Woche vor Abschluss des Projektes durch den RPLidar Sensor ersetzt. Zuvor hatten wir die KinectOne als Tiefensensor verwendet, indem wir über das ROS Package `depthimage_to_laserscan` [7] das KinectOne Tiefenbild zu einem Laserscan transformierten.

Unsere SLAM Validation zeigte aber, dass der Algorithmus zwar mit der KinectOne funktionieren kann, allerdings nicht in ausreichender Qualität (5.1 Tiefenscan-Sensoren Evaluation, 5.3.2 Kinect)

Die KinectOne verfügt zudem über eine RGB Kamera, für welche wir nun die ZED verwenden und ein Mikrofon, welches wir nicht brauchten.

2.1.5 RPLidar Development Kit

Der RPLidar [8] ist ein kostengünstiger LIDAR (Light Detection and Ranging) 360° Laserscan Sensor. Die erhöhte Genauigkeit und die Sichtweite von 360° hilft dem SLAM Algorithmus in der Proposal Distribution die möglichen Positionen des Roboters stärker einzuschränken, da es auch den neuesten Scan mitbeachtet (Siehe 3.3.3 GMapping).

2.1.6 Pan&Tilt-Vorrichtung

IANA verfügt über eine Lynxmotion Large Pan&Tilt-Vorrichtung [9], welche auf der Top-Plattform montiert ist. An der Pan&Tilt-Vorrichtung ist die ZED angebracht. Dadurch kann sich das Sichtfeld der ZED horizontal (ohne, dass der Roboter sich drehen muss) und auch vertikal verändern. Die Ausrichtung der Vorrichtung geschieht über zwei leistungsstarken Servo-Motoren [REFERENZ!], die über PWM Signale von einem separaten Arduino angesteuert werden. Der Arduino erhält die gewünschte Ausrichtung über USB von der ROS Node `iana_camera_pan_tilt`, welche auf dem Jetson TX1 läuft.

2.1.7 D-Link Highspeed Kamera

Das Bild der ZED ist nicht detailliert genug, um den Text der FHNW Türschilder zu erkennen (Siehe 2.1.3 ZED). Deshalb verwenden wir die D-Link Highspeed DCS-3710. Kamera mit dem 5,0 - 50 mm CS-Mount Computar Objektiv [10]. So sind detaillierte Bildaufnahmen der FHNW Türschilder möglich (Siehe Abbildung 4).

2.1.8 Turtlebot 2 Aufbau

Der Turtlebot 2 Aufbau [11] dient als Plattform zum Anbringen von Hardware-Komponenten auf den Kobuki Roboter. Auf insgesamt drei Ebenen können Sensoren, Notebooks und Boards nach Wunsch angebracht werden.

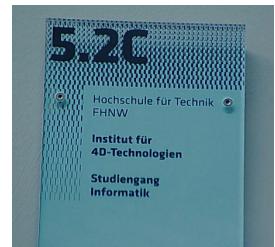


Abbildung 4:
Aufnahme eines
FHNW Türschildes mit
der D-Link Highspeed
Kamera mit Objektiv.

2.2 Software

2.2.1 Jetson Jetpack 3.1

Jetson Jetpack (Jetson SDK) [12] ist ein für die „NVIDIA Jetson Embedded Platform“ entwickeltes Softwarepaket, welches neben den benötigten Treibern eine speziell für Robotik und Computer Vision ausgelegte Palette an Programmen und Bibliotheken beinhaltet. In der Komplettinstallation enthalten sind Ubuntu 16.04 (L4T 28.1 Kernel), TensorRT 2.1, cuDNN 6.0, VisionWorks 1.6, CUDA 8.0 und die Jetson Multimedia API. Auf IANAs Jetson Board ist die komplette Version installiert.

2.2.1.1 *Ubuntu 16.04 (L4T 28.1 Kernel)*

Ubuntu 16.04 ist das zum Zeitpunkt dieser Arbeit aktuellste Linux System mit Langzeitsupport, der L4T 28.1 Kernel [13] der aktuellste Release der Tegra-Kernel Familie. Ubuntu 16.04 wird von ROS Kinetic empfohlen [14].

2.2.1.2 *TensorRT 2.1*

NVIDIA TensorRT [15] ist eine High-Performance Deep Learning Inferenzen Optimierungs- und Laufzeitumgebung, welche speziell für NVIDIAAs GPUs entwickelt wurde. Sie ist darauf ausgelegt, mittels vor trainierter Neuronaler Netzwerke und GPU-Unterstützung schnelle Inferenzen zu liefern.

2.2.1.3 *cuDNN 6.0*

NVIDIA CUDA Deep Neural Network Library (kurz: cuDNN) [16] ist eine GPU beschleunigte Software Bibliothek, welche Grundbausteine für Deep Neural Networks zur Verfügung stellt.

2.2.1.4 *VisionWorks 1.6*

NVIDIA VisionWorks [17] ist eine GPU beschleunigte Computer Vision und Bildverarbeitungs-API. Sie ist eine erweiterte CUDA Implementierung des OpenVX Standards [18].

2.2.1.5 *CUDA 8.0*

Das NVIDIA CUDA Toolkit bietet eine Entwicklungsumgebung für die Entwicklung GPU beschleunigter Software.

2.2.1.6 *Jetson Multimedia API*

Die Jetson Multimedia API bietet eine Sammlung von Low-Level APIs zur Steuerung von Kameras und Videobearbeitung.

2.2.2 Kernel Modul

Einige Hardware-Komponenten von IANA benötigen die Installation zusätzlicher Kernel Module.

2.2.2.1 *CP210x Treiber*

Der RPLidar ist via einer USB to UART Bridge von Silicon Labs mit dem Jetson Board verbunden. Der zur Ansteuerung benötigte CP210x Treiber ist im L4T Source Tree enthalten, wird jedoch standardmässig nicht gebaut.

2.2.2.2 *CDC-ACM Treiber*

Der CDC-ACM Treiber muss installiert werden, damit das Arduino Boards korrekt erkannt wird. Auch dieser Treiber ist im L4T Source Tree enthalten, wird jedoch standardmässig nicht gebaut.

2.2.3 ROS

ROS steht für Robotic Operating System hat aber mit einem klassischen Betriebssystem nichts zu tun. ROS ist eine Robotic-Middleware [19], die durch eine Nodes-Architektur, eine klar definierte Kommunikationsschnittstelle über TCP [20], Tools zur Unterstützung [21] [22], Standardlösungen für häufige Robotics-Probleme [23] [24] und eine aktive Community [25] weit verbreitet ist [26] [27] sowie

Komponenten leicht austauschbar und integrierbar macht. Beispielsweise konnten wir für unser Projekt die Komponente GMapping (Siehe 3.3.3 GMapping) verwenden.

Für IANA verwenden wir die zu Projektstart neuste ROS Version, ROS Kinetic.

In diesem Abschnitt werden die wichtigsten ROS Konzepte und Schlüsselworte beschrieben.

2.2.3.1 Catkin

Catkin [21] ist ein das offizielle Build Tool von ROS und baut seine zusätzlichen Funktionen auf dem weit verbreiteten CMake auf.

Für die Organisation von Nodes, Messages, Action und Services dienen ROS Packages. Für das Organisieren von Packages und um alle zusammenhängenden Packages auf einmal mit Catkin bauen zu können, dienen dem Entwickler Workspaces (Siehe Abbildung 5). Die für den Build-Prozess benötigten Abhängigkeiten sind jeweils pro Package in einer CMakeLists.txt und einer package.xml Datei angegeben.

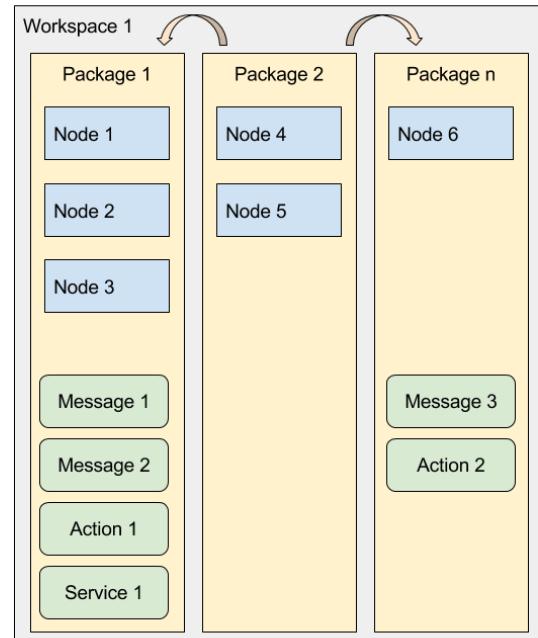


Abbildung 5: Catkin Workspaces

2.2.3.2 roslaunch und rosrun

Für die Laufzeit spielen Workspaces keine Rolle mehr und Packages gelten den ROS Nodes einzlig als Kontext. Gestartet werden einzelne ROS Nodes über den «rosrun»-Befehl.

Da ein ROS System meist aus vielen Nodes besteht und man nicht alle von Hand starten möchte, kann man eine Gruppe von Nodes mittels einer Launch-Datei (*.launch) gemeinsam starten. Eine Launch-Datei wird mit dem Befehl «roslaunch» gestartet.

2.2.3.3 Architektur

ROS besteht aus einer Graphen Architektur: Einzelne Nodes [28] laufen als eigene Prozesse und kommunizieren mittels TCP Verbindungen [20] miteinander. Dank TCP als Kommunikationsmittel können ROS Nodes eines Systems ohne Weiteres auch auf verschiedenen Maschinen im gleichen Netzwerk laufen (Siehe Abbildung 6).

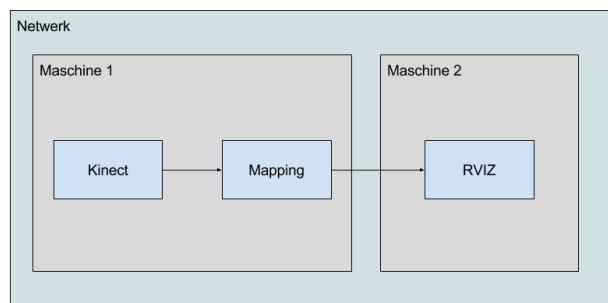


Abbildung 6: Kommunikation zwischen verteilten ROS Nodes über das Netzwerk

Grundlegend für die Architektur ist der pro ROS System einmalige ROS-Core [29], welcher aus einem ROS Master mit Parameterserver und einer Logging-Node besteht. Diese bilden zusammen die Voraussetzung eines ROS Systems.

2.2.3.4 ROS Master

Beim ROS Master melden sich neue Nodes mit Namen an und allen Nodes werden vom ROS Master zur Laufzeit die jeweiligen Zuhörer und jeweiligen Sender mitgeteilt – die tatsächliche Kommunikation läuft dann Peer-to-Peer zwischen den einzelnen Nodes ab.

Der ROS Master wird über eine XMLRPC API [30] angesteuert, welches die rospy Library hinter den Kulissen verwendet. Während der Entwicklung von IANA hatten wir nie direkt mit der XMLRPC API zu tun.

2.2.3.5 Parameterserver

Der Parameterserver [31] speichert zentral die Konfiguration des ROS Systems. ROS Nodes können zur Laufzeit über Parameternamen auf Werte zugreifen oder diese verändern. Zu jeder Zeit kann so an einem Ort die Konfiguration systemweit überprüft und geändert werden. Der Parameterserver läuft als Teil des ROS Masters.

2.2.3.6 Logging-Node

In ROS gibt es eine zentrale Node für Logging [32]. Andere Nodes loggen, in dem sie an die Topic «/rosout» der Logging-Node eine Log Message senden.

2.2.3.7 Nodes

Nodes sind eigene Prozesse und sind üblicherweise in C++ oder Python geschrieben. Eine Node kann als Blackbox angesehen werden, die meistens mit Messages (Typen) auf bestimmte Topics (Kanäle) hört, dadurch Daten anderer Nodes erhält, diese weiterverarbeitet und dann mit Messages auf anderen Topics veröffentlicht. Eine Node ist aber nichts anderes als ein Prozess und kann daher auch Hardware Treiber ansteuern, einen Webserver bereitstellen, Daten persistieren, usw.

Für komplexere Systeme können Nodes auch Services und Actions anbieten, die von anderen Nodes genutzt werden können.

2.2.3.8 Kommunikation

Die ROS Kommunikation läuft über TCP. Es gibt zwei Kern-Kommunikationsmittel, die im Protokoll verankert sind: Messages und Services. Zudem gibt es ein auf Message aufbauendes Kommunikationsmittel: Actions.

2.2.3.8.1 Topic / Message

Die einfachste Kommunikation von Nodes in einem ROS System sind Topics mit Messages [33]. Ein Topic ist ein benannter Kanal, unter welchem Messages veröffentlicht oder empfangen werden. Eine Message hat einen durch eine .msg Datei klar definierten Typ und kann aus Grundtypen [34] und anderen Messages bestehen.

Eine Node informiert den ROS Master auf welchem Topic (unter welchem Namen), es welchen Message veröffentlichen will (Publisher) oder auf welchem Topic es welche Message empfangen möchte (Subscriber). Dabei können mehrere Nodes auf das gleiche Topic veröffentlichen und zuhören.

2.2.3.8.2 Service

Ein Service folgt dem Request-Response Prinzip [35]. Eine Node (Server) bietet einen Service an, wo ein typisierter Request von jeder Node (Client) eingehen kann. Der Server antwortet dem Client mit einer typisierten Response.

Services werden auf dem ROS Master registriert und abgefragt, der Client ruft Services über eine ServiceProxy Klasse [36] auf.

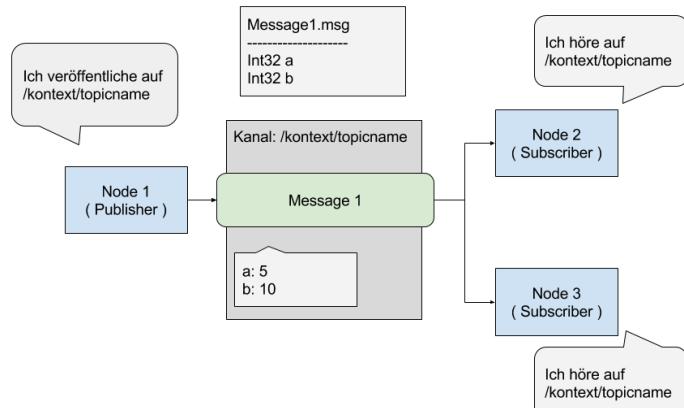


Abbildung 7: Ein Beispiel einer ROS Kommunikation über eine Message.

2.2.3.8.3 Action

Eine Action [37] hat ebenfalls einen Server (Anbieter) und einen Client (Benutzer). Der Client schickt ein Ziel (Goal) an den Server. Der Server kann an den Client Zwischenresultate (Feedbacks) schicken und endet mit einem Resultat (Result).

Actions sollten verwendet werden, wenn ein Resultat erst nach einer zeitlichen Dauer erreicht werden kann, beispielsweise wenn ein physischer Roboter an ein bestimmtes Ziel steuert. Wird eine Action sofort ausgeführt, ist es meist klüger einen Service zu verwenden.

Eine Action kann auch von beiden Seiten abgebrochen werden und hat beim Client und beim Server einen unterschiedlichen Status [38].

2.2.3.9 Standardlösungen

ROS bietet für verschiedene Probleme der Robotik Standardlösungen an (z.B. TF). Auch Actions kann man als eine Standardlösung für eine Kommunikation für Ziele ansehen.

2.2.3.9.1 TF

2.2.3.9.1.1 Problem

Bei einem Roboter haben wir unterschiedliche Koordinaten-Frames. Beispielsweise sind die Sensordaten eines Laserscanners relativ zu dessen physischer Position und die Position des Laserscanners relativ zur Basis des Roboters. Um über die ganze Zeit des Prozesses die Übersicht über all diese Frames zu behalten, muss man die Beziehungen dieser Frames zu jedem Zeitpunkt abbilden können.

2.2.3.9.1.2 Lösung

ROS bietet das Paket TF2 an (TF1 ist veraltet). Mit TF2 können die Beziehungen von den verschiedenen Koordinaten Frames des Roboters als Baumstruktur beschreiben werden. Diese Beziehungen werden von der jeweiligen Node über das /tf Topic veröffentlicht, und so kann zu jedem Zeitpunkt eine realistische Transformation zwischen zwei Koordinaten Frames im gleichen Baum berechnet werden. Dafür bieten die ROS Bibliotheken wie rosmsg oder roscpp Hilfsfunktionen an.

2.2.4 OpenCV

Für verschiedenste Tasks verwenden wir OpenCV (Open Source Computer Vision Library) [39]. OpenCV ist eine optimierte Bildverarbeitungs-Bibliothek.

2.2.5 DLib

Von DLib [40] verwenden wir die Landmarks Detection für die Gesichtserkennung. DLib ist ein C++ Toolkit mit Fokus auf Machine Learning Algorithmen. Es existiert auch eine Python Anbindung für DLib, welche im Hintergrund die Native DLib Funktionen verwendet.

2.2.6 OpenFace

Bei der Gesichtserkennung verwenden wir OpenFace [41]. OpenFace ist eine Torch Implementierung des FaceNet Papers [42].

2.2.7 Torch

Torch [43] ist ein Machine Learning Framework mit GPU Support.

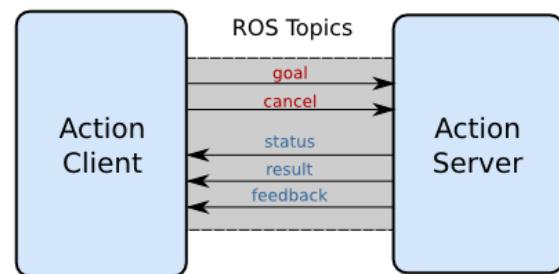


Abbildung 8: Kommunikation zwischen Action Client und Action Server (Quelle: [38])

3 Konzept

3.1 Übersicht

IANA ist ein Roboter, der versucht rational (Siehe 8.1 Definition Rationalität) die erhaltenen Befehle auszuführen. IANAs Umgebung ist mit ihren Sensoren teilweise beobachtbar und durch die herumlaufenden Menschen und verstellbaren Möbel leicht dynamisch meistens aber statisch.

Für die Aufgabenbereiche IANAs wurden verschiedene Komponenten neu entworfen oder vorhandene genutzt (grau dargestellt). Alle Komponenten zusammen bilden schliesslich IANA (Siehe Abbildung 9).

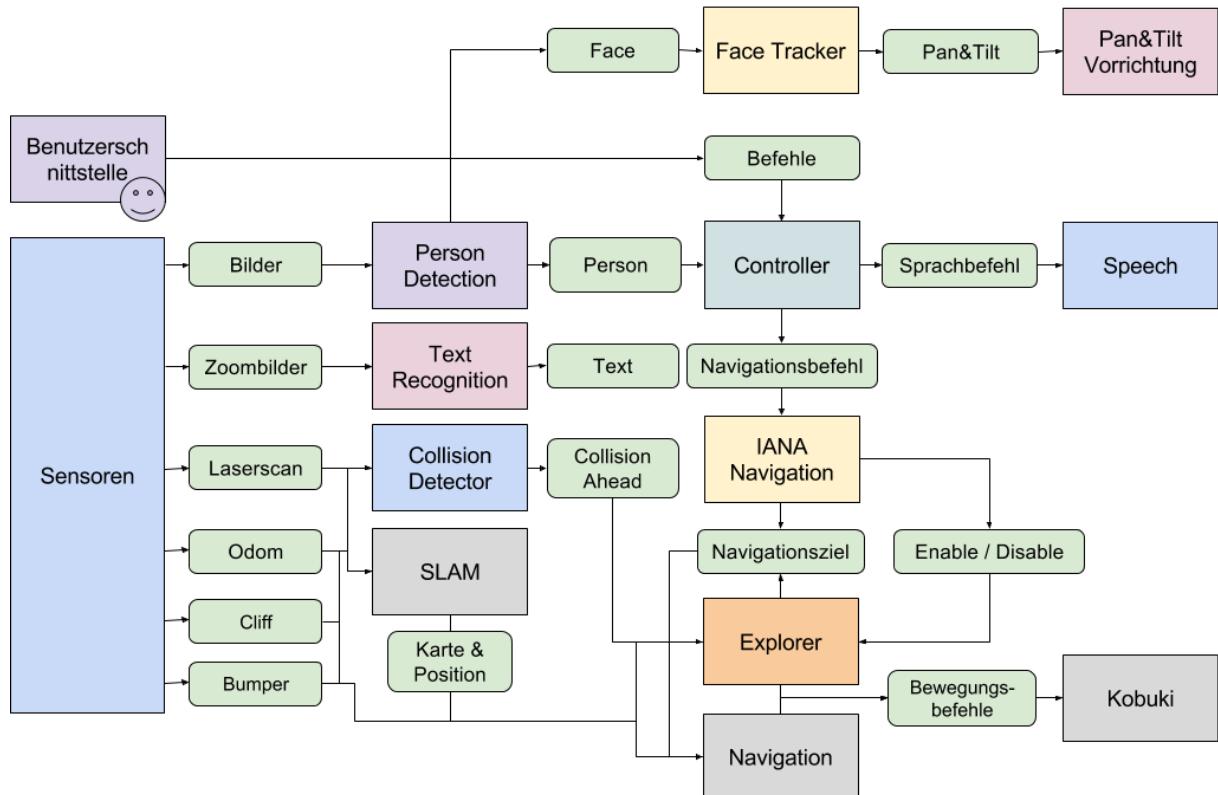


Abbildung 9: IANAs Komponenten und deren Interaktion

Name	Beschreibung
Sensoren	IANAs Sensoren liefern Daten wie Bumper-Events, Odometrie, Bilder oder einen Laserscan, welche von weiteren Komponenten verwendet werden können.
SLAM	Die SLAM (Simultaneous localization and mapping) Komponente erstellt über die Zeit eine Karte der zu erkundenden Umgebung und erkennt gleichzeitig die Position IANAs auf dieser Karte.
Navigation	Die Navigationskomponente nimmt ein Ziel, die aktuelle Karte, Odometrie Informationen und den momentanen Laserscan entgegen. Damit berechnet sie die nächsten Bewegungsbefehle für den Roboter.

Controller	Die Controller-Komponente steuert die ankommenden Befehle des Benutzers sowie Personen-Events. Die Verarbeitung der Befehle sollen IANA intelligent wirken zu lassen.
IANA Navigation	IANA Navigation dient als einheitliche Schnittstelle für alle Navigationsbefehle, welche die darunterliegenden Komponenten abstrahiert. Sie nimmt Navigationsbefehle entgegen und führt diese entweder über die Navigation- oder die Explorer-Komponente aus.
Kobuki	Kobuki setzt die Bewegungsbefehle in die Realität um.
Collision Detector	Die Collision Detector Komponente ermittelt aus den Laserscandaten, wie weit das nächste Hindernis entfernt ist und meldet dies an eine ROS-Topic.
Explorer	Der Explorer ist aktiv wenn IANA den aktuellen Raum erkundet. Er hat die Aufgabe, die Zeit, die IANA zur Erkundung zur Verfügung hat bestmöglich zu nutzen.
Person Detection	Die Personenerkennung nimmt ein Bild entgegen und erkennt über die Zeit, welche Personen in IANAs Blickfeld eintreten und welche Personen das Blickfeld verlassen.
Face Tracker	Der Face Tracker steuert über die Pan&Tilt-Vorrichtung die ZED Kamera und folgt entdeckten Gesichtern. Dies hilft IANA, Personen besser zu erkennen und lässt IANA menschlicher wirken.
Pan&Tilt-Vorrichtung	Die Pan&Tilt-Vorrichtung kann über ROS-Topics gesteuert werden, um den Blickwinkel der ZED zu verändern.
Benutzerschnittstelle	Die Benutzerschnittstelle dient dazu, als Benutzer mit IANA zu kommunizieren. Man kann ihr Befehle geben, welche dann die Controller Komponente zu Tasks übersetzt.
Text Recognition	Die Texterkennung erkennt den Text auf den FHNW Türschildern. Die erkannten Texte sind Raumnummern und Namen der Institutsmitarbeiter sowie der Institute und müssen in die Karte eingebaut werden.
Speech	Die Speech Komponente ist für die Ausgabe von Text verantwortlich. Sie ist eine zentrale Stelle, so kann z.B. alles was IANA sagt, ohne grossen Aufwand geloggt werden.

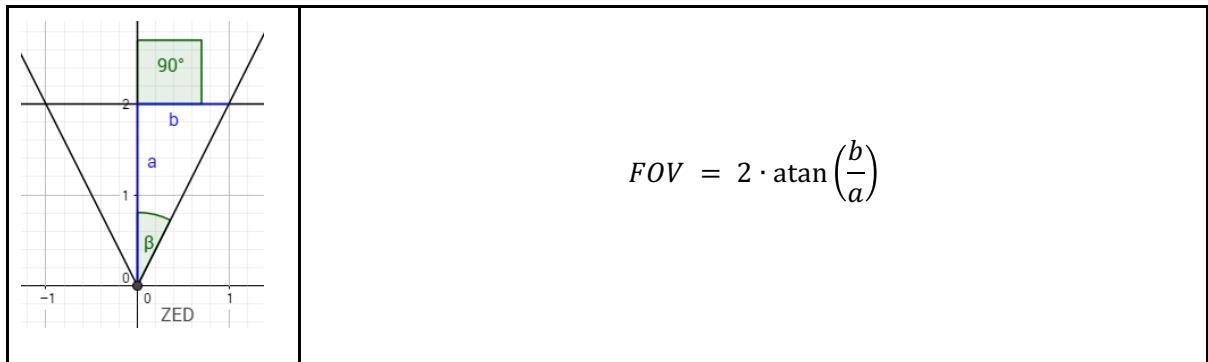
3.2 Sensoren

Die Sensoren geben Rohdaten weiter, auf welchen weitere Komponenten Operationen ausführen werden. IANA hat folgende Sensoren und Rohdaten für folgende Zwecke:

Sensor	Rohdaten	Zweck
ZED Kamera	<u>Bilder</u> HD Bilder der Kamera mit der Auflösung 1280 x 720.	Gesichtserkennung Personenerkennung
D-Link Highspeed Kamera	<u>Zoombilder</u> HD Bilder der Kamera mit einem Zoom-Objektiv mit Auflösung 1280 x 1024.	Texterkennung
Kobuki Sensoren	<u>Cliff, Wheel-Drop</u> Zeigt an, ob der Roboter an einem Abhang steht.	Navigation
	<u>Bumper</u> Zeigt an, ob der Roboter gegen etwas gefahren ist.	Navigation
	<u>Odom</u> Gibt die Odometrie des Roboters über die Räder und den Gyro-Sensor an.	Navigation, SLAM
RPLidar	<u>Laserscan</u> Gibt Tiefendaten der Umgebung in 360 Grad.	SLAM

3.2.1 ZED Sichtbereich

Den Sichtbereich der ZED haben wir mittels Testaufnahmen mit 2 Meter entfernten waagrecht und vertikal angebrachten Messstangen ermittelt. Da die Messstangen senkrecht zur Winkelhalbierenden von ZEDs Sichtbereich angebracht sind, kann mittels einfachen trigonometrischen Berechnungen der genaue Sichtbereich ermittelt werden:

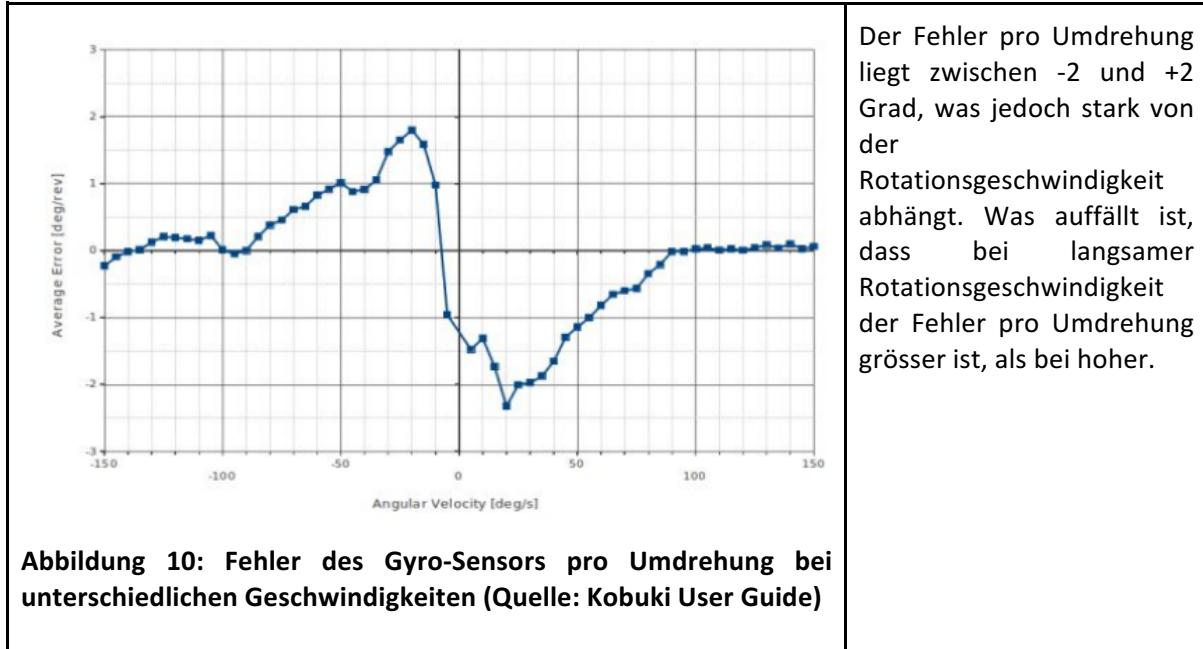


Mit diesem Vorgehen konnten wir folgenden Sichtbereich eruieren:

Horizontal	110 Grad (+- 1 Grad)
Vertikal	54 Grad (+- 1 Grad)

3.2.2 Kobuki Odometrie

Kobuki ist mit einem 3-Achsen Gyro-Sensor ausgestattet, welcher ab Werk kalibriert ist. Im Kobuki User Guide [44] sind die Performance-Tests, wie genau der Gyro-Sensors misst, dokumentiert:



3.2.2.1 Rotation an Ort und Stelle

Zur Bestimmung der Genauigkeit des Gyro-Sensors führten wir selbst nochmals Messungen durch. IANA drehte sich mit der Geschwindigkeit von 30 Grad in der Sekunde fünf Mal an Ort und Stelle. Wir massen die Differenz zwischen der Ausrichtung, an der sich IANA schätzte und der tatsächlichen Messung (Ground Truth).

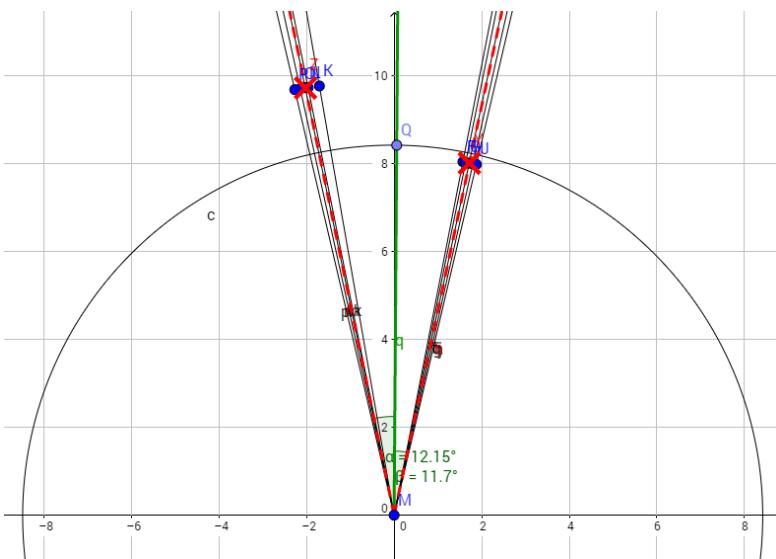


Abbildung 11: Messergebnisse des Gyro-Fehlers nach 5 Umdrehungen

Bei unseren Tests lag die Abweichung bei etwas mehr als 2 Grad pro Umdrehung. Dies ist ein nur leicht höherer Wert, als der im Kobuki User Guides angegebenen Wert für die gleiche Drehgeschwindigkeit.

Da GMapping von ungenauen Odometriedaten ausgeht (Siehe 3.3.3) und diese mit den Laserscan Daten verbessert, sollte die Gyro-Ungenauigkeit kein Problem darstellen. Unsere SLAM-Tests (Siehe 5.3) bestätigen diese Annahme.

3.3 SLAM

3.3.1 Einleitung

SLAM steht für Simultaneous Localization and Mapping und ist ein bekanntes Problem in der Robotik. Der Roboter weiss bei einer zu Beginn unbekannten Umgebung nicht, wo er ist und wie seine Umgebung aussieht. Simultan muss nun herausgefunden werden, wie die Umgebung aussieht (Mapping) und wo der Roboter sich in dieser Umgebung befindet (Localization). Die Schwierigkeit des Prozesses liegt in der hohen Korrelation dieser Informationen: Wissen wir, wo der Roboter ist, wird das Mapping einfacher (Occupancy grid mapping [45]). Kennen wir die Umgebung, wird die Localization einfacher [46].

3.3.2 Faktoren

Es gibt verschiedene Faktoren, die einen SLAM Algorithmus ausmachen:

- Umgebungsstruktur
 - Beispiel: Befinden wir uns in einem Gebäude (Indoor) oder befinden wir uns in einem Park, einem Wald (Outdoor)?
- Full vs Online (Siehe Introduction to Robot Mapping [47], Folie 19)
 - Full: Es wird der gesamte Pfad des Roboters geschätzt.
 - Online: Es wird immer nur die aktuelle Pose des Roboters geschätzt.
- Karte (Siehe Grid Maps [48], Folie 2)
 - Feature Map: Die Karte besteht aus erkannten und relative zueinanderstehenden Landmarks. Dafür wird ein zur Umgebung passenden Landmark Detektor benötigt.
 - Volumetric Map: Gibt an, welche Stellen einer Karte besetzt, frei oder unbekannt sind. Beispielsweise eine 2D Karte der Umgebung, die mit einem Grid diskretisiert wurde.

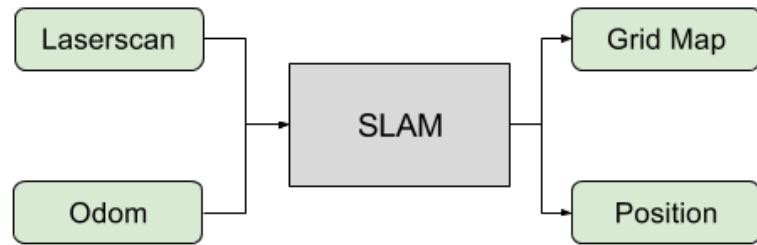
- Sensorqualität
 - Für einen SLAM Algorithmus sind genauere Sensoren immer besser, da er so die Unsicherheit verringern kann – je mehr und je genauer der Roboter sieht, desto einfacher.
 - SLAM Algorithmen nutzen verschiedene Eigenschaften von Sensoren au
 - GMapping (Siehe 3.3.3) nutzt Laserscan Präzision aus um die Proposal Distribution zu verbessern.
 - Hector SLAM [49] erwartet eine hohe Frequenz des Laserscans und braucht die Odometrie Information des Roboters nicht, um zu funktionieren.

Für unseren Anwendungsfall benötigen wir einen Indoor, Online SLAM Algorithmus mit einer Volumetric Map als Output. Volumetric Maps helfen der Navigation, da sie genau beschreiben, wo der Roboter durchfahren kann.

Als SLAM Algorithmus verwenden wir GMapping. Der Algorithmus arbeitet Indoor, ist ein Online SLAM Algorithmus und erstellt eine 2D Grid Map. Zudem gibt es im Internet viele Quellen dazu. Er ist damit ideal für unseren Anwendungsfall. Mit einem besseren Laserscanner könnte Hector SLAM in Betracht gezogen und verglichen werden.

3.3.3 GMapping

GMapping [50] nimmt die Laserscan-Daten und die Odometrie des Roboters entgegen und akkumuliert diese über Zeit zu einer 2D Grid Map und lokaliert gleichzeitig den Roboter auf dieser Karte.



An der Universität von Freiburg gibt es einen Masterkurs über SLAM, in welchem die nötigen

Abbildung 12: Interaktion der SLAM Komponente (GMapping) mit der ROS-Ausenwelt

Vorkenntnisse und GMapping unterrichtet werden [51]. Dieser Kurs war unsere Hauptquelle für das Verständnis des Algorithmus.

GMapping nutzt dafür einen Rao-Blackwellized Particle Filter und eine Proposal Distribution, die die letzte Beobachtung (den letzten Laserscan) ebenfalls berücksichtigt, um die Odometrie-Unsicherheit zu verringern und somit die Anzahl Partikels ohne Verlust zu verkleinern. Dadurch wird die Performance ins plausible erhöht. Wie diese genau funktionieren ist hier beschrieben, doch zuerst werden Kernbegriffe erklärt.

3.3.3.1 Particle Filter

Particle Filter ist ein generelles Prinzip und hat verschiedene Einsatzgebiete (z.B. die Monte Carlo Localization [46]), wenn es um das Schätzen von verborgenen Zustände über ungenaue Messungen in einem stetigen, nichtlinearen Raum geht [52]. Die Idee eines Particle Filters ist eine non-parametric Verteilung mit Partikeln (Beispielen) anzunähern. Je mehr Partikel wir haben, desto flächendeckender können wir die tatsächliche Verteilung annähern.

Der Prozess besteht aus zwei sequenziellen Schritten:

1. Führe ein Importance Sampling Principle durch:

- a. Definiere eine Proposal Distribution (z.B. eine Gaussian) und ziehe k Particles heraus:
 $x_t \sim \pi(x_t | \dots)$
 - b. Gewichte die Particle anhand der Target Distribution: $target(x[j]_t) / proposal(x[j]_t)$
2. Resampling
- a. Ziehe k Partikel, wobei jedes Partikel seine Gewichtung als Wahrscheinlichkeit gezogen zu werden erhält [53]. Da wir eher verschiedene Partikel ziehen wollen, nutzt GMapping das Stochastic Universal Sampling [54].

3.3.3.2 Rao-Blackwellized Particle Filter

Ein Problem von Partikelfiltern ist, dass sie brachial eingesetzt nur in tief dimensionalen Räumen in vernünftiger Zeit funktionieren [55]. In höher dimensionalen Räumen, wäre eine zu grosse Anzahl an Partikel nötig, um die Verteilung im gesamten möglichen Raum genügend abzudecken.

Um Partikel Filter für höher dimensionale Räume einzusetzen nutzt ein Rao-Blackwellized Particle Filter folgende Idee: Wenn wir die Verteilung für Eigenschaften a und b also $p(a, b)$ abdecken wollen und wir aus a b effizient berechnen können, können wir einzig $p(a)$ mit Partikeln abdecken und $p(b | a)$ für jedes Partikel separat berechnen. Diese Technik wird Rao-Blackwellisation genannt.

3.3.3.3 GMapping

Der in GMapping verwendete Rao-Blackwellized Particle Filter nähert den zurückgelegten Weg des Roboters (a) an. Aus dem Weg können wir die dazugehörige Karte (b) für jedes Particle effizient berechnen [45].

Mathematisch:

$x_{0:t}$: Positionen des Roboters vom Start (0) bis zum Zeitpunkt t .

m : Karte der Umgebung

$z_{1:t}$: Beobachtungen des Roboters vom Zeitpunkt 1 bis zum Zeitpunkt t

$u_{1:t}$: Steuerungssignale des Roboters vom Zeitpunkt 1 bis zum Zeitpunkt t

$$p(x_{0:t}, m | z_{1:t}, u_{1:t}) = p(x_{0:t} | z_{1:t}, u_{1:t}) p(m | x_{1:t}, z_{1:t})$$

Die reine Odometrie-Verteilung $x[k]_t \sim p(x_t | x[k]_{t-1}, u_t)$ als Proposal Distribution des Partikel Filters ist nicht ausreichend (für die Abdeckung der Unsicherheit wären zu viele Partikel nötig) und führt zu nicht realistischen Ergebnissen (Siehe Grid-Based FastSLAM [56], Folie 11). GMapping verwendet daher eine verbesserte Proposal Distribution, welche die letzte Beobachtung (den letzten Laserscan) ebenfalls berücksichtigt: $x[k]_t \sim p(x_t | x[k]_{t-1}, m[k], u_t, z_t)$

Beim Resampling Step macht GMapping ebenfalls eine Optimierung: Es nutzt Selective Resampling [57]: Der Resampling Schritt wird nur ausgeführt, wenn er sinnvoll ist, also wenn die Gewichtung der einzelnen Partikel stark voneinander abweicht. Ansonsten haben wir eine höhere Gefahr richtige Partikel frühzeitig zu streichen und zu vergessen. Selective Resampling wird über den Wert $n_t = (\sum_i^N (w[i]_t))^{-2}$ gesteuert (die Gewichte sind normalisiert). Dieser Wert liegt zwischen 1 und der Anzahl an Partikel (N) und beschreibt, wie viele effektive Partikel es in etwa noch gibt. Erst wenn n_t einen definierbaren Threshold unterschreitet wird ein Resampling durchgeführt.

3.3.4 Grid Map

Die von GMapping erstellte Karte ist eine 2D Grid Map. Eine Grid Map ist eine einheitliche Diskretisierung der Umgebung. Die Umgebung wird in Kacheln unterteilt, welche entweder besetzt

(schwarz), frei (weiss) oder unbekannt (grau) sein können. Es interessiert dabei nicht, was ein Hindernis ist, sondern nur wo die Hindernisse sind.

3.4 Navigation

3.4.1 Einleitung

Die Navigation ist ein bekanntes Problem in der Robotik: Mit oder ohne Karte muss bestmöglich ein geografisches Ziel erreicht werden. Dabei muss der Roboter auf dynamische oder unbekannte Hindernisse reagieren können, sprich den Hindernissen ausweichen.

3.4.2 Mögliche Technologien

In der ROS Community hat sich das Navigation Stack Package durchgesetzt [58], welches für kleine wie grössere Roboter designet wurde.

Wir greifen auf diese Standardlösung für das Navigationsproblem zurück.

3.4.3 Navigation Stack

Der Navigation Stack nimmt Sensordaten als Input und berechnet daraus mit der Karte eine Global Costmap sowie eine Local Costmap. Der Global Planner berechnet mit der Global Costmap einen Pfad von der aktuellen Roboterposition zum gewünschten Ziel. Auf unerwartete Hindernissen (Menschen oder durch eine falsch angenommene Roboterposition entstanden) reagiert der Local Planner, welcher den Plan vom Global Planner, die Local Costmap sowie die Odometrie Information des Roboters (Geschwindigkeit) nutzt, um die finalen Bewegungsbefehle zu berechnen und zu veröffentlichen.

Steckt der Roboter fest, schaltet der Navigation Stack verschiedene Recovery Behaviors durch.

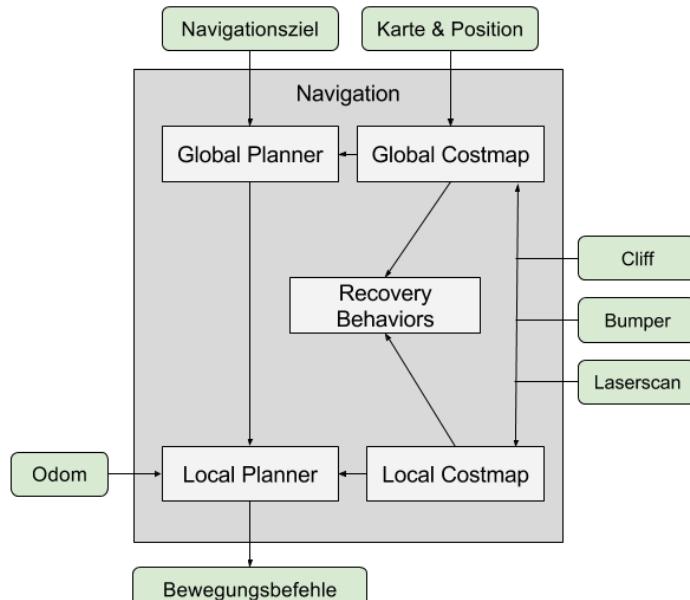


Abbildung 13: Aufbau der Navigation Stack Komponente

3.4.3.1 Global Planner

Der Navigation Stack kennt drei Global Planner Optionen:

- carrot_planner
- navfn
- global_planner

3.4.3.1.1 carrot_planner

Der `carrot_planner` [59] ist sehr einfach. Er sucht den Punkt, der vom Roboter aus am nächsten zum Ziel ist, ohne das ein Hindernis zwischen Punkt und Roboter ist und fährt geradlinig dorthin.

3.4.3.1.2 navfn

Navfn [60] ist ein Planner, der in der Global Costmap mittels dem Dijkstra Algorithmus einen Weg vom Roboter zum Ziel sucht.

3.4.3.1.3 global_planner

Global_planner ist eine flexiblere Weiterentwicklung von navfn. Optional kann anstelle des Dijkstra Algorithmus der A* Algorithmus verwendet werden, was die Suche nach einem optimalen Pfad beschleunigen kann. Allerdings hat das global_planner Package den A* nur mit einer Von Neumann Neighborhood [61] umgesetzt, anstelle einer Moore Neighborhood [62], was zu suboptimalen Wegen führen kann (Siehe Fehler! Verweisquelle konnte nicht gefunden werden.).

3.4.3.1.4 Entscheidung

Obwohl der global_planner flexibler ist, haben wir uns für den navfn Planner entschieden, da wir die extra Features vom global_planner nicht benötigen: A* ist zwar schneller, findet aber nicht den optimalen Pfad und wir haben keine Performanceprobleme festgestellt.

Falls die Navigation bei grösseren Karten zu Performanceproblemen führt, könnte man den navfn Planner schnell über einen Parameter in move_base.yaml mit dem global_planner ersetzen oder einen eigenen Global Planner mit A* Algorithmus und Moore Neighborhood implementieren.

3.4.3.2 Local Planner

Navigation Stack kennt drei Local Planner Optionen:

- dwa local planner
- eband local planner
- teb local planner

3.4.3.2.1 dwa local planner

Der dwa_local_planner nutzt ein Dynamic Window Approach, also simuliert verschiedene Möglichkeiten, bewertet diese und selektiert die beste – dies wird dann in einem vorgegebenen Intervall wiederholt. Die beste Möglichkeit ist nicht gecrasht, von Hindernissen entfernt, näher am Ziel und näher am berechneten Pfad des Global Planners.

3.4.3.2.2 eband local planner

Nur holonomic Roboter werden vom eband_local_planner unterstützt [63]. Somit ist er für uns nicht interessant.

3.4.3.2.3 teb_local_planner

Der teb_local_planner ist ähnlich dem dwa_local_planner, da beide mit non-holonomic Robotern funktionieren. Allerdings nutzt er einen anderen Algorithmus, um das Ziel zu erreichen.

3.4.3.2.4 Entscheidung

Wir haben den dwa_local_planner gewählt, da er die «go-to choice» ist [64]. Die Ergebnisse waren ausreichend (Siehe 5.4 Navigation), daher haben wir den teb_local_planner nicht ausprobiert.

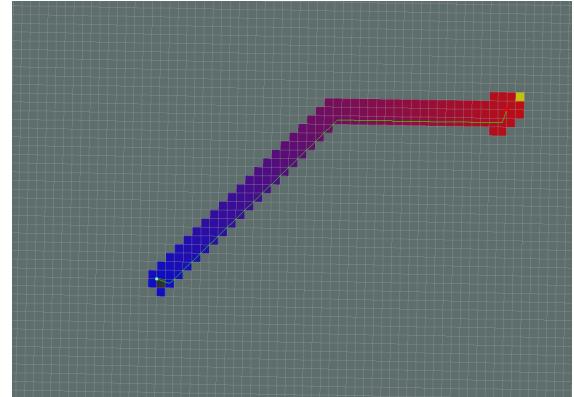


Abbildung 14: Suboptimaler Weg bei Verwendung des A* mit Von Neumann Neighborhood (Quelle: http://wiki.ros.org/global_planner#A.2A)

3.4.3.3 Recovery Behavior

Im Navigation Stack gibt es zwei Recovery Behaviors:

- `rotate_recovery` [65]
Der Roboter dreht sich um 360 Grad
- `clear_costmap_recovery` [66]
Die Local Costmap wird verworfen

Wir verwenden für IANA beide Recovery Behaviors.

3.5 Controller

3.5.1 Einleitung

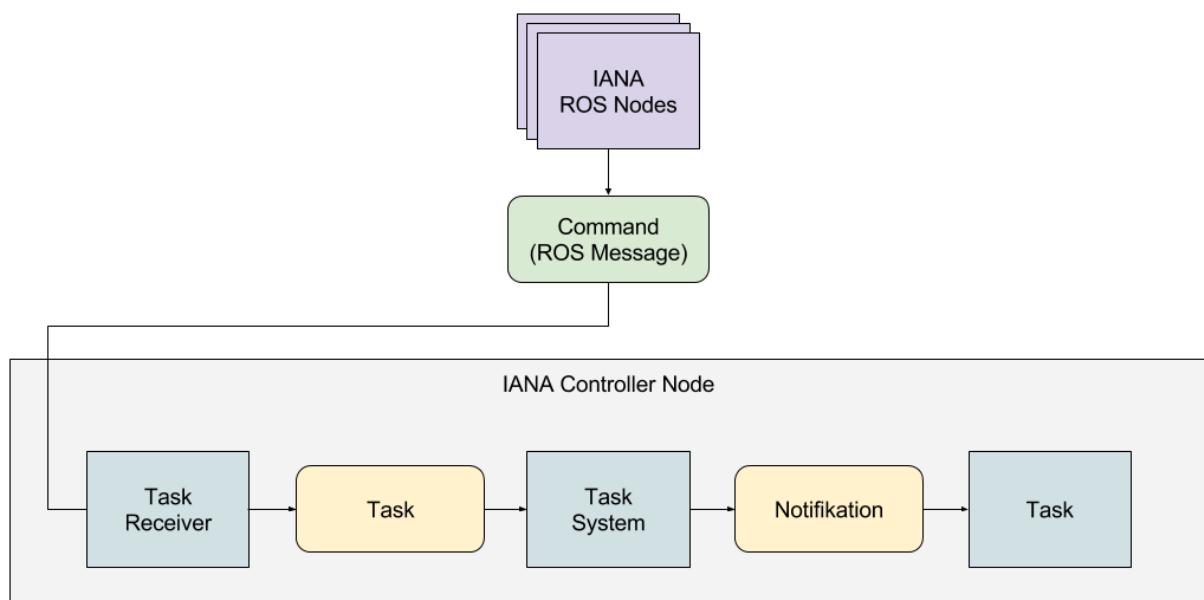


Abbildung 15: Übersicht über die Controller Komponente

Die Controller Komponente ist die Steuerungseinheit von IANA und entscheidet darüber, welcher Befehl wie und wann ausgeführt wird. Sie erhält interne Befehle von IANA selbst sowie externe Befehle von einer Benutzerschnittstelle. Der Task Receiver empfängt alle Befehle in Form von ROS Messages und instanziert die entsprechenden Tasks. Diese übermittelt er anschliessend dem Task System.

Das Task System verwaltet, plant und führt die zugewiesenen Tasks aus. Es stellt sicher, dass immer nur ein Task gleichzeitig läuft. Im Task selbst läuft ein Prozess ab, der den entsprechenden Befehl erfüllt.

Hier ist eine Liste aller Befehle, die IANA kennt:

Befehl	Beschreibung	Herkunft
Explore (s)	Erkunde für s Sekunden die Umgebung.	extern
Goto (x, y, q, qw)	Gehe zum Punkt P(x, y) mit der Zielorientierung des Quaternion Q(q, qw).	extern

Goto (x, y)	Gehe zum Punkt P(x, y) mit der Ausrichtung eines Quaternion mit initialer Orientierung.	extern
Greet (bp) bp ist bekannt	Begrüsse eine bekannte Person bp.	intern
Greet (up) up ist unbekannt	Begrüsse eine unbekannte Person up.	intern
GetToKnow(up)	Lerne eine unbekannte Person kennen, indem du nach dem Namen fragst.	intern

3.5.2 Task

In Tasks ist definiert, wie das Verhalten von IANA auf eingegangene Befehle zu sein hat. Jeder Task hat einen vom Task System verwalteten Zustand, welcher ihm über entsprechende Methoden-Aufrufen mitgeteilt wird. Ein Task wird ausgeführt, indem dessen Update-Methode in einem konstanten Zeitintervall aufgerufen wird.

3.5.2.1 Zustände

Von dem Zeitpunkt seiner Instanziierung bis zu seiner Terminierung kann ein Task vier unterschiedliche Zustände einnehmen:

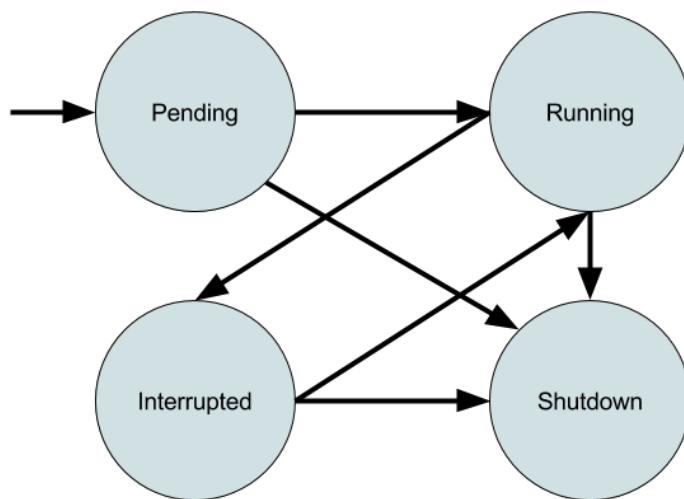


Abbildung 16: Task Zustandsdiagramm

Zustand	Beschreibung
Pending	Der Task wurde erstellt, jedoch noch nicht ausgeführt.
Running	Der Task wurde gestartet und läuft momentan.
Interrupted	Der Task wurde gestartet, ist momentan jedoch durch einen oder mehrere Tasks unterbrochen.
Shutdown	Der Task hat terminiert und wurde aufgeräumt.

3.5.2.2 Notifikationen

Jedem Task wird sein Zustandswechsel über einen entsprechenden Methoden-Aufruf mitgeteilt.

Notifikation	Beschreibung
Start	Wird beim Zustandswechsel von «Pending» auf «Running» aufgerufen. Diese Methode wird garantiert nur einmal und garantiert vor dem ersten Update-Aufruf aufgerufen.
Interrupt	Wird beim Zustandswechsel von «Running» auf «Interrupted» aufgerufen. Nach dem Aufruf dieser Methode erfolgt vor dem nächsten ResUME-Aufruf garantiert kein Update-Aufruf mehr.
ResUME	Wird beim Zustandswechsel von «Interrupted» auf «Running» aufgerufen. Diese Methode wird garantiert vor dem ersten Update-Aufruf und nach dem Interrupt-Aufruf aufgerufen.
Shutdown	Wird aufgerufen, nachdem ein Task durch das Task System beendet und entfernt wurde. Dies kann entweder nach dem Terminieren eines Tasks (Normalfall) oder beim kontrollierten Shutdown des Task Systems erfolgen. Diese Methode wird garantiert nur einmal aufgerufen und ist garantiert die letzte Notifikation.
Update	Wird in einem konstanten Intervall aufgerufen, falls sich der Task im Zustand «Running» befindet. Es wird erwartet, dass die Ausführzeit der aufgerufenen Update-Methode, die Länge des Update-Intervalls nicht überschreitet.

3.5.2.3 Unterbrechung

Tasks können durch einen anderen Task unterbrochen werden. Dies geschieht aber nur, falls ein Task dies explizit zulässt. Dazu hat jeder Task die Methode `InterruptibleBy(Task) → Bool`, über welche er angefragt werden kann, ob der entsprechende Task ihn unterbrechen darf oder nicht. Falls ein Task diese Methode nicht überschreibt, wird angenommen, dass er nicht unterbrechbar ist.

3.5.2.4 Terminierung

Tasks verfügen über ein `Terminated`-Flag über das sie dem Task System ihre Terminierung mitteilen.

3.5.3 Task System

Das Task System dient als Scheduler für die eingehenden Tasks. Es ist verantwortlich dafür, dass die Tasks in der korrekten Reihenfolge abgearbeitet werden, wobei zu jedem Zeitpunkt maximal ein Task ausgeführt wird. Alle anderen Tasks müssen warten und werden in Wartelisten eingereiht.

Im Detail hat das Task System folgende Aufgaben:

Aufgabe	Beschreibung
Entgegennehmen neuer Tasks	Eingehende Tasks werden vom Task System entgegengenommen und sofort ausgeführt, falls dies möglich ist [SIEHE TASK → UNTERBRECHUNG]. Falls dies nicht der Fall ist, werden Tasks in Warteschlangen eingereiht, bis sie an der Reihe sind.
Ausführung der Tasks	Das Task-System ermittelt, welcher Task an der Reihe ist und führt diesen aus, bzw. setzt dessen Ausführung fort.

Verwaltung der Zustände aller Tasks	Das Task System verwaltet den Zustand jedes einzelnen Tasks.
Durchführung von Zustandswechsel	Das Task System trifft alle nötigen Vorkehrungen für einen Zustandswechsel eines Tasks. Vorkehrungen sind Notifikation des betroffenen Tasks und ändern dessen Zustandszuordnungen.
Terminierte Tasks aufräumen	Ein Task, welcher dem Task System seine Terminierung mitteilt, wird aus den Bearbeitungslisten des Task System entfernt und anschliessend über sein Herunterfahren (Shutdown) notifiziert.

3.6 IANA Navigation

3.6.1 Einleitung

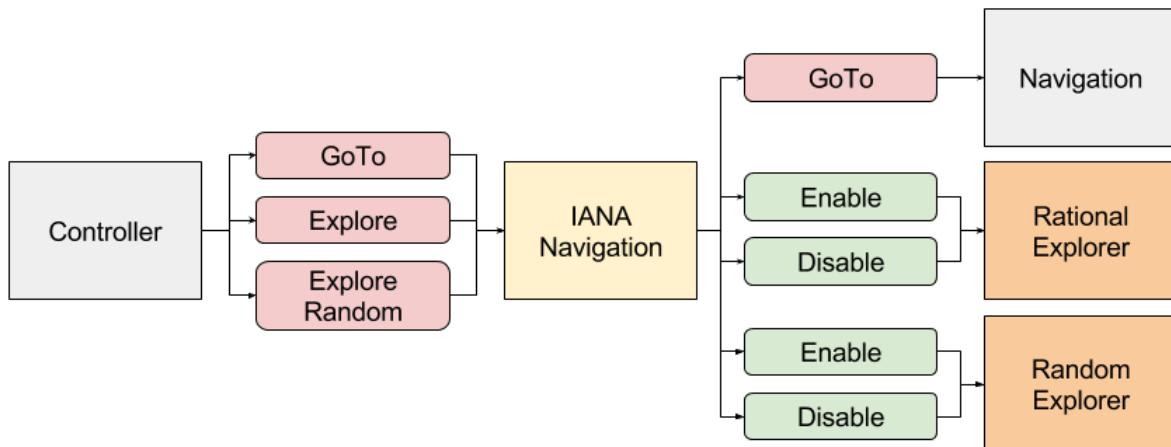


Abbildung 17: Übersicht über die IANA Navigation Komponente

Die IANA Navigation Komponente dient als Abstraktion der Navigation und Explorer Komponenten. Sie bietet dem Controller ein einheitliches Interface, welches die dahinterliegenden Komponenten versteckt. Dies reduziert die Anzahl Komponenten, die der Controller kennen muss und ermöglicht ein Austausch der Navigation und Explorer Komponenten ohne die Controller Tasks anpassen zu müssen.

Zudem garantiert die IANA Navigation, dass nur eine der versteckten Komponenten gleichzeitig aktiv ist. So veröffentlichen nicht zwei Komponenten gleichzeitig unterschiedliche Steuerungsbefehle, was die Navigation des Roboters unbrauchbar machen würde.

3.7 Collision Detector

3.7.1 Einleitung

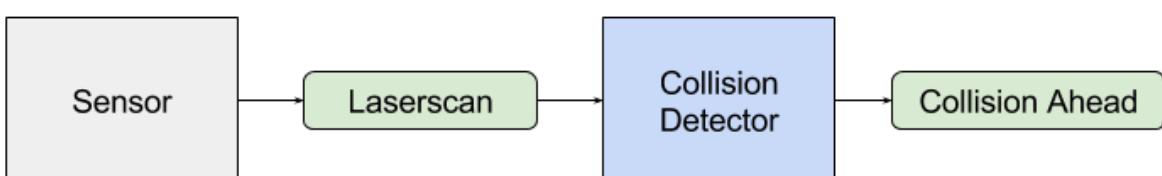


Abbildung 18: Übersicht über die Collision Detector Komponente

Die Collision Detector Komponente ermöglicht es IANA zu erkennen, falls eine Kollision bevorsteht. Als Input erhält sie die Laserscandaten des Tiefenscan-Sensors und veröffentlicht die Distanz zum nächsten Hindernis in Fahrtrichtung.

Für die eingeschränkte Scanweite der Kinect oder der ZED und für die 360 Grad Scanweite existieren zwei unterschiedliche Implementierung des Collision Detectors. Je nach Sensor kann die passende Komponente verwendet und gestartet werden.

3.7.2 KinectOne und ZED

Der Collision Detector für die KinectOne und die ZED funktioniert indem wir zuerst nur die innere des Laserscans verwenden. Ansonsten würde IANA Hindernisse die zu ihrer rechten oder linken Seite liegen als Frontale Hindernisse erkennen. Anschliessend überprüfen wir, wie viele Werte im unter dessen Minimum liegen. Ist dies der Fall gehen wir davon aus, dass ein Hindernis vor uns liegt, da beide Sensoren nahe Objekte nicht erkennen können (Siehe 0

Testszenario 2). Zuletzt filtern wir Werte unter dem Minimum aus den Laserscan Daten, da die KinectOne ein Tiefenbild mit Noise liefert. Dann nehmen wir die kleinste Distanz der übrig gebliebenen Laserscan-Daten und veröffentlichen sie als ROS Message.

3.7.3 RPLidar

Beim RPLidar reduzieren wir die 360 Grad Range auf die Frontalen 45 Grad zu und selektieren anschliessend das Minimum der Laserscan-Daten.

3.8 Explorer

3.8.1 Einleitung

Die Explorer Komponente ist dafür zuständig, dass IANA selbstständig unbekannte Räume erkundet. Der Explorer gibt dafür dem Roboter vor, wo er hinfahren soll, um über die Kartographie neue Bereiche zu erschliessen.

Wir haben zwei verschiedene Verhalten eines Explorers definiert:

- Zufälliges Herumfahren, fortan «Random Explorer» genannt
- Rationales Erkunden durch Erkennen und Ansteuern von unbekannten Regionen, fortan «Rational Explorer» genannt

3.8.2 Random Explorer

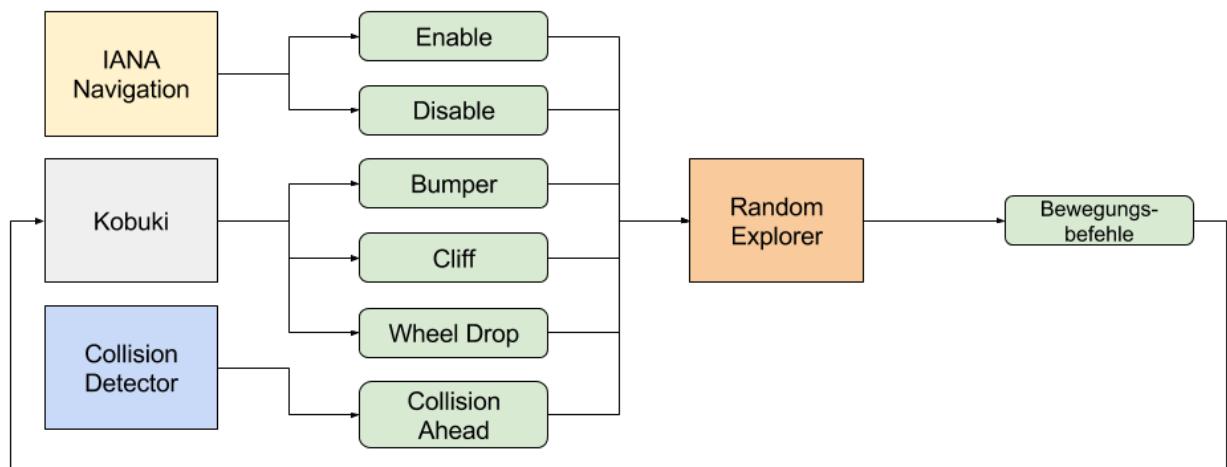


Abbildung 19: Übersicht über die Random Explorer Komponente

Der Random Explorer soll IANA zufällig durch den Raum navigieren. Durch den Zufall wird IANA mit genügend Zeit (mit einer Wahrscheinlichkeit die gegen 1 strebt) auch unbekannte Bereiche der Karte befahren und kartographieren.

Informationen über IANAs Umwelt erhält der Random Explorer über die Kobuki Bumper-, Cliff- und Wheel-Drop-Sensoren sowie den «Collision Ahead»-Notifikationen des Collision Detectors.

3.8.2.1 IANA Random Explorer

3.8.2.1.1 Definition

Es gibt unterschiedliche Möglichkeiten, das Verhalten eines Random Explorers zu definieren. Unserer Random Explorer hat folgendes Verhalten: Falls eine Kollision unmittelbar bevorsteht oder bereits eingetreten ist, dreht IANA um einen zufälligen Winkel. Ansonsten fährt IANA stur geradeaus.

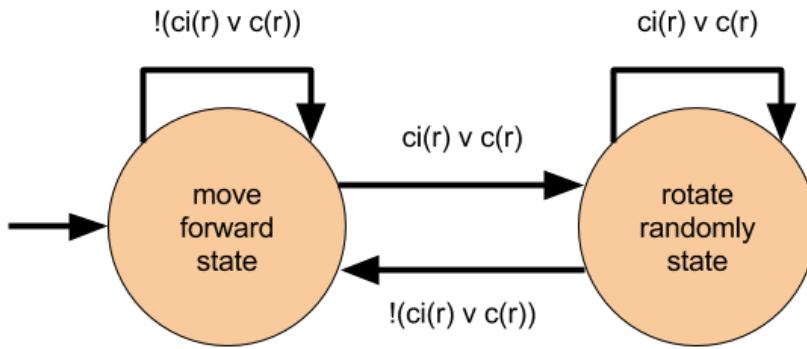


Abbildung 20: IANA Random Explorer Zustandsdiagramm

Zustand	Beschreibung
Forward	Der Roboter bewegt sich mit einer Geschwindigkeit von $vf(r)$
Rotate Random	Der Roboter rotiert an Ort und Stelle mit einer Geschwindigkeit von $va(r)$ um seine vertikale Achse bis er eine zufällig gewählte Rotation a erreicht.

3.8.2.1.2 Parameter

Parameter	Beschreibung
COLLISION_AHEAD	Threshold, bei dessen Unterschreitung eine Kollision unmittelbar bevorsteht.
COLLISION_WARN	Threshold, bei dessen Unterschreitung eine Kollision in kurzer Zeit bevorsteht und nur mit MIN_FORWARD_VEL Geschwindigkeit gefahren werden kann.
COLLISION_INFO	Threshold, bei dessen Unterschreitung eine Kollision in mittlerer Zeit bevorsteht und nur mit MEDIUM_FORWARD_VEL Geschwindigkeit gefahren werden kann.
MIN_FORWARD_VEL	Mindestgeschwindigkeit
MEDIUM_FORWARD_VEL	Mittlere Geschwindigkeit
MAX_FORWARD_VEL	Maximale Geschwindigkeit
MIN_ANGULAR_VEL	Minimale Rotationsgeschwindigkeit
MAX_ANGULAR_VEL	Maximale Rotationsgeschwindigkeit

3.8.2.1.3 Funktionen

Funktion	Beschreibung

ci(r)	<p>Collision Imminent</p> <p>Der Random Explorer ermittelt anhand der Distanz zum nächsten Hindernis in Fahrtrichtung, ob eine Kollision unmittelbar bevorsteht. Die Distanz zum nächsten Hindernis wird ihm von der Collision Detector Komponente mitgeteilt und mit dem Threshold «COLLISION_AHEAD» verglichen. Falls ein Messwert diesen Threshold unterschreitet, steht eine Kollision unmittelbar bevor. Der Threshold muss so gewählt werden, dass IANA noch genügend Zeit zum Bremsen bleibt und darf die Mindestscanweite des verwendeten Sensors nicht unterschreiten.</p> <table border="1" data-bbox="366 534 1377 714"> <thead> <tr> <th>Funktionswert</th><th>Bedingung</th></tr> </thead> <tbody> <tr> <td>True</td><td>r.MinDistance <= COLLISION_AHEAD</td></tr> <tr> <td>False</td><td>Sonst</td></tr> </tbody> </table>	Funktionswert	Bedingung	True	r.MinDistance <= COLLISION_AHEAD	False	Sonst				
Funktionswert	Bedingung										
True	r.MinDistance <= COLLISION_AHEAD										
False	Sonst										
c(r)	<p>Collision</p> <p>Falls eine unerwartete Kollision stattgefunden hat, erfährt der Random Driver dies via Bumper-, Cliff- und Wheel-Drop-Sensordaten. Dies kann durchaus eintreffen, da der Laserscan-Sensor nur für eine Höhe die Distanz zu seiner Umgebung misst. Neben Abgründen werden Hindernisse, die den Roboter auf einer anderen Höhe treffen, so erkannt.</p> <table border="1" data-bbox="366 990 1377 1260"> <thead> <tr> <th>Funktionswert</th><th>Bedingung</th></tr> </thead> <tbody> <tr> <td>True</td><td>Any(r.Bumper) oder Any(r.Cliff) oder Any(r.Wheel-Drop)</td></tr> <tr> <td>False</td><td>Sonst</td></tr> </tbody> </table>	Funktionswert	Bedingung	True	Any(r.Bumper) oder Any(r.Cliff) oder Any(r.Wheel-Drop)	False	Sonst				
Funktionswert	Bedingung										
True	Any(r.Bumper) oder Any(r.Cliff) oder Any(r.Wheel-Drop)										
False	Sonst										
vf(r)	<p>Forward Velocity</p> <p>Berechnet die Vorwärtsgeschwindigkeit des Roboters im aktuellen Zustand r. Ziel ist es, dass IANA in jedem Fall genügend schnell abbremsen kann, daher muss die Geschwindigkeit in der Nähe von Hindernissen reduziert werden. Dazu wird wiederum die Distanz zum nächsten Hindernis mit diesmal drei Thresholds verglichen. Je nachdem zwischen welche Thresholds die Distanz fällt, wird eine andere Geschwindigkeit zurückgegeben.</p> <table border="1" data-bbox="366 1574 1377 1956"> <thead> <tr> <th>Funktionswert</th><th>Bedingung</th></tr> </thead> <tbody> <tr> <td>MAX_FORWARD_VEL</td><td>r.MinDistance >= COLLISION_INFO</td></tr> <tr> <td>MEDIUM_FORWARD_VEL</td><td>r.MinDistance < COLLISION_INFO und r.MinDistance >= COLLISION_WARN</td></tr> <tr> <td>MIN_FORWARD_VEL</td><td>r.MinDistance < COLLISION_WARN und r.MinDistance >= COLLISION_AHEAD</td></tr> <tr> <td>0</td><td>r.MinDistance < COLLISION_AHEAD</td></tr> </tbody> </table>	Funktionswert	Bedingung	MAX_FORWARD_VEL	r.MinDistance >= COLLISION_INFO	MEDIUM_FORWARD_VEL	r.MinDistance < COLLISION_INFO und r.MinDistance >= COLLISION_WARN	MIN_FORWARD_VEL	r.MinDistance < COLLISION_WARN und r.MinDistance >= COLLISION_AHEAD	0	r.MinDistance < COLLISION_AHEAD
Funktionswert	Bedingung										
MAX_FORWARD_VEL	r.MinDistance >= COLLISION_INFO										
MEDIUM_FORWARD_VEL	r.MinDistance < COLLISION_INFO und r.MinDistance >= COLLISION_WARN										
MIN_FORWARD_VEL	r.MinDistance < COLLISION_WARN und r.MinDistance >= COLLISION_AHEAD										
0	r.MinDistance < COLLISION_AHEAD										

va(r, a)	<p>Angular Velocity Berechnet die Rotationsgeschwindigkeit des Roboters im aktuellen Zustand r mit dem Zielwinkel a.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 5px;">Funktionswert</th><th style="padding: 5px;">Bedingung</th></tr> </thead> <tbody> <tr> <td style="padding: 5px;">MIN_ANGULAR_VEL</td><td style="padding: 5px;">$(r.Rotation - a) < 30^\circ$</td></tr> <tr> <td style="padding: 5px;">MAX_ANGULAR_VEL</td><td style="padding: 5px;">Sonst</td></tr> </tbody> </table>	Funktionswert	Bedingung	MIN_ANGULAR_VEL	$(r.Rotation - a) < 30^\circ$	MAX_ANGULAR_VEL	Sonst						
Funktionswert	Bedingung												
MIN_ANGULAR_VEL	$(r.Rotation - a) < 30^\circ$												
MAX_ANGULAR_VEL	Sonst												
r	<p>Roboter Zustand Über IANAs Sensoren erhält der Random Explorer Informationen über IANAs Umgebung.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="padding: 5px;">Velocity</td><td style="padding: 5px;">Die momentane Geschwindigkeit von IANA</td></tr> <tr> <td style="padding: 5px;">MinDistance</td><td style="padding: 5px;">Die Distanz zum nächsten Hindernis in Fahrtrichtung</td></tr> <tr> <td style="padding: 5px;">Bumper</td><td style="padding: 5px;">Informationen, ob einer von IANAs Bumper aktiviert wurde.</td></tr> <tr> <td style="padding: 5px;">Cliff</td><td style="padding: 5px;">Informationen, ob einer von IANAs Cliff (Abgrund) Sensoren einen Abgrund registriert hat.</td></tr> <tr> <td style="padding: 5px;">Wheel-Drop</td><td style="padding: 5px;">Informationen, ob einer von IANAs Wheel-Drop) Sensoren einen Abgrund registriert hat.</td></tr> <tr> <td style="padding: 5px;">Rotation</td><td style="padding: 5px;">Die momentane Rotation von IANA.</td></tr> </tbody> </table>	Velocity	Die momentane Geschwindigkeit von IANA	MinDistance	Die Distanz zum nächsten Hindernis in Fahrtrichtung	Bumper	Informationen, ob einer von IANAs Bumper aktiviert wurde.	Cliff	Informationen, ob einer von IANAs Cliff (Abgrund) Sensoren einen Abgrund registriert hat.	Wheel-Drop	Informationen, ob einer von IANAs Wheel-Drop) Sensoren einen Abgrund registriert hat.	Rotation	Die momentane Rotation von IANA.
Velocity	Die momentane Geschwindigkeit von IANA												
MinDistance	Die Distanz zum nächsten Hindernis in Fahrtrichtung												
Bumper	Informationen, ob einer von IANAs Bumper aktiviert wurde.												
Cliff	Informationen, ob einer von IANAs Cliff (Abgrund) Sensoren einen Abgrund registriert hat.												
Wheel-Drop	Informationen, ob einer von IANAs Wheel-Drop) Sensoren einen Abgrund registriert hat.												
Rotation	Die momentane Rotation von IANA.												

3.8.3 Rational Explorer

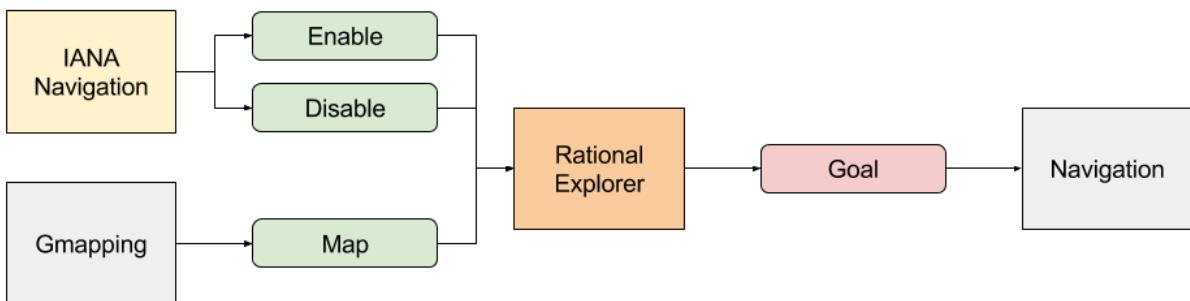


Abbildung 21: Übersicht über die Rational Explorer Komponente

Der im vorhergehenden Abschnitt beschriebene Random Explorer ist sehr simpel und erfüllt den Zweck des Erkundens von Räumen. Die Erkundung mittels zufälligem Umherfahren benötigt jedoch unnötig viel Zeit, vor allem enge Passagen wie Türen oder Gänge werden aufgrund des Zufalls nur sehr selten passiert. Dies erschwert insbesondere das Erkunden von Umgebungen mit mehreren Räumen, wie dies am Institut der Fall ist.

Mit dem Rational Explorer wollen wir einen Erkundungsalgorithmus einführen, der IANA nicht zufällig, sondern möglichst effizient seine Umgebung erkundet lässt. Seine Kernaufgabe ist es, auf Basis seines

Wissens über die Umwelt von IANA die vielversprechendsten Stellen für die Erkundung zu ermitteln und als Ziele an die Navigation Komponente weiterzugeben.

3.8.3.1 *Frontier Based Approach*

Eine vielversprechende Methode zur Erkennung dieser Zielpunkte ist der «Frontier-Based Exploration» Ansatz [67]: Es werden Grenzen zwischen freien und unbekannten Regionen in der Karte gefunden und anschliessend vom Roboter angesteuert. An diesen Grenzen bekommt der Roboter dann unbekannte Regionen zu sehen, und das bekannte Territorium wird erweitert.

Wichtig bei diesem Ansatz ist, dass die ausgewählten Grenzen für den Roboter zugänglich sein müssen.

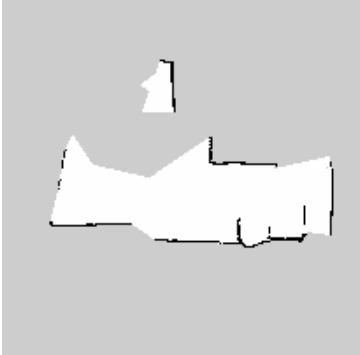
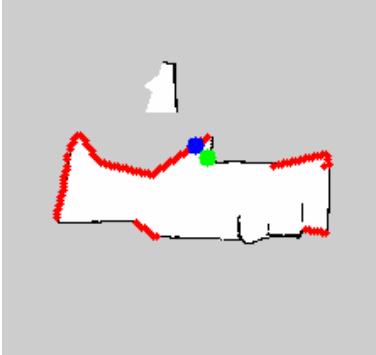
3.8.3.2 *Detect Frontiers Algorithmus*

Der von uns entworfene Algorithmus zur Bestimmung der Fronten beruht auf einer einfachen Anwendung des Flood-Fill Algorithmus.

Die Inputs des Algorithmus sind die momentane Roboterposition s , die vom SLAM-Algorithmus produzierte Grid-Map M sowie ein Threshold t , ab welchem Felder der Grid-Map als besetzt gelten. Der Algorithmus terminiert sofort, falls die M an der Position s unbekannt (-1) oder besetzt ($\geq t$) ist. Ansonsten wird die Karte mittels BFS und von Neumann Neighborhood traversiert, wobei nur jene Felder welche frei (≥ 0 und $< t$) sind, besucht werden. Um zu verhindern, dass freie Felder mehrere Male besucht werden, werden sie beim ersten Auffinden auf bearbeitet (-2) gesetzt. Falls der Algorithmus auf ein unbekanntes Feld stösst, handelt es sich um einen Grenzpunkt und wird in die Resultat-Menge aufgenommen.

`DetectFrontiers(s, M, t)`

```
1.    F = {}
2.    if map[start] == -1 || map[start] >= t
3.        return F
4.    neighbours = { (1, 0), (-1, 0), (0, 1), (0,-1) }
5.    Q = List()
6.    Q.PushBack(s)
7.    map[q] = -2
8.    while Q is not empty
9.        curr = Q.PopFirst()
10.       for each n ∈ neighbours
11.           next = curr + neighbour
12.           if 0 <= map[next] < t
13.               map[next] = -2
14.               Q.PushBack(next)
15.           else if map[next] == -1
16.               F = F ∪ {next}
17.    return F
```

	
Abbildung 22: Grid-Map als Input des Detect Frontiers Algorithmus	Abbildung 23: Resultat des Detect Frontiers nach Anwendung auf die Grid-Map aus Abbildung 22

Die von GMapping erstellte Grid-Map: weiss = frei, schwarz = besetzt, grau = unbekannt

Mittels Detect Frontiers Algorithmus erkannte Fronten: grün = Roboter Position, rot = Frontier-Punkte, blau = nächstgelegener Frontier-Punkt.

3.8.3.3 Auswahl des Zielpunktes

Der Detect Frontiers Algorithmus gibt als Resultat eine Liste aller Frontier-Punkte zurück, welche von der momentanen Roboter-Position erreicht werden können. Da die Navigation-Komponente ein genaues Ziel erwartet, muss aus dieser Menge ein Punkt ausgewählt werden. Wir haben folgende zwei Auswahlstrategien umgesetzt:

Methode	Vorteile	Nachteile
Nächstgelegener Punkt	- IANA fährt möglichst wenig Strecke ab, um den ganzen Raum zu erkunden	- Risiko für ewiges Hin- und Herpendeln zwischen zwei Punkten, die zueinander am nächsten liegen aber von SLAM nicht erfasst werden können (z.B. bei Scheiben).
Zufälliger Punkt	- Kein ewiges Hin- und Herpendeln - Große Frontiers mit vielen Frontier-Punkten haben eine größere Wahrscheinlichkeit angefahren zu werden als kleine.	- Nicht kürzeste Route bei perfektem SLAM, um den gesamten Raum zu erkennen.

3.8.3.4 Navigation

Die schlussendliche Navigation zu den Grenzen übernimmt die Navigation-Komponente (Siehe 3.4 Navigation). Der Explorer veröffentlicht ihr lediglich die Koordinaten auf der Karte des ausgewählten Frontier-Punktes.

3.8.3.5 Schwäche

Falls sich der Roboter auf der Karte des SLAM Algorithmus auf einem unbekannten oder auf einem besetzten Feld befindet, wird die Suche nach Grenzen sofort abgebrochen. Der Detect Frontiers Algorithmus gibt also eine leere Liste von Grenzen zurück und es kann keine Grenze angesteuert

werden. Der Rational Explorer müsste warten, bis der SLAM Algorithmus die Karte oder die Position des Roboters erneuert.

3.8.4 Erweiterter Rational Explorer

Der Erweiterte Rational Explorer verbindet den Random Explorer und Rational Explorer. Findet der Rational Explorers keine Grenzen, greift der Erweiterte Rational Explorer auf den Random Explorer für eine gewisse Anzahl an Sekunden zurück. Unter folgenden Umständen werden keine Grenzen gefunden:

- IANA steht auf einem Feld, welches den Status «unbekannt» oder «besetzt» hat.
 - Es existieren keine Fronten mehr, da IANA eine geschlossene Karte gefunden hat.

In der ersten Situation wird IANA durch den Random Explorer das Gebiet um sich erkunden und somit wieder auf Felder mit Status «frei» gelangen. Falls IANA eine geschlossene Karte gefunden hat, heisst dies aber nicht, dass die Karte sicherlich richtig ist. Ein zufälliges Umherfahren ist sinnvoll (solange IANA noch Zeit für die Erkundung hat), um die Karte zu verbessern bzw. Fehler in der geschlossenen Karte zu erkennen.

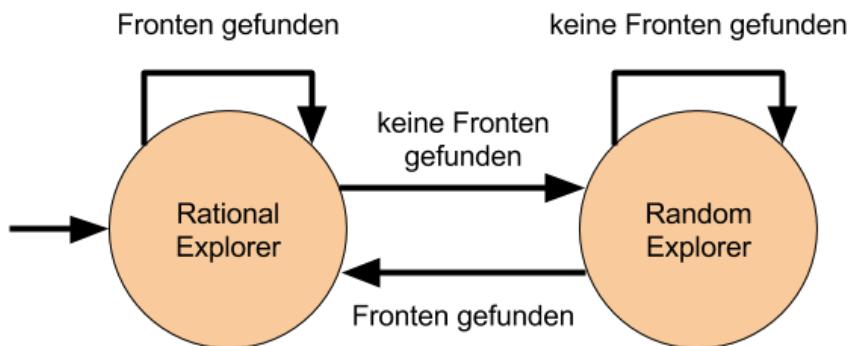


Abbildung 24: Erweiterter Rational Explorer Zustandsdiagramm

3.9 Person Detection

3.9.1 Einleitung

Die Person Detection (Personenerkennung) wurde aus dem IP5 «IANA» Projekt (Separates Dokument «IANA – IMVS Autonomous Neural Assistant») übernommen und in unsere neue ROS Architektur integriert. Sie nimmt RGB Bilder entgegen und wirft verschiedene Personen-Events. Person Detection speichert zudem die erkannten Personen mit deren Namen ab. Dazu verwendet es die Person Data Komponente.

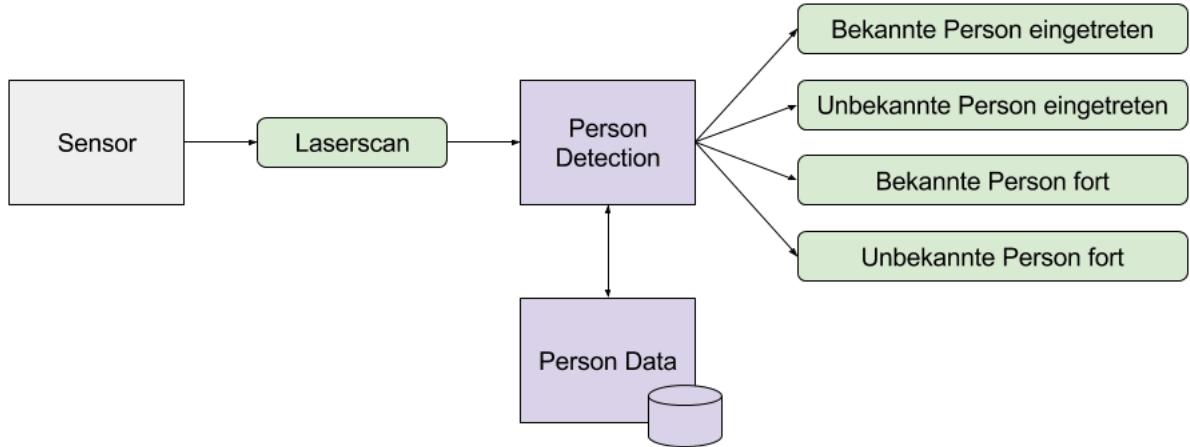


Abbildung 25: Übersicht über die Person Detection Komponente

3.9.2 Verbesserungen gegenüber dem IP5 «IANA» Projekt

3.9.2.1 Schnellere Gesichtsdetektion

Auf IANA ist die DLib Gesichtsdetektion deutlich langsamer als auf unseren Rechnern im IP5 «IANA» Projekt – im Schnitt benötigt IANA auf einem herunterskalierten Bild zwei bis drei Sekunden. Daher haben wir den DLibFaceDetector beschleunigt und zudem den HaarFaceDetector mit OpenCV und vortrainiertem Modell umgesetzt. Der HaarFaceDetector liefert zwar mehr False Positives als DLib, läuft dafür deutlich schneller. Der Geschwindigkeitsunterschied ist auf IANA deutlich stärker, da OpenCV die Grafikkarte ausnutzt, DLib aber nicht (Siehe 5.6.1 Gesichtsdetektion).

Im DLibFaceDetector vom IP5 wurde ein Feature von DLib verwendet, welches zuerst ein up-scaling auf dem Bild anwendet. Dieses wurde für das IP6 ausgeschaltet, da es besser ist unser down-scaling Pre-processing (wir haben ein 720p Videostream) zu drosseln. Ein up-scaling ist nur sinnvoll, wenn das Originalbild in nicht genügend guter Qualität vorhanden wäre, oder wenn die Performanz keine Rolle spielt.

In der Person Detection Node sind beide Gesichtsdetektionen verfügbar, da beide Vor- und Nachteile aufweisen. Für IANA haben wir uns für den neuen und schnellere HaarFaceDetector entschieden.

3.9.2.2 Bessere Gesichtserkennung

Die Gesichtserkennung wurde verbessert, indem wir unterschiedliche Bilder für die Gesichtsdetektion und für die Gesichtserkennung verwenden (im IP5 «IANA» Projekt wurde ein herunter skaliertes Bild für die gesamte Pipeline verwendet). Wir skalieren nur für die Face Detection (aus Performancegründen) das Bild herunter und schneiden die gefundenen Gesichter aus dem Originalbild heraus. So erhalten wir mehr Details in den Gesichtsbildern, was OpenFace hilft die Gesichter genauer zu beschreiben.

3.9.2.3 Aktualisierung der Gesichts Feature-Vektoren

Die zu Personen gespeicherten Gesichtsdaten werden neu mit den Feature-Vektoren von wiedererkannten Gesichtern aktualisiert. Dies ermöglicht IANA Änderungen am Gesicht über die Zeit (wie etwa Alterung oder Gesichtsbehaarung) zu verfolgen und ihr Wissen über die Gesichtseigenschaften anzupassen.

Um zu verhindern, dass kurzzeitige Änderungen (wie etwa Pflaster oder ein Schal) zu starken Verschiebungen des Gesichtsvektors führen, stellen wir sicher, dass nur jene Feature-Vektoren übernommen werden, die einen definierten Threshold im Confidence-Score erreicht haben (Siehe 4.9 Person Detection). So werden nur sehr ähnliche Vektoren für die Aktualisierung verwendet, was nur

kleine Veränderungen nach sich zieht. Nur wenn also die Änderung am Gesicht über längere Zeit bestand hat, führt sie auch zu einer merklichen Verschiebung des Durchschnittsvektors an die neue Stelle.

Ein Threshold für die maximale Anzahl gespeicherter Feature-Vektoren soll es IANA ermöglichen, veraltete Feature-Vektoren zu vergessen und Änderungen am Gesicht nach einiger Zeit vollständig zu akzeptieren.

3.9.2.4 Ansprechende Benutzerschnittstelle

Im IP5 «IANA» Projekt gab es einzig eine Konsolen-Benutzerschnittstelle, die den Nutzer aufforderte, den Namen einzutragen. Nun befindet sich diese Funktionalität in der Benutzerschnittstelle-Komponente, wo mit Vorschaubild die Person nach dem Namen gefragt wird (Siehe 3.12 Benutzerschnittstelle).

3.9.2.5 Neue Events

Im IP5 «IANA» Projekt wurde erkannt, wenn eine Person das Blickfeld IANAs betritt, nicht aber, wenn diese Person das Blickfeld wieder verlässt. Dies führte dazu, dass IANA ewig (oder eine fixe vorgegebene Zeit) auf den Namen einer unbekannten Person warten musste, auch wenn diese längst fort war. Dies führte auch dazu, dass IANA sich nicht von Personen hätte verabschieden können. Im IP6 wurde dies nun verbessert: IANA bricht die Frage nach dem Namen ab, wenn die unbekannte Person aus dem Blickfeld verschwindet. IANA kann ebenfalls Personen verabschieden, wenn diese das Blickfeld verlassen.

3.9.3 Persistieren der vorgestellten Personen

Wenn sich eine Person bei IANA vorstellt, wird diese mit ihren Gesicht-Daten gespeichert. Dazu haben wir die Komponente «Person Data» entworfen. Die Funktionalität des Persistierens wurde für diese Komponente vom IP5 «IANA» übernommen.

3.10 Pan&Tilt-Vorrichtung

IANAs RGB-Kamera hat einen eingeschränkten Sichtbereich (Siehe 3.2.1 ZED Sichtbereich). Eine Pan&Tilt-Vorrichtung gibt IANA die Möglichkeit, Dinge ausserhalb des eigentlichen Sichtfeldes zu verfolgen oder anzuvisieren, indem die Kamera um eine horizontale und eine vertikale Achse innerhalb von 180° rotiert wird.

3.10.1 Servo Motoren

Die bei IANA eingesetzte Pan&Tilt-Vorrichtung verwendet zwei leistungsstarke Servo-Motoren, welche über ein PWM-Signal [68] angesteuert werden. Pulsweitenmodulation (PWM) steht für eine Modulationsart, bei der mit einer konstanten Frequenz ein Rechtecks-Impuls übermittelt wird. Die Länge dieses Impulses entspricht dabei gerade dem übertragenen Wert. Für die Ansteuerung der Servos gilt bei einer Frequenz von 50hz folgende Wertetabelle (Werte zwischen 1ms und 2ms werden linear interpoliert):

Impulsläng	Wert
<= 1 ms	0° Rotation
1.5 ms	90° Rotation
>=2 ms	180° Rotation

Oder als Funktion der Impulslänge t:

$$pan(t), tilt(t) = \begin{cases} \min(180, \frac{180}{t-1}), & t > 1 \\ 0, & \text{sonst} \end{cases}$$

3.10.2 Ansteuerung

Zur Ansteuerung der Pan&Tilt-Vorrichtung wurden von uns zwei Methoden untersucht, via den vorhandenen Kobuki Digital Output Pins sowie via einem zusätzlichen Arduino Boards. In beiden Fällen müssen PWM Signale für die Servos der Pan&Tilt-Vorrichtung erzeugt werden.

3.10.2.1 Via Kobukis Digital Output Pins

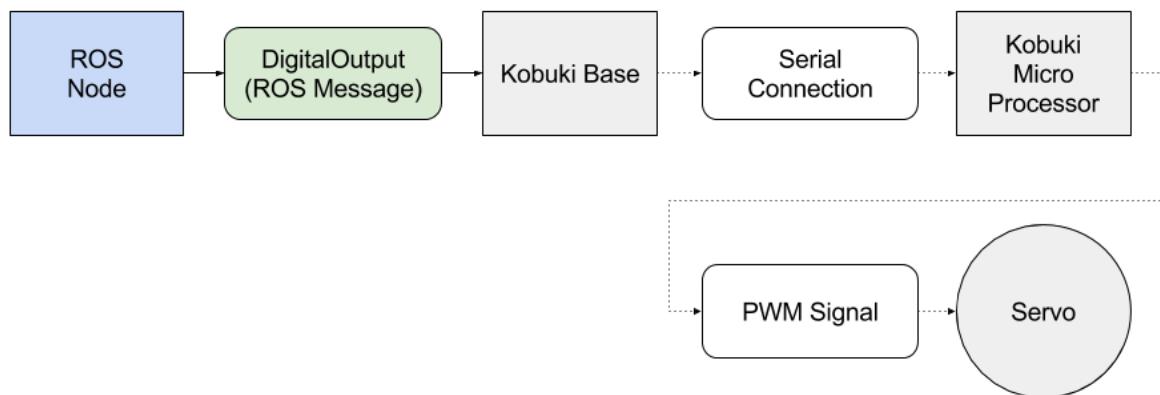


Abbildung 26: Ansteuerung der Servos via Kobukis Digital Output Pins

Der Kobuki Roboter verfügt über einen Serial Port mit Input und Digital Output Pins. Die Ansteuerung der Digital Output Pins läuft via Kobuki ROS-Node, über welche Digital Output Pins ein- oder ausgeschaltet werden können. Die Kobuki ROS-Node wiederum kommuniziert via serieller Schnittstelle mit dem Kobuki Mikroprozessor, welcher dann das Signal auf den Pins generiert. Durch getimpftes An- und Ausschalten der Output Pins kommt so ein PWM Signal zustande.

Vorteile	Nachteile
<ul style="list-style-type: none"> + <u>Hardware ist bereits vorhanden</u> Es ist keine zusätzliche Hardware nötig. 	<ul style="list-style-type: none"> - <u>Steuerung läuft als Prozess</u> Die Kobuki ROS-Node läuft als Prozess auf dem Ubuntu System. Als Prozess haben wir keine Echtzeitsgarantie, der Prozess könnte jeder Zeit unterbrochen werden. Dies kann zu einer ungenauen Impulsweite für die PWM Signale führen. - <u>Nicht verwendbar</u> Unsere Tests (Siehe Abbildung 27) haben folgendes ergeben: Der Kobuki Micro Processor steuert die Digital Output Pins mit einer maximalen Frequenz von 100 Hz an. Zur Steuerung der Servos sind PWM-Impulse mit einer Pulsweite zwischen 1 und 2 ms nötig – also 1000Hz - 500Hz. Also kann der «Kobuki Micro Processor» und damit die Digital Output Pins so nicht verwendet werden.

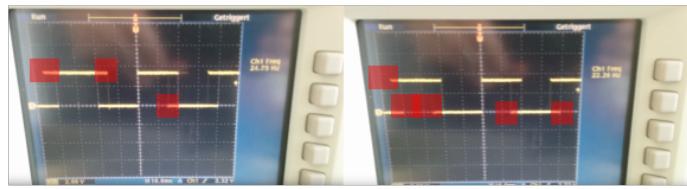


Abbildung 27: Das Signal auf 2 und 1 Millisekunde ist nicht konstant und schwankt stark (rot markiert)

3.10.2.2 Via Arduino Board

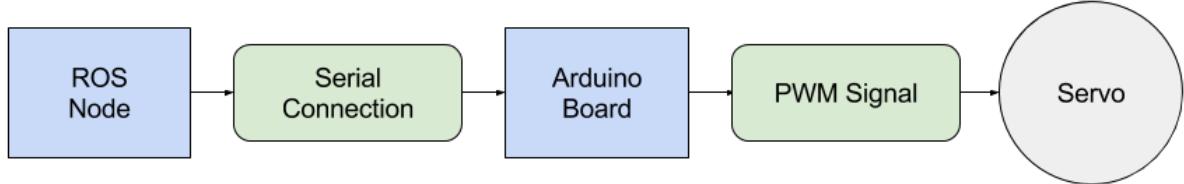


Abbildung 28: Ansteuerung der Servos via Arduino Board

Ein Arduino Board kann von uns programmiert werden und steuert dann einzig dieses Programm. So erreichen wir eine ausreichende Echtzeitgarantie für das generieren der PWM Signale.

Die Arduino Standard Library [69] stellt bereits ein Interface bereit zur Generierung von PWM-Signalen auf den PWM Output Pins. Über eine serielle Schnittstelle kommuniziert das Arduino Board mit dem Jetson Board.

Vorteile	Nachteile
<ul style="list-style-type: none"> + <u>Ausreichende Echtzeitgarantie</u> Mit diesem Ansatz erhalten wir eine ausreichende Echtzeitgarantie, was ein akkurates und zuverlässiges PWM-Signal produziert. 	<ul style="list-style-type: none"> - <u>Zusätzliche Hardware notwendig</u> Mit diesem Ansatz ist ein extra Arduino Board notwendig.

3.10.2.3 Fazit

Die Servo Motoren der Pan&Tilt-Vorrichtung können über die Digital Output Pins vom Kobuki nicht zuverlässig angesteuert werden. Daher haben wir uns für die Variante mit dem Arduino Board entschieden

3.11 Face Tracker

3.11.1 Einleitung

Aufgabe der Face Tracker Komponente ist es, die Pan&Tilt-Vorrichtung so zu steuern, dass ein erkanntes Gesicht jeweils möglichst im Zentrum des ZED-Bildes liegt. Gesichter bleiben somit länger im Sichtbereich von IANA, was zu einer längeren Gesichtserkennung führt. Zudem schaut IANA so Menschen nach und nimmt «Blickkontakt» mit ihnen auf, dies lässt IANA menschlicher wirken.

Als Input dienen die Positionen der erkannten Gesichter von der «Person Detection» Komponente. Falls mehrere Gesichter in einem Bild erkannt werden, wählt der Face Tracker das vorderste Gesicht für das Tracking aus (die Gesichter sind nach der Fläche der erkannten Bounding Box sortiert).

3.11.2 Annahmen

Wir treffen folgende Annahmen zur Vereinfachung der Winkelberechnung:

- Die Pan- und Tilt-Rotationsachsen schneiden sich im Mittelpunkt der Kamera.
- Das Kamerabild entspricht einem dem Sichtbereich entsprechenden Ausschnitt der Projektion der Umwelt auf eine Kugel.

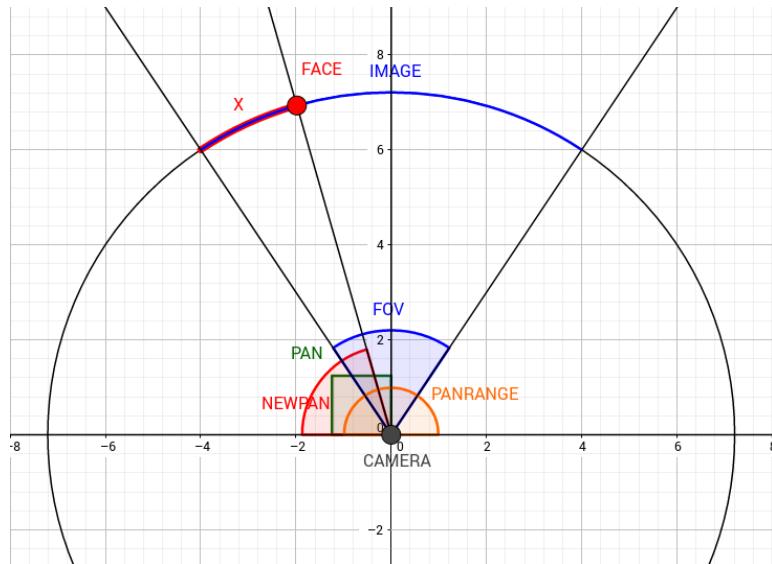
Die Annahmen geben uns eine gute Näherung zum wirklichen Winkel, was ausreichend ist. Der exakt berechnete Winkel wäre sowieso veraltet und daher auch nicht exakt, wenn das Gesicht relative zur ZED in Bewegung ist.

3.11.3 Berechnung der Pan- und Tilt-Winkel

Gegeben seien ein Kamerabild I_t zum Zeitpunkt t und die Position $F_t := (x_t, y_t)$ des vordersten erkannten Gesichtes in I_t . Weiter seien $res := (res_{horizontal}, res_{vertical})$ die Bildgrösse der Kamerabilder, $fov := (fov_{horizontal}, fov_{vertical})$ das Sichtfeld der Kamera sowie $state_{t-1} := (pan_{t-1}, tilt_{t-1})$ die Pan&Tilt-Winkel zum Zeitpunkt $t - 1$ gegeben.

So lassen sich unter Berücksichtigung der getroffenen Annahmen die Pan&Tilt-Winkel $state_t$ für das Gesicht F_t wie folgt berechnen:

$$state_t := (pan_t, tilt_t) = state_{t-1} + \frac{F_t \cdot fov}{res} - \frac{fov}{2}$$



Legende	
I_t	IMAGE
F_t	FACE
x_t	X
fov	FOV
pan_{t-1}	PAN
pan_t	NEWPAN

Abbildung 29: Schematische Darstellung der Pan-Ausrichtung

3.12 Benutzerschnittstelle

3.12.1 Einleitung

IANA erhält von aussen Befehle (externe Tasks), welche das Tasksystem des IANA Controllers abarbeitet. Diese Befehle werden in der jetzigen Version von IANA über eine Webschnittstelle an IANA

mitgeteilt, sollten allerdings in Zukunft über eine NLP-Schnittstelle mitgeteilt werden. Es ist daher wichtig, dass diese Komponente ohne grossen Aufwand austauschbar ist.

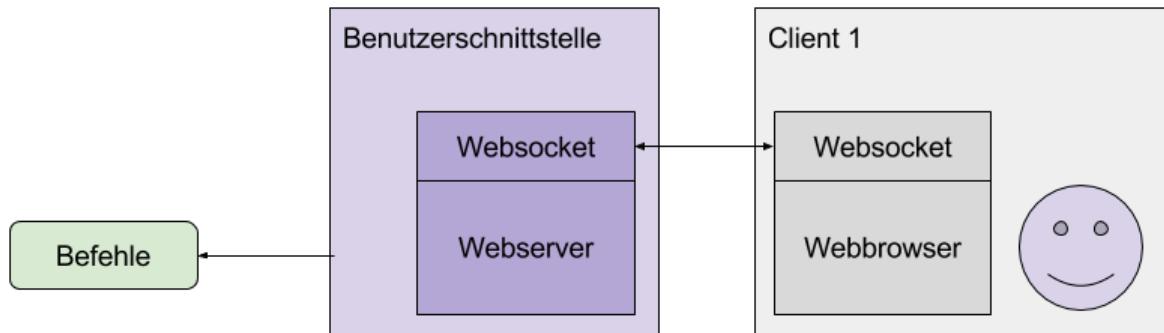


Abbildung 30: Übersicht über Benutzerschnittstelle Komponente

3.12.1.1 Webserver

Die Benutzerschnittstelle ist eine ROS Node die einen Webserver startet und von Clients über einen Webbrowser mit Websocket Support aufgerufen werden kann. Über Formulare und eine visuelle Karte kann der Benutzer Befehle an den Server schicken, welche in der Benutzerschnittstelle in ROS Messages umgewandelt werden, auf welche der IANA Controller hört.

Für den Webserver wurde das schlanke Python Autobahn Framework verwendet. Zur Kommunikation zwischen Server und Client werden beidseitig immer zuerst der Befehl und anschliessend die dazugehörigen Daten mitgeteilt – ein schlichtes Protokoll, aber für unseren Anwendungsfall ausreichend.

3.12.1.2 Benutzerschnittstelle ändern

Ohne weiteren Aufwand kann die Benutzerschnittstelle ersetzt werden, indem die Node «Benutzerschnittstelle» durch eine neue Benutzerschnittstelle ersetzt wird und dieselben Befehle (von uns definierte ROS Messages) an die richtigen ROS-Topsics veröffentlicht.

3.13 Texterkennung

Die Texterkennung nimmt ein Bild entgegen und gibt aus, welcher Text sich darauf befindet. Sie besteht aus zwei Schritten:

1. Text detection: Wo im Bild befindet sich Text?
2. Text recognition: Was für Text befindet sich dort?

Texterkennung mit starken Annahmen über die Schriftart des Textes und den Hintergrund des Textes ist ein meist einfaches Problem. Die generelle Texterkennung ist ein schwierigeres Problem und es gibt in den letzten Jahren viele verschiedene Ansätze mit vielversprechenden Resultaten [70].

Um die Texterkennung für uns zu vereinfachen, treffen wir Annahmen: Der Text steht auf den FHNW Türschildern, also Schriftart und Hintergrund sind bekannt und der Text ist korrekt ausgerichtet. Diese Annahme führt dazu, dass IANA angepasst werden muss, wenn sich die Schriftart oder der Hintergrund der FHNW Türschilder verändert.

3.13.1 FHNW Türschilder

Die Türschilder der FHNW haben alle das gleiche Muster im Hintergrund und die gleiche Schriftart (Siehe Abbildung 28).

Es gibt zwei stark unterschiedliche Textgruppen:

- Die Raumnummer, hier: «5.2C». Die Raumnummer ist grösser und befindet sich auf einem gekachelten Hintergrund
- Die Namen, hier «Hochschule für Technik FHNW», «Institut für 4D-Technologien» und «Studiengang Informatik». Die Namen sind kleiner und befinden sich auf einem einheitlichen Hintergrund.

Obwohl die Textgruppen unterschiedlich sind, können sie mit dem gleichen Prinzip erkannt werden. Letztendlich unterscheiden sich die beiden Algorithmen für die unterschiedlichen Textgruppen nur leicht voneinander.

Nun wird zuerst das Prinzip erklärt und anschliessend die jeweiligen Abweichungen für die einzelnen Textgruppen.

3.13.2 Prinzip

3.13.2.1 *Text Detection*

Die Idee der Text Detection ist es auszunutzen, dass Texte viele verschiedene Kanten nahe beieinanderhaben. Wir können diese Kanten erkennen, ineinander verwischen und anschliessend die daraus entstehende Fläche als Kontur finden [71].

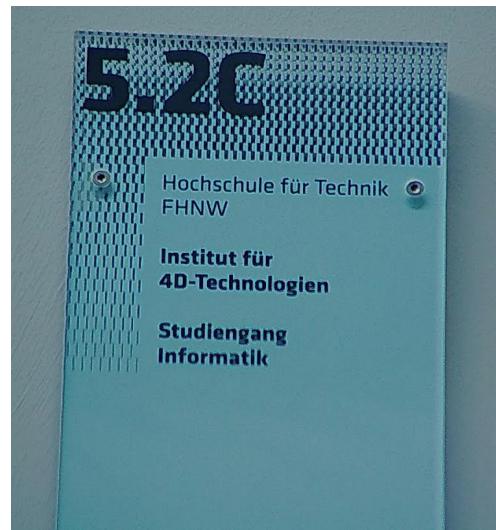


Abbildung 31: Aufnahme FHNW Türschild mit DLink Kamera

Die einzelnen Schritte des Algorithmus:

1. Umwandlung zum Binärbild

Das Farbbild wird zu einem Graustufenbild und anschliessend zu einem Binärbild umgewandelt, da wir binäre Morphologische Filter darauf anwenden wollen.

2. Hintergrund-Muster entfernen

Wenn nötig muss ein Hintergrundmuster entfernt werden. Dafür wird ein Morphologischer Filter auf dem Binärbild angewandt.

3. Detektieren der Kanten

Nun werden die Kanten im Bild erkannt.

4. Verwischen der Kanten

Mit einem Closing werden die erkannten Kanten der Texte, sowie die Wörter der Texte zu einer gemeinsamen weissen Fläche geschlossen. Dies Nutzt die Grundidee dieses Ansatzes: Texte haben viele nahe beieinanderstehende Kanten.

5. Detektieren der Konturen

Schlussendlich werden alle Konturen im finalen Bild als Bounding boxen erkannt und untersucht. Durch das Verwischen der Kanten sollten auch die Textstellen als einzelne Konturen erkannt werden. Es gibt viele Konturen im Bild, doch kann man die sinnlosen Konturen herausfiltern: Konturen müssen eine minimale Breite und Höhe aufweisen sowie einen Anteil an Vordergrund-Pixel innerhalb der Bounding box besitzen.

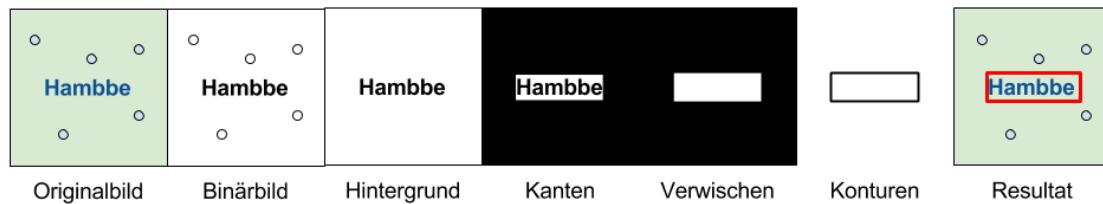


Abbildung 32: Das Text Detection Prinzip in der Übersicht

3.13.2.2 Text Recognition

Für die Texterkennung wird Tesseract genutzt. Wir führen Tesseract mit den gefundenen Textstellen aus und erhalten als Resultat die erkannten Texte. Wenn nötig kann man zuerst auf den Textstellen ein Pre-processing durchführen. Zum Schluss gibt es ein Post-processing auf allen erkannten Texten. Nicht erkannte Texte (leere Texte) werden herausgefiltert, da wir sie als ein False Positive der Text detection vermuten. Die einzelnen Schritte des Algorithmus:

1. Textstelle aus Bild schneiden

2. Pre-processing auf Textausschnitt

Wenn nötig kann ein Pre-processing auf den Ausschnitt des Bildes ausgeführt werden.

Beispielsweise kann der Hintergrund entfernt werden.

3. Texterkennung mit Tesseract

Für die Texterkennung verwenden wir Tesseract.

4. Post-processing auf alles erkannten Texten

Zum Schluss kann, wenn nötig, ein Post-processing auf allen Texten ausgeführt werden.

3.13.3 Namens-Erkenner

Der Namens-Erkenner erkennt die Textgruppe der Namen auf den FHNW Türschilder. Für die Text detection und Text recognition verwendet er das zuvor beschriebene Prinzip, hier sind lediglich die Abweichungen beschrieben.

3.13.3.1 *Text detection*

1. Umwandlung zum Binärbild
2. Hintergrund-Muster entfernen
Hier nicht notwendig, da die Namen bereits auf einem einheitlichen Hintergrund stehen.
3. Detektieren der Kanten
4. Verwischen der Kanten
5. Detektieren der Konturen



Abbildung 33: Beispiel für die Erkennung von Namen auf einem FHNW Türschild

3.13.3.2 *Text recognition*

1. Pre-processing auf Textausschnitt

Bild zu Binärbild umwandeln. Ausrichtung des Textes korrigieren (Siehe 3.13.5 Ausrichtung eines Textes).



Abbildung 34: Ausrichtung eines erkannten Wortes

2. Texterkennung mit Tesseract
3. Post-processing auf alles Texten

Die zusammengehörenden Zeilen müssen noch miteinander verbunden werden.

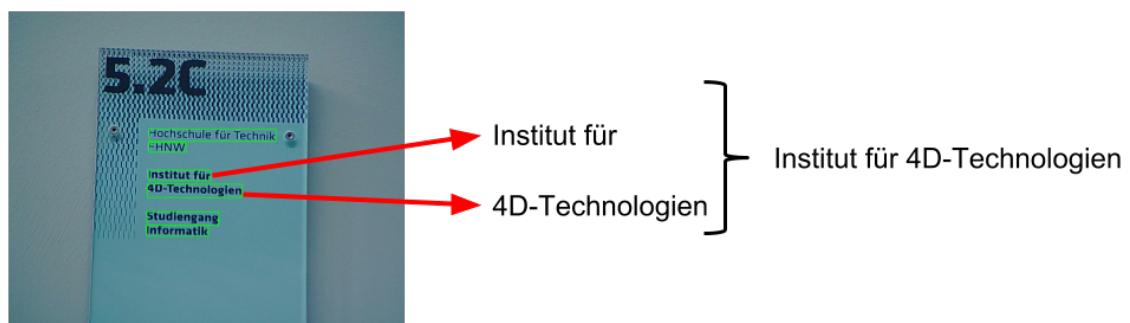


Abbildung 35: Verbinden von zusammengehörenden Zeilen

Dafür nutzen wir wieder die Annahme der FHNW Türschilder aus: Zeilen stehen linksbündig untereinander. Bei zusammenhängenden Zeilen ist die Y-Distanz geringer, als bei nicht zusammenhängenden Zeilen. Eine Zeile muss also an eine andere Zeile angehängt werden, wenn die linke obere Ecke nahe der linken unteren Ecke der anderen Zeile ist.

Um herauszufinden welche Zeilen zusammenhängen vergleichen wir einfach jede Zeile mit jeder. Dabei merken wir uns für jede Zeile den gefundenen Nachfolger. Wir gehen davon aus, dass keine Zeile zwei Nachfolger und keine Zeile zwei Vorgänger hat. Wären mehrere Nachfolger oder Vorgänger vorhanden, würde der Algorithmus nicht das gewünschte Resultat liefern.

Zum Schluss verbinden wir die Texte anhand der gefundenen Nachfolgern.

3.13.4 Raumnummer-Erkennen

Der Raumnummer-Erkennen erkennt die Textgruppe der Raumnummern auf den FHNW Türschildern. Für die Text detection und Text recognition verwendet er das zuvor beschriebene Prinzip, hier sind lediglich die Abweichungen beschrieben.

3.13.4.1 *Text detection*

Die Text detection besteht aus folgenden Schritten:

1. Umwandlung zum Binärbild
 2. Hintergrund-Muster entfernen
- Um das Muster zu einer gemeinsamen Fläche zu schliessen, wird ein Closing angewandt. Nun ist die Raumnummer isoliert auf dem weissen Hintergrund.
3. Detektieren der Kanten
 4. Verwischen der Kanten
 5. Detektieren der Konturen

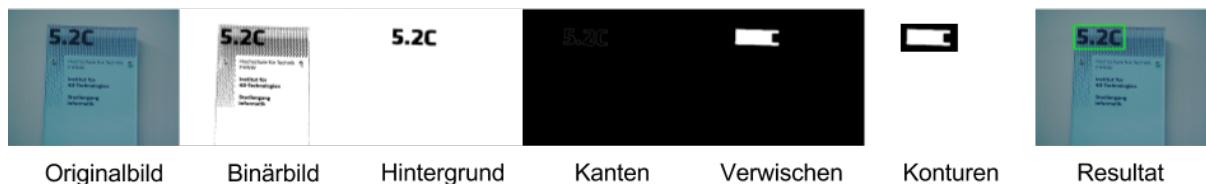


Abbildung 36: Beispiel für die Erkennung von Raumnummer auf einem FHNW Türschild

3.13.4.2 *Text recognition*

1. Pre-processing auf Textausschnitt
Bild zu Binärbild umwandeln. Hintergrund-Muster entfernen (wie bei der «Text detection»). Ausrichtung des Textes korrigieren (Siehe 3.13.5 Ausrichtung eines Textes).



Abbildung 37: Umwandlung zu Binärbild und Ausrichtung

2. Texterkennung mit Tesseract
3. Post-processing auf alles Texten

Hier ist kein Post-Processing nötig, da die Raumnummer in einer Zeile steht.

3.13.5 Ausrichtung eines Textes

Der Algorithmus um den Text auszurichten haben wir vom Internet übernommen [72] und leicht angepasst. Die Schwierigkeit ist es den Winkel herauszufinden, denn die letztliche Rotation ist eine einfache Affine Abbildung. Der Winkel wird mit Hilfe des minimalen angewinkelten Rechteckes um die Pixelkoordinaten des Textes berechnet. Den Winkel des gefundenen Rechteckes wäre bei einem ausgerichteten Text 90 Grad. Also rotieren wir das Bild um den fehlenden Betrag.

Beispiel:



Abbildung 38: Beispiel für die Ausrichtung des Textes «FHNW»

3.13.6 Ausrichten des Roboters

Wegen der limitierten Sichtweite der D-Link Highspeed Kamera mit Objektiv ist es sehr unwahrscheinlich, dass IANA zufällig ein Türschild findet. IANA muss daher Türschilder im ZED Bild erkennen und sich danach ausrichten. Diese Komponenten wurde aus Zeitgründen nicht umgesetzt. Unsere Ideen für diese Komponenten findet man im Kapitel «7.4 Erkennung von Türschilder und Ausrichtung des Roboters».

4 Umsetzung

4.1 Sensoren

4.1.1 D-Link Highspeed Kamera Wrapper

Für die D-Link Highspeed Kamera programmierten wir einen einfachen Wrapper, der die Bilder von der Netzwerkkamera holt und sie über eine ROS-Topic ins ROS-System speist.

4.1.1.1 ROS Package Informationen

ROS-Package Name: camera_wrapper

ROS-Node Name: url_wrapper

4.1.1.1.1 Publizierte ROS-Topics

/image (sensor_msgs::Image)

4.2 SLAM

4.2.1 ROS Package Informationen

ROS-Package Name: gmapping

ROS-Node Name: slam_gmapping

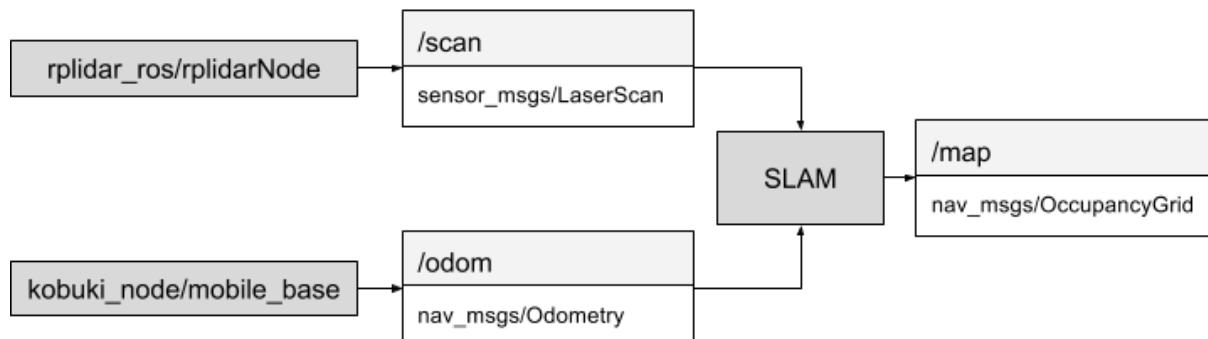


Abbildung 39: Abonnierte und publizierte ROS Topics der SLAM Komponente

Für den GMapping SLAM-Algorithmus existiert bereits eine Umsetzung als ROS Node [73].

Die meisten Parameterwerte für die GMapping Node wurden von turtlebot_navigation übernommen [74]. Folgende Werte wurden verändert:

4.2.1.1 KinectOne Parameter

Parameter	Wert	Grund
/maxUrangle	5.0 statt 6.0	Das Tiefenbild der KinectOne wird ab einer Tiefe von fünf Meter ungenauer.
/maxRange	7.0 statt 8.0	Die KinectOne hat eine Reichweite von 7 Metern.

4.2.1.2 RPLidar Parameter

Parameter	Wert	Grund
/maxUrange	5.5 statt 6.0	Der RPLidar Sensor
/maxRange	6.0 statt 8.0	Der RPLidar Sensor hat eine Reichweite von 6 Metern.
/particles	80 statt 120	Da wir nun einen besseren Sensor haben, ist die Proposal Distribution von GMapping sicherer und braucht weniger Partikel.
/minimumScore	0 statt 200	Der Minimum-Score muss gesetzt werden, wenn ein ungenauer Sensor verwendet wird, da sonst falsche Scan Matchings passieren können, was den Roboter sprunghaft versetzen kann. Der RPLidar ist genug genau, dass wir /minimumScore auf 0 setzen können.
/linearUpdate	1.0 statt 0.5	Wir verarbeiten nun nur noch all Meter einen Laserscan, da der RPLidar besser misst, ist sich GMapping sicherer mit der Karte und veröffentlicht bereits genug oft neue Karten (Siehe 5.3.4 Entwicklung der Karte).
/angularUpdate	0.5 statt 0.436	Was für /linearUpdate gilt, gilt auch für /angularUpdate.

4.3 Navigation

4.3.1 ROS Package Informationen

4.3.1.1 Parameter

Die Navigation ist über den Navigation Stack umgesetzt, ein externes ROS Paket. Allerdings ist der Navigation Stack und sein Local- sowie Global-Planner und deren Costmap-Berechnung über eine Vielzahl von Parameter konfigurierbar:

Die Parameter können über folgende YAML-Dateien [75] gesetzt werden:

- `costmap_common_params.yaml`
Gemeinsame Parameter der Globalen und der Lokalen Costmap-Berechnung
- `global_costmap_params.yaml`
Parameter der Globalen Costmap-Berechnung
- `local_costmap_params.yaml`
Parameter der Lokalen Costmap-Berechnung
- `dwa_local_planner_params.yaml`
Parameter des DWA Local Planner
- `global_planner_params.yaml`
Parameter des Globalen Planner

- `move_base_params.yaml`
Parameter über den Navigation Stack, wie welchen Local- und Global-Planner .
- `navfn_global_planner_params.yaml`
Parameter des navfn Global Planner

Die meisten Parameterwerte wurden von `turtlebot_navigation` übernommen [76]. Es folgen alle Abweichungen von der `turtlebot_navigation` Parametrisierung. Für die Wahl der korrekten Parameter half uns der Navigation Tuning Guide aus dem ROS Wiki [77] und der ROS Navigation Tuning Guide [64].

4.3.1.2 `costmap_common_params.yaml`

Parameter	Wert	Grund
<code>/scan/min_obstacle_height</code>	0.0 anstatt 0.25	Wir interessieren uns auch für tiefe Hindernisse.
<code>/scan/max_obstacle_height</code>	0.6 anstatt 0.3	Wir interessieren uns für Hindernisse die bis zu 60 Zentimeter hoch sind, da IANA 60 Zentimeter hoch ist.
<code>/inflation_layer/cost_scaling_factor</code>	10.0 anstatt 5.0	Die beiden Parameter helfen dem Navigation Stack bei der Berechnung des Inflation Layers der Costmaps. Bei der Global Costmap von der Karte des SLAM Algorithmus werden Hindernisse aufgeblasen. Bei der Local Costmap wird der aktuelle Laserscan aufgeblasen und eingebaut. <code>/cost_scaling_factor</code> gibt an, wie stark Kosten skaliert werden.
<code>/inflation_layer/inflation_radius</code>	0.25 anstatt 0.5	<code>/inflation_radius</code> gibt an, mit welchem Radius um ein Hindernis skaliert werden soll.

4.3.1.3 `global_costmap_params.yaml`

Parameter übernommen von `turtlebot_navigation`.

4.3.1.4 `local_costmap_params.yaml`

Parameter übernommen von `turtlebot_navigation`.

4.3.1.5 `dwa_local_planner_params.yaml`

Parameter übernommen von `turtlebot_navigation`.

4.3.1.6 `global_planner_params.yaml`

Parameter übernommen von `turtlebot_navigation`.

4.3.1.7 `move_base_params.yaml`

Parameter übernommen von `turtlebot_navigation`.

4.3.1.8 `navfn_global_planner_params.yaml`

Parameter übernommen von `turtlebot_navigation`.

4.4 Controller

4.4.1 Ros Package Informationen

4.4.1.1 Abonnierte ROS-Topics

/iana/user_command/explore (iana_controller::Explore)
/iana/user_command/explore_random (iana_controller::ExploreRandom)
/iana/user_command/go_to (iana_controller::GoTo)
/iana/person_detection/unknown/entered (iana_person_detection::UnknownPersonEntered)
/iana/person_detection/known/entered (iana_person_detection::KnownPersonEntered)
/iana/person_detection/unknown/left (iana_person_detection::UnknownPersonLeft)
/iana/person_detection/known/left (iana_person_detection::KnownPersonLeft)

4.4.1.2 Verwendete ROS-Action-Topics

/iana/navigation/explore (iana_navigation::ExploreAction)
/iana/navigation/explore_random (iana_navigation::ExploreRandomAction)
/iana/navigation/go_to (iana_navigation::GoToAction)
/iana/speech/say (iana_speech::SayAction)

4.4.2 ROS Messages (Commands) zu Tasks

Der Controller übersetzt gemäss folgenden Regeln ROS Messages zu Task:

ROS Message (Commands)	Task
Explore.msg std_msgs/Time until	Explore (s)
ExploreRandom.msg std_msgs/Time until	ExploreRandom(s)
Goto.msg geometry_msgs/PoseStamped target_pose	Goto (x, y, q, qw)
KnownPersonEntered.msg iana_person_data/Person person	Greet (bp) bp ist bekannt
UnknownPersonEntered.msg uint32 person_id sensor_msgs/Image preview_image FaceVector[] face_vectors	Greet (up) up ist unbekannt
UnknownPersonEntered.msg uint32 person_id sensor_msgs/Image preview_image FaceVector[] face_vectors	GetToKnow(up)
UnknownPersonLeft.msg uint32 person_id	

4.4.3 Task Interrupt Regeln

Tasks lassen sich von anderen Tasks wie folgt unterbrechen:

Task	Unterbrechbar durch
Explore	Alle Tasks
ExploreRandom	Alle Tasks
Goto	Alle Tasks, ausser andere Goto Tasks
GreetKnown	Alle Tasks
GreetUnknown	Alle Tasks
GoodbyeKnown	Alle Tasks
GoodbyeUnknown	Alle Tasks
GetToKnow	Alle Tasks

4.5 IANA Navigation

4.5.1 ROS Package Informationen

ROS-Package Name: iana_navigation

ROS-Node Name: iana_navigation_node

4.5.1.1 *Publizierte ROS-Topics*

/iana/driver_random/enable (std_msgs::Empty)
/iana/driver_random/disable (std_msgs::Empty)
/iana/driver_explore/enable (std_msgs::Empty)
/iana/driver_explore/disable (std_msgs::Empty)

4.5.1.2 *Publizierte ROS-Action-Topics*

/move_base/goal (move_base_msgs::MoveBaseActionGoal)

4.5.1.3 *Verwendete ROS-Action-Topics*

/iana/navigation/explore (iana_navigation::ExploreAction)
/iana/navigation/explore_random (iana_navigation::ExploreAction)
/iana/navigation/go_to (iana_navigation::GoToAction)

4.6 Kobuki

4.6.1 ROS Package Informationen

ROS-Package Name: kobuki_node

ROS-Node Name: kobuki_node

Die Kobuki ROS-Node ist im ROS-Wiki ausführlich beschrieben [78]. Wir verwenden die Standardparametrisierung.

4.7 Collision Detector

4.7.1 ROS Package Informationen

ROS-Package Name: iana_collision_detector

ROS-Node Name: look_ahead_rplidar, look_ahead_kinect_one

4.7.1.1 *Publizierte ROS-Topics*

/iana/collision_detector/collision_ahead (std_msgs::Float32)

4.7.1.2 *Publizierte ROS-Topics*

/scan (sensor_msgs::LaserScan)

/lookAhead/considered_scan (sensor_msgs::LaserScan)

4.8 Explorer

4.8.1 Random Explorer

4.8.1.1 *ROS Package Informationen*

ROS-Package Name: iana_driver

ROS-Node Name: driver_random_node

4.8.1.1.1 Parameter

Name	Standardwert	Beschreibung
min_linear_velocity	0.1	Minimale Lineare Geschwindigkeit in m/s, falls die Minimaldistanz zu einem Hindernis unter collision_warn_threshold fällt.
medium_linear_velocity	0.25	Mittlere Lineare Geschwindigkeit in m/s, falls die Minimaldistanz zu einem Hindernis unter collision_info_threshold fällt.
max_linear_velocity	0.5	Maximale Lineare Geschwindigkeit in m/s, falls die Minimaldistanz unter keinen der Thresholds gefallen ist.
min_angular_velocity	0.1	Minimale Rotations-Geschwindigkeit in m/s, initiale Geschwindigkeit bei Rotationsmanöver und Geschwindigkeit unmittelbar vor Manöverende. Während dem Manöver wird bis auf maximal max_angular_velocity beschleunigt.
max_angular_velocity	1	Die maximale Rotationsgeschwindigkeit bei einem Rotationsmanöver
linear_acceleration	0.2	Die Lineare Beschleunigung in $\frac{m}{s^2}$
angular_acceleration	0.2	Die Rotationsbeschleunigung in $\frac{deg}{s^2}$

collision_info_threshold	2.3	Threshold in m, ab dem eine Kollision absehbar ist und die Geschwindigkeit etwas verringert werden sollte (medium_velocity).
collision_warn_threshold	1.3	Threshold in m, ab dem eine Kollision nahe bevorsteht und die Geschwindigkeit stark verringert werden sollte (medium_velocity).
collision_ahead_threshold	0.8	Threshold in m, ab dem eine Kollision unmittelbar bevorsteht und sofort angehalten werden muss

Die Standardwerte sind für den RPLidar Sensor zusammen mit Kobuki optimiert. Die Thresholds für die Mindestdistanzen zu den Hindernissen sind auf den RPLidar Sensor abgestimmt. Die Geschwindigkeiten sind so gewählt, dass der Roboter möglichst sanft beschleunigt und bremst.

4.8.1.1.2 Publizierte ROS-Topics

/mobile_base/commands/velocity (geometry_msgs::Twist)

4.8.1.1.3 Abonnierte ROS-Topics

/iana/driver_random/enable (std_msgs::Empty)
 /iana/driver_random/disable (std_msgs::Empty)
 /collision_ahead (std_msgs::Float32)

4.8.2 Rational Explorer

4.8.2.1 ROS Package Informationen

ROS-Package Name: iana_driver

ROS-Node Name: driver_explore_node

4.8.2.1.1 Parameter

Name	Standardwert	Beschreibung
frontier_selection	random	«random» oder «closest» Es wird entweder ein Zufälliger Front-Punkt ausgewählt (random) oder der nächste genommen (closest)
min_distance	1	Minimum Distanz in Meter von der aktuellen Roboterposition zum nächsten Front-Punkt
nav_max_tries	3	Maximale Anzahl Versuche, ermittelte Front-Punkte mittels Navigation-Stack zu erreichen. Wird diese Anzahl überschritten wechselt der Rational Explorer vorübergehend in den Random-Zustand.

4.8.2.1.2 Publizierte ROS-Action-Topics

/move_base/goal (move_base_msgs::MoveBaseActionGoal)

4.8.2.1.3 Abonnierte ROS-Topics

/iana/driver_explore/enable (std_msgs::Empty)
 /iana/driver_explore/disable (std_msgs::Empty)
 /map (nav_msgs::OccupancyGrid)
 /odom (nav_msgs::Odometry)

4.9 Person Detection

4.9.1 ROS Package Informationen

ROS-Package Name: iana_person_detection

ROS-Node Name: person_detection

4.9.1.1 Parameter

Name	Standardwert	Beschreibung
start_x	-1	Möchte man nicht im ganzen Bild die Gesichtsdetektion durchführen, sondern nur in einem Teil des Bildes, kann mit diesen Parametern den gewünschten Bereich definieren.
end_x	700	
start_y	0	Der Wert von -1 in start_x zeigt an, dass das gesamte Bild verwendet werden soll.
end_y	300	
scale_factor	1	Ein Faktor mit welchem das Bild herunterskaliert wird, bevor die Gesichtsdetektion durchgeführt wird. Somit kann wichtige Performance gewonnen werden, zum Preis einer vielleicht schlechteren Gesichtsdetektion.
ignore_images_older_than	1	Die Personenerkennung sollte möglichst in Echtzeit ablaufen. Daher werden zu alte Bilder schon zu Beginn verworfen, da wahrscheinlich bereits ein neueres Bild vom Sensor in der Subscriber Queue ist. Die Queue Grösse auf 1 zu setzen, funktioniert leider nicht, da teilweise übertragene Nachrichten überleben und TCP Pakete vom Betriebssystem gebuffert werden können. Die Einheit dieses Parameters ist Sekunden.
min_confidence_update	0.9	Minimaler Confidence Wert für einen neuen Feature Vektoren, dass er in die gespeicherten Feature Vektoren eines Gesichtes aufgenommen wird
person_max_feature_vector_count	30	Die Maximale Anzahl Feature Vektoren, die für ein Gesicht gespeichert werden. Wird dieser

		Überschritten, wird der älteste Feature Vektor gelöscht.
--	--	--

4.9.1.2 *Publizierte ROS-Topics*

/iana/faces_detected (iana_person_detection::FaceBoundingBoxes)
 /iana/person_detection/known/entered (iana_person_detection::KnownPersonEntered)
 /iana/person_detection/unknown/entered (iana_person_detection::UnknownPersonEntered)
 /iana/person_detection/known/left (iana_person_detection::KnownPersonLeft)
 /iana/person_detection/unknown/left (iana_person_detection::UnknownPersonLeft)
 /new_person (iana_person_data::Person)
 /updated_person (iana_person_data::Person)

4.9.1.3 *Abonnierte ROS-Topics*

/face_image (sensor_msgs::Image)
 /person_image (sensor_msgs::Image)

4.9.1.4 *Verwendete ROS-Services*

/get_all_persons (iana_person_data::GetAllPersons)
 /update_person_features (iana_person_data::UpdatePersonFeatures)

4.9.1.5 *Persistieren der vorgestellten Personen*

Die «Person Data» Komponente ist mit einer sqlite Datenbank implementiert. Die dazugehörige .db Datei liegt unter «/usr/local/iana/database.db».

Möchten die Personendaten und Gesicht-Daten verworfen werden, kann einfach diese Datei gelöscht werden.

4.10 Face Tracker

4.10.1 ROS Package Informationen

ROS-Package Name: iana_face_tracker

ROS-Node Name: iana_face_tracker_node

4.10.1.1 *Parameter*

Name	Standardwert	Beschreibung
fov_h	110	Horizontales Sichtfeld der Kamera in Grad
fov_v	54	Vertikales Sichtfeld der Kamera in Grad
camera_img_width	1280	Kamerabildbreite
camera_img_height	720	Kamerabildhöhe

hold_position_time	3.0	Zeit in Sekunden, die die Pan & Tilt an einer neu eingenommenen Position auf neue Gesichtsinformationen wartet, bis die Vorrichtung in die Ursprungsausrichtung zurück rotiert wird.
--------------------	-----	--

Die Standardwerte sind die optimalen Werte für die ZED im 720p HD-Modus.

4.10.1.2 *Publizierte ROS-Topics*

/iana/camera/set_pan_tilt (iana_camera_pan_tilt::PanTilt)

4.10.1.3 *Abonnierte ROS-Topics*

/iana/driver_explore/enable (std_msgs::Empty)
/iana/driver_explore/disable (std_msgs::Empty)
/iana/faces_detected (iana_person_detection::FaceBoundingBoxes)
/iana/camera/pan_tilt (iana_camera_pan_tilt::PanTilt)

4.11 Pan&Tilt-Vorrichtung

4.11.1 ROS Package Informationen

ROS-Package Name: iana_camera_pan_tilt

ROS-Node Name: arduino_controller

4.11.1.1 *Parameter*

Name	Standardwert	Beschreibung
initial_pan	90	Initiale Pan-Ausrichtung in Grad
initial_tilt	90	Initiale Tilt-Ausrichtung in Grad
serial_port	/dev/ttyACM0	Der Serial Port, an dem das Arduino Board angeschlossen ist
serial_baudrate	9600	Die Baudrate, mit der mit dem Arduino Board kommuniziert wird
init_timeout	5	Timeout für die Initialisierung der seriellen Schnittstelle
write_timeout	3	Timeout für das Senden von Nachrichten über die serielle Schnittstelle
read_timeout	1	Timeout für das Lesen von Nachrichten über die serielle Schnittstelle

4.11.1.2 *Publizierte ROS-Topics*

/iana/camera/pan_tilt (iana_camera_pan_tilt::PanTilt)

4.11.1.3 *Abonnierte ROS-Topics*

/iana/camera/set_pan_tilt (iana_camera_pan_tilt::PanTilt)

4.12 Benutzerschnittstelle

Die Benutzerschnittstelle zeigt die momentane Karte mit der aktuellen Roboterposition (1), ein Explore-Formular (2), ein Explore-Random-Formular (3), ein Goto-Formular (4), die aktuellen Tasks (5) an (Siehe Abbildung 40).

Wenn eine unbekannte Person erkannt wird, öffnet sich ein Popup, in welchem nach dem Namen gefragt wird. Über ein Input Feld kann dieser eingegeben und via Buttonklick bestätigt werden (Siehe Abbildung 41).

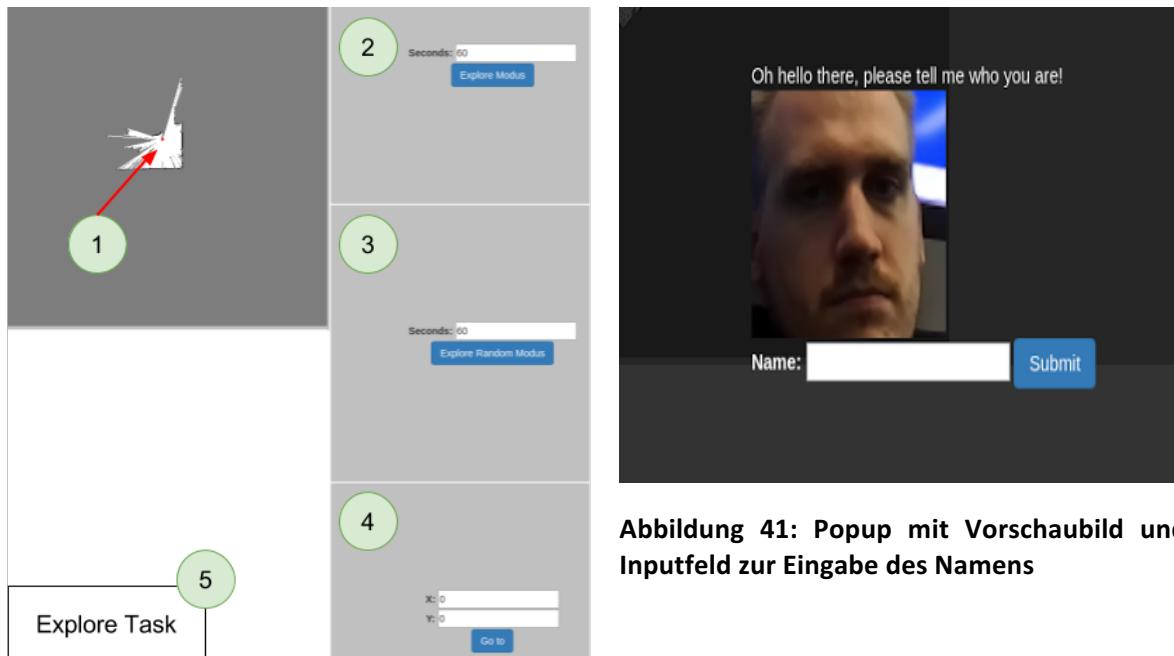


Abbildung 40: IANA Benutzerschnittstelle

4.12.1 Technologie

Für die Serverseite wurde das Python Framework Autobahn [79] verwendet.

4.12.2 Protokoll

Client und Server kommunizieren über HTTP Websockets. Darauf aufbauend nutzen wir ein eigen kreiertes Textprotokoll mit klar definierten Befehlen für Client zu Server und Server zu Client.

4.12.2.1 Client zu Server

Befehle	Beschreibung
name name	Der eingegebene Name der erkannten Person.
explore seconds	Erkunde für seconds Sekunden.
explore_random seconds	Erkunde rein zufällig für seconds Sekunden.
goto x,y,qx,qy,qz,qw	Gehe zur Position x, y mit der Orientierung des Quaternion qx, qy, qz, qw.

4.12.2.2 Server zu Client

Befehle	Beschreibung
request_name	Frage den Benutzer nach dem Namen.
refresh_map resolution,width,height,pose_x,pose_y,map	Eine neue Karte soll angezeigt werden. Resolution ist die Auflösung der Karte und pose_x und pose_y die Position der Karte im globalen TF Frame (gebraucht für GoTo Befehle). Und map sind width*height Zahlen mit 0 für frei, 1 für besetzt und -1 für unbekannt.
refresh_robot_position x,y,z	Die neue Roboter Position soll angezeigt werden.
refresh_tasks tasks	Eine Liste von den aktuellen Tasks. Wird veröffentlicht, wenn sich die Taskliste verändert.

4.13 Text Detection

4.13.1 ROS Package Informationen

ROS-Package Name: iana_text_recognition

ROS-Node Name: imvs_sign_text_recognition

4.13.1.1 Parameter

Name	Standardwert	Beschreibung
Namens-Erkennung		
name_detection/kernel_grad_shape	(2, 2)	Kernelform für die morphologische Kantenerkennung mit Rechteck. Der erste Wert ist die Breite, der zweite die Höhe des Rechteckes.
name_detection/kernel_close_shape	(20, 10)	Kernelform morphologischen der Verwischungsoperation.
name_detection/threshold_min_text_ratio	0.5	Wie viele Pixel müssen prozentual verschmierter Text sein.
name_detection/threshold_min_height	15	Wie Hoch muss eine erkannte rechteckige Kontur mindestens sein.
name_detection/threshold_min_width	50	Wie Breit muss eine erkannte rechteckige Kontur mindestens sein.

Raumnummer-Erkennung		
number_detection/kernel_remove_background_shape	(10, 10)	Kernelform für die morphologische Entfernung des Hintergrundes mit Rechteck.
number_detection/kernel_grad_shape	(2, 2)	Kernelform für die morphologische Kantenerkennung mit Rechteck.
number_detection/kernel_close_shape	(100, 50)	Kernelform für die morphologische Verwischungsoperation.
number_detection/threshold_min_text_ratio	0.5	Wie viele Pixel müssen prozentual im Bild verschmierter Text (Vordergrund) sein, so dass sie als Text gelten.
number_detection/threshold_min_height	80	Wie Hoch muss eine erkannte rechteckige Kontur mindestens sein, so dass sie als Text gilt.
number_detection/threshold_min_width	40	Wie Breit muss eine erkannte rechteckige Kontur mindestens sein, so dass sie als Text gilt.
Shared Parameters		
language	deu	Verwendete Sprache von Tesseract für die Texterkennung.

4.13.1.2 *Publizierte ROS-Topics*

/texts_in_image (iana_text_recognition::TextsInImage)

4.13.1.3 *Abonnierte ROS-Topics*

/image (sensor_msgs::Image)

4.14 IANA Starten

In ROS helfen Launch-Dateien (*.launch), um alle nötigen Nodes eines ROS Systems zu starten. Wir haben alle unsere Launch Dateien zentral in der iana_start_up Node unter /launch gelegt. Launch-Dateien mit Endung «startup» starten ein ganzes ROS System. Launch-Dateien ohne Endung «startup» werden, wenn nötig, von anderen Launch-Dateien importiert.

IANA startet man mit dem Befehl:

```
> rosrun iana_start_up startup.launch
```

4.14.1 Launch Dateien

Hier eine Liste in alphabetischer Reihenfolge der Launch Dateien.

Launch Datei	Beschreibung
iana.launch	Startet alle nötigen IANA Komponenten wie iana_person_data, iana_person_detection, usw.
kinect-one-gmapping.launch	Startet GMapping für die KinectOne als Tiefensensor.
kinect-one-startup.launch	Startet ein gesamte ROS System bestehend aus IANA, der ZED, der D-Link Highspeed Kamera, Kobuki, der KinectOne, GMMapping für die KinectOne, dem Collision Detector für die KinectOne und dem Navigation Stack.
rplidar-gmapping.launch	Startet GMMapping für den RPLidar als Tiefensensor.
rplidar-startup.launch	Startet ein gesamte ROS System bestehend aus IANA, der ZED, der D-Link Highspeed Kamera, Kobuki, dem RPLidar Sensor, GMMapping für den RPLidar, dem Collision Detector für die RPLidar und dem Navigation Stack.
rplidar.launch	Startet die RPLidar Wrapper Node.
startup-debug.launch	Startet startup.launch mit zwei Path Servern, die sich den gefahrenen Pfad des Roboters merken.
startup.launch	Startet rplidar-startup.launch
zed-gmapping.launch	Startet GMMapping für die ZED als Tiefensensor.
zed-startup.launch	Startet ein gesamte ROS System bestehend aus IANA, der ZED, der D-Link Highspeed Kamera, Kobuki, GMMapping für die ZED Kamera, dem Collision Detector für die KinectOne (auch einsetzbar für die ZED) und dem Navigation Stack.

4.15 Codebase

4.15.1.1 Komponente

Die Codebase enthält alle unsere entwickelten Softwarekomponenten. Jede Komponente ist in einem eigenen Verzeichnis abgelegt. In der folgenden Übersicht ist für jedes Verzeichnis die darin liegende Komponente beschrieben. Falls wir die Komponente bereits in einem Kapitel ausführlich beschrieben haben, verweisen wir darauf. Diese Zeilen sind grau hinterlegt.

Name	Beschreibung oder Verweis
camera_test	Speichert Bilder mit der D-Link Highspeed Kamera. Wurde gebraucht, um Testdaten für die Texterkennung aufzunehmen.

camera_wrapper	Siehe 4.1.1 D-Link Highspeed Kamera Wrapper
debug_camera	Published Bilder der Notebook Kamera. Wurde gebraucht, um die Gesichtserkennung lokal auf den Entwicklermaschinen zu testen und zu entwickeln.
debug_maps_saver	Speichert jede neu veröffentlichte Karte auf dem /map Topic in einen vorgegebenen Ordner. Wurde für die Validierung des SLAM Algorithmus verwendet.
debug_text_recognition	Zeichnet die Bounding boxen der erkannten Texte auf das Ursprungsbild und veröffentlicht dieses. So kann man leicht sehen, wo Texte erkannt wurden.
face_detection_validation	Angepasster Validierungstest für die Gesichtsdetektion aus dem IP5 Projekt.
frontier_detection_prototype	Prototype für den Frontier Detector verwendet im Explore Driver.
iana_arduino_programs	Die Servos der Pan&Tilt-Vorrichtung steuern wir über einen Arduino. In diesem Ordner liegt der Source Code für das Programm, was schliesslich auf dem Arduino läuft.
iana_camera_pan_tilt	Siehe 3.10 Pan&Tilt-Vorrichtung
iana_collision_detector	Siehe 3.7 Collision Detector
iana_controller	Siehe 3.5 Controller
iana_driver	Siehe 3.8 Explorer
iana_driver_old	Eine Implementierung unseres ersten Random Drivers. Wurde ersetzt durch iana_driver.
iana_driver_prototype	Für Odometrie-Tests wurde ein Programm geschrieben, welches klare Fahrbefehle ausführen kann.
iana_face_tracker	Siehe 3.11 Face Tracker
iana_image_capture	Speichert Bilder von einem ROS Topic auf die Festplatte. Wurde benötigt für Testaufnahmen mit der ZED.
iana_navigation	Siehe 3.6 IANA Navigation
iana_person_data	Siehe 3.9.3 Persistieren der vorgestellten Personen
iana_person_detection	Siehe 3.9 Person Detection
iana_speech	Zentrale Stelle die das Sprechen für IANA übernimmt.
iana_start_up	Siehe 4.14 IANA Starten

iana_text_recognition	Siehe 3.13 Texterkennung
iana_user_io	Siehe 3.12 Benutzerschnittstelle
ocr_prototype	Prototype für die Texterkennung. Aufnahmen für den Bericht wurden mit diesem Prototyp erstellt.

4.15.1.2 Zeilen an Code

Zur Übersicht vom Umfang des Projektes sind hier die Zeilen Programmiercode in der Codebase dargestellt.

Language	files	blank	comment	code
XML	21	238	709	34982
CSS	5	15	24	7381
Python	100	974	395	3276
Javascript	3	582	204	1611
C/C++ Header	12	216	25	603
CMake	19	616	2825	415
C++	9	99	9	404
HTML	1	61	0	361
YAML	9	54	78	164
Arduino Sketch	1	12	0	65
SUM:	180	2867	4269	49262

Abbildung 42: Anzahl Zeilen Code unseres Git Repositories (mit „colc \$(git ls-files)“ berechnet)

Bei den Ergebnissen muss beachtet werden, dass viel XML von den Modellen des HaarFaceDetectors kommt und dass CSS bis auf 56 Zeilen und alles JavaScript von der Library Bootstrap [80] kommt.

5 Test und Validation

5.1 Tiefenscan-Sensoren Evaluation

5.1.1 Einleitung

In dieser Arbeit kommen drei verschiedene Tiefensensoren zum Einsatz (Siehe 2.1 Hardware). In den hier beschriebenen Evaluationstests vergleichen wir, welcher Sensoren die besten Messdaten für IANA liefert. Dabei haben wir für Testszenarios in klar definierten Umgebungen die Tiefescans und die von GMapping generierte Map verglichen. Alle Tests werden in Innenräumen durchgeführt, was dem schlussendlichen Einsatzgebiet von IANA entspricht.

Aufgrund der Art und Weise, wie der RPLidar Sensor auf IANA montiert ist, entsteht von IANA aus gesehen gegen hinten ein toter Winkel, in dem keine Tiefendaten gemessen werden können. Eigentlich beträgt die Scanweite des Sensors jedoch 360°.

5.1.2 Testszenario 1

IANA befindet sich in einem 1.95m x 1.85m Raum, der über einen kleinen Spalt mit einem 0.4m x 0.6m Raum verbunden ist. Sie ist Richtung rechte obere Ecke ausgerichtet. Für jeden Sensor wird die entsprechende Sensoransteuerung zusammen mit GMapping solange ausgeführt, bis ein erstes Mal eine Map publiziert wird. Die resultierende Maps und die Tiefescans zum Zeitpunkt der Aufnahme werden unter den Sensoren miteinander verglichen.

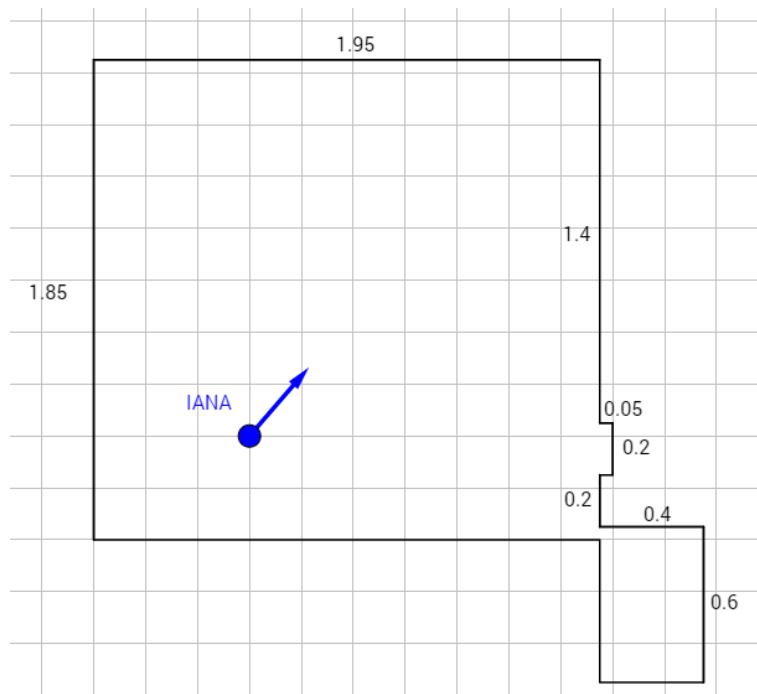
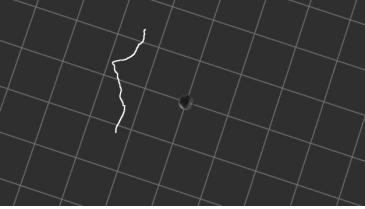
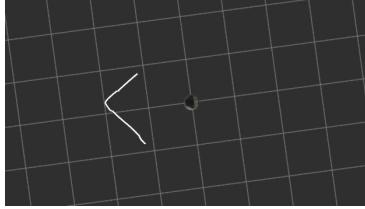
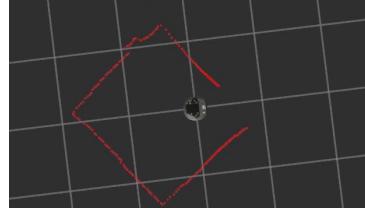
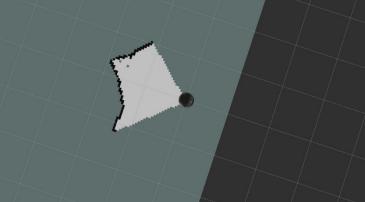
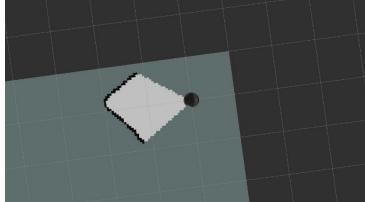


Abbildung 43: Tiefenscan-Sensoren Evaluation Testszenario 1 mit IANA

5.1.2.1 Tiefenscan

ZED	KinectOne	RPLidar
 <p>Abbildung 44: ZED Tiefenscan</p> <p>Die ZED fällt in diesem Vergleich ganz klar hinten ab. Die anvisierte Ecke ist zwar erkennbar, die eigentlich geraden Wände sind jedoch gegen aussen stark gekrümmmt. Ausserdem ist die Scanweite sehr eingeschränkt. Hohe Scandichte.</p>	 <p>Abbildung 45: KinectOne Tiefenscan</p> <p>Die anvisierte Ecke ist klar erkennbar und die anliegenden Wände gerade. Die Scandaten der Kinect sind auf die getestete Distanz exakt, die Scanweite ist jedoch sogar noch etwas mehr eingeschränkt als bei der ZED. Hohe Scandichte.</p>	 <p>Abbildung 46: RPLidar Tiefenscan</p> <p>Nahezu der ganze Raum ist mit einem Scan erfasst, die Scandaten sind auf die getestete Distanz exakt. Niedrigere Scandichte als bei ZED & Kinect.</p>

5.1.2.2 Map

ZED	KinectOne	RPLidar
 <p>Abbildung 47: ZED Map</p> <p>Die resultierende Map entspricht nicht der Realität, es tauchen analog zum Tiefenscan Krümmungen in eigentlich geraden Abschnitten auf. Die Map zeigt nur die anvisierte Ecke.</p>	 <p>Abbildung 48: KinectOne Map</p> <p>Die resultierende Map zeigt nur die anvisierte Ecke, ist jedoch genau und entspricht der Realität.</p>	 <p>Abbildung 49: RPLidar Map</p> <p>Die resultierende Map ist genau, entspricht der Realität und zeigt bis auf den toten Winkel direkt hinter dem Sensor den gesamten Raum. Die Map weist noch einige unbekannte Löcher auf, die auf die niedrigere Scandichte zurückzuführen sind.</p>

5.1.3 Testszenario 2

IANA befindet sich in einem 0.64m x 0.36m Raum, 15 cm vor einer Wand. Für jeden Sensor wird der Tiefenscan aufgezeichnet.

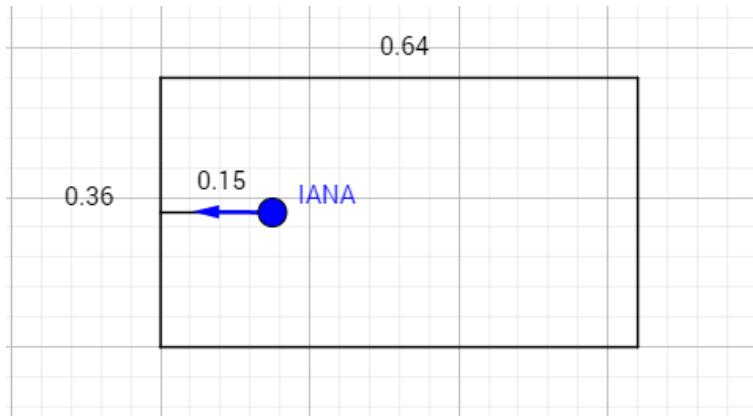


Abbildung 50: Tiefenscan-Sensoren Evaluation Testszenario 2 mit IANA

5.1.3.1 Tiefenscan

ZED	KinectOne	RPLidar
Abbildung 51: ZED Tiefenscan Die Wand ist zu nahe, es werden keine Tiefendaten registriert.	Abbildung 52: KinectOne Tiefenscan Die Wand ist zu nahe, es werden keine Tiefendaten registriert.	Abbildung 53: RPLidar Tiefenscan Bis auf den toten Winkel hinter dem Sensor werden für alle 360° Tiefendaten registriert.

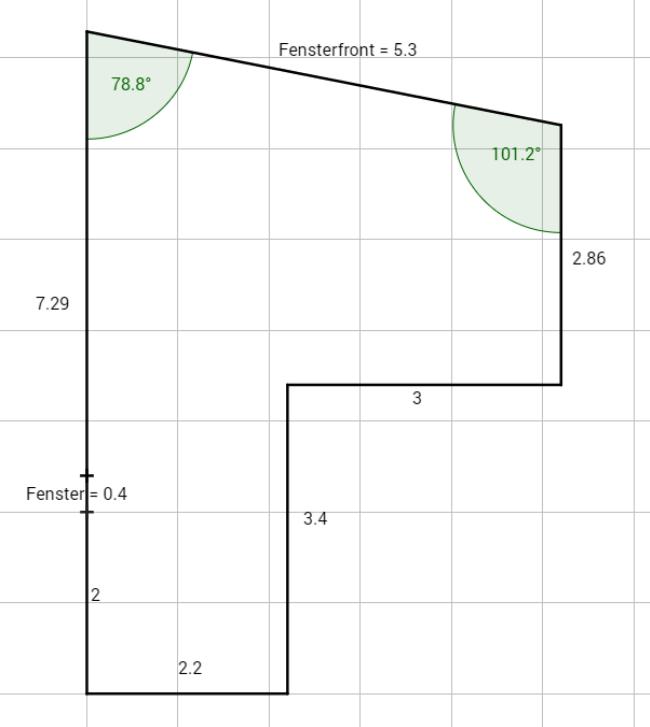
5.1.4 Fazit

Bei allen durchgeführten Testszenarien hat der RPLidar-Sensor die besten Ergebnisse geliefert. Die 360° Scanweite und die exakte Tiefenmesswerte auch bei sehr kleinen Distanzen sprechen klar für die Verwendung dieses Sensors. Während die Kinect bei der Genauigkeit der Messdaten etwa gleichauf ist mit dem RPLidar-Sensor und auch die Maximalreichweite in einem vergleichbaren Bereich liegt, verfügt sie nur über eine sehr eingeschränkte Scanweite und ist nicht fähig, im sehr nahen Bereich Tiefe zu erkennen. Die ZED weist bei allen Tests das deutlich schlechteste Resultat auf.

5.2 Testräume

Für die Tests und Validierungen für den Explore Driver, die Navigation und den SLAM Algorithmus haben wir einen gemeinsamen Testraum verwendet. Der Testraum hat verschiedene Winkel in den Ecken, eine Fensterfront und ist L-förmig. Fensterfronten machen die Aufgaben meist deutlich schwieriger, sind aber näher am späteren Einsatzgebiet des Roboters, dem IMVS. Der Testraum hilft

uns, die Tests nachvollziehbarer zu gestalten und schafft eine Basis für Vergleiche zwischen verschiedener Tests. Für den Testraum haben wir vier Settings bereitgestellt:

Settings	Fensterfront	Hindernisse
<u>Testraum</u> 	Ja	Nein
<u>Testraum ohne Fensterfront</u> Gleich wie Testraum nur ist die Fensterfront abgedeckt, was den Raum vereinfacht.	Nein	Nein

<p><u>Testraum mit Hindernissen</u></p>	Ja	Ja
<p><u>Testraum mit Hindernissen und ohne Fensterfront</u></p> <p>Gleich wie Testraum mit Hindernissen nur ist die Fensterfront abgedeckt, was den Raum vereinfacht.</p>	Nein	Ja

Welcher Test welches Setting verwendet, ist im jeweiligen Test angegeben.

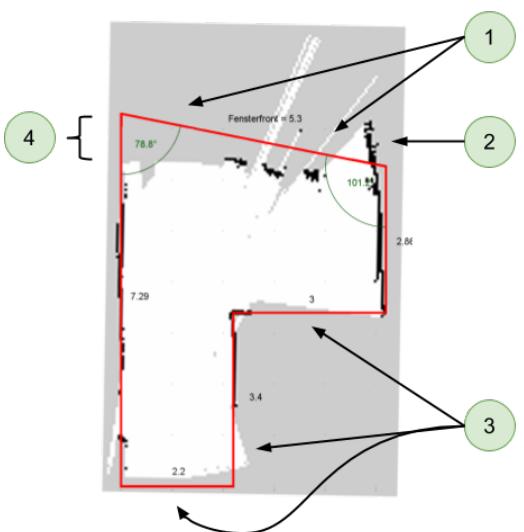
5.3 SLAM

5.3.1 Einleitung

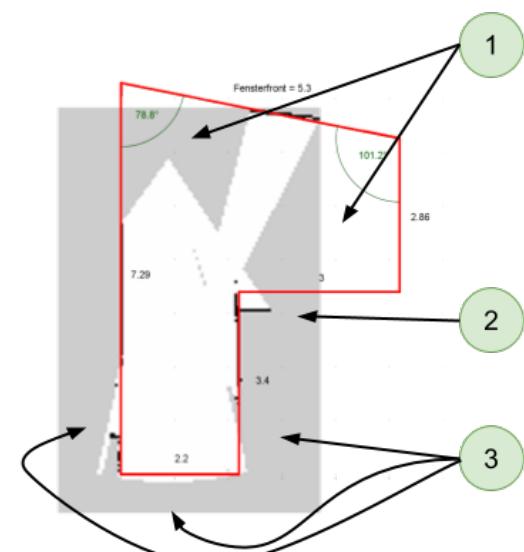
Als Tiefensor haben wir zuerst die KinectOne und später den RPLidar verwendet. Die Tests dieser beiden Sensoren sind hier beschrieben und werden anschliessend in einem Fazit miteinander verglichen.

5.3.2 Kinect

5.3.2.1 Offene Fensterfront

 <p>Abbildung 56: Kinect SLAM Map mit offener Fensterfront</p>	<p>Fünf Minuten hatte IANA Zeit den «Testraum» mit der KinectOne als Tiefensensor zu erfassen.</p> <p>Es zeigt sich folgendes:</p> <ol style="list-style-type: none"> 1. Die Fensterfront wird schlecht bis kaum erkannt. 2. Die Kinect erkennt manche Wände und das Fenster falsch, was auf eine falsch geschätzte Odometrie IANAs zurückzuführen ist. 3. Viele Wände wurden nicht als Hindernisse (schwarz) erkannt, sondern blieben unbekannt (grau). 4. Die Länge des Raums stimmt nicht mit der Wirklichkeit überein.
--	--

5.3.2.2 Geschlossene Fensterfront

 <p>Abbildung 57: Kinect SLAM Map mit geschlossener Fensterfront (1)</p>	<p>Fünf Minuten hatte IANA Zeit den «Testraum ohne Fensterfront» mit der KinectOne als Tiefensensor zu erfassen.</p> <p>Es zeigte sich folgendes:</p> <ol style="list-style-type: none"> 1. Der Raum wurde teilweise gar nicht erkannt, obwohl IANA den ganzen Raum durchfuhr. Der GMapping Algorithmus war sich allerdings durch die schlechten Scandaten zu unsicher. 2. Eine Kante wurde verschoben erkannt. Dies ist auf eine fälschlich geglaubte Odometrie zurückzuführen. 3. Viele Wände und das Fenster wurden nicht als Hindernisse erkannt, sondern blieben unbekannt.
--	---

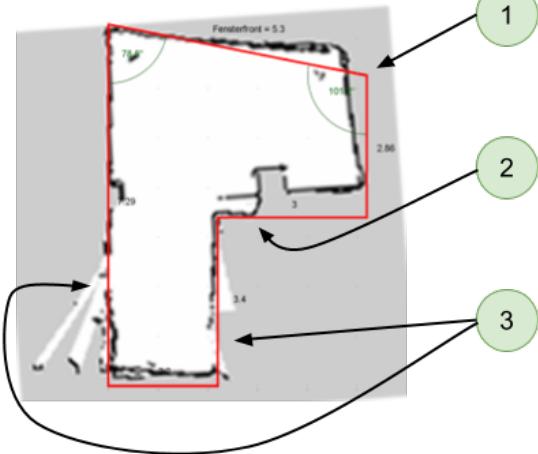


Abbildung 58: Kinect SLAM Map mit geschlossener Fensterfront (2), nach 11 Minuten

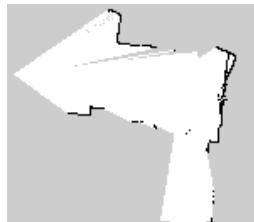


Abbildung 59: Kinect SLAM Map mit geschlossener Fensterfront (2), nach 5 Minuten

Dieses Setting ist der «Testraum» mit anders gelegten Hindernissen. Leider haben wir die Hindernisse in diesem Setting nicht ausgemessen. Rückwirkend können wir nicht sagen, wo die Hindernisse in Wirklichkeit genau liegen. Allerdings zeigt dieses Setting, dass die Kinect mit mehr Zeit und Hindernissen für die Orientierung (sozusagen «Landmarks») eine bessere Karte erstellen kann.

Elf Minuten hatte IANA Zeit dieses Setting mit der KinectOne als Tiefensensor zu erfassen.

Es zeigt sich folgendes:

- Die Raum ist deutlich besser als der Test mit weniger Zeit. In der Abbildung 59 ist die Karte nach fünf Minuten sichtbar.

 1. Der obere Teil der Karte ist gegenüber der Wirklichkeit verschoben, was auf eine falsch angenommene Ausrichtung IANAs zurückzuführen ist.
 2. Ein Wandstück wurde doppelt erfasst.
 3. Es gibt noch offene Stellen und das Fenster wurde nicht erkannt.

5.3.3 RPLidar

5.3.3.1 Offene Fensterfront



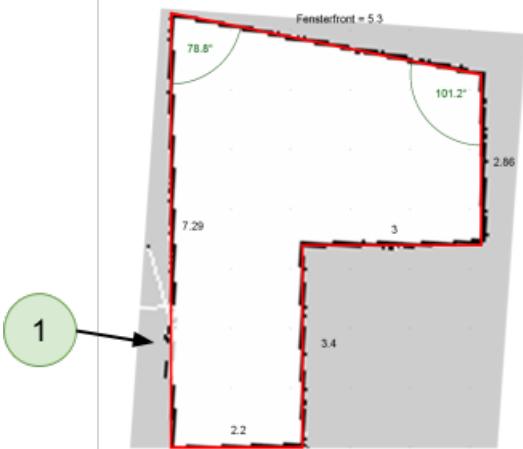
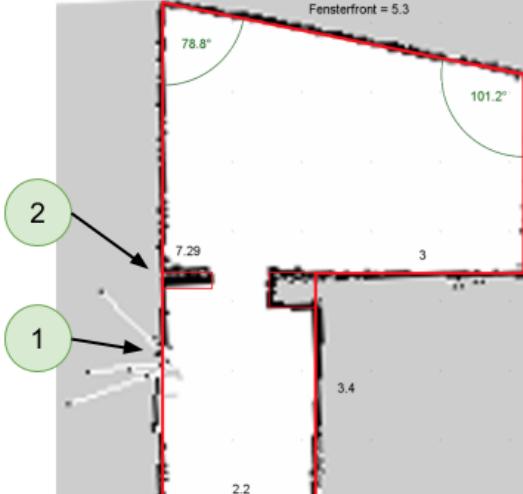
Abbildung 60: RPLidar SLAM Map mit offener Fensterfront

Fünf Minuten hatte IANA Zeit den «Testraum» mit dem RPLidar als Tiefensensor zu erfassen.

Es zeigte sich folgendes:

1. Die Fensterfront und das Fenster wurden nicht erkannt.
2. Die Länge des Raums wurde ein wenig zu kurz gemessen.
- Die Karte stimmt stark mit der Wirklichkeit überein und kann für eine erfolgreiche Navigation im Raum verwendet werden.

5.3.3.2 Geschlossene Fensterfront

 <p>Abbildung 61: RPLidar SLAM Map mit geschlossener Fensterfront</p>	<p>Fünf Minuten hatte IANA Zeit den «Testraum ohne Fensterfront» mit dem RPLidar als Tiefensensor zu erfassen.</p> <p>Es zeigte sich folgendes:</p> <ol style="list-style-type: none"> 1. Das Fenster wurde nicht erkannt. • Die Karte stimmt stark mit der Wirklichkeit überein und kann für eine erfolgreiche Navigation im Raum verwendet werden.
 <p>Abbildung 62: RPLidar SLAM Map im Testraum mit Hindernissen ohne Fensterfront</p>	<p>Fünf Minuten hatte IANA Zeit den «Testraum mit Hindernissen und ohne Fensterfront» mit dem RPLidar als Tiefensensor zu erfassen.</p> <p>Es zeigte sich folgendes:</p> <ol style="list-style-type: none"> 1. Das Fenster wurde nicht erkannt. 2. Ein Hindernis weist kleine Abweichungen von seiner eigentlichen Form auf. • Die Karte stimmt stark mit der Wirklichkeit überein und kann für eine erfolgreiche Navigation im Raum verwendet werden.

5.3.4 Entwicklung der Karte

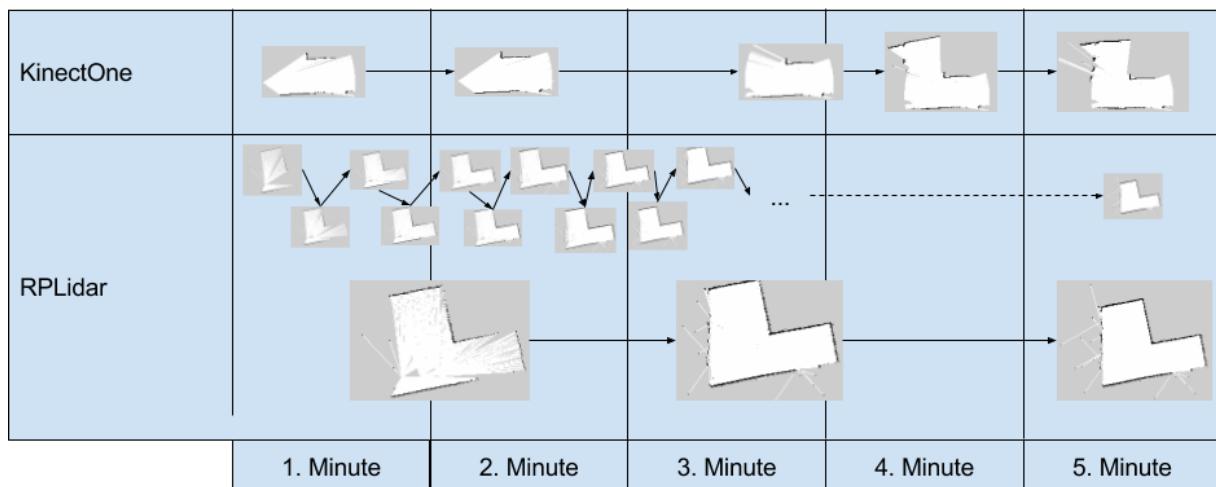


Abbildung 63: Entwicklung der SLAM Karte im Verlauf der Zeit (KinectOne vs. RPLidar)

Mit dem RPLidar als Tiefensensor veröffentlicht GMmapping viel öfters eine neue Karte. Dank den präziseren Laserscans und der präziseren Odometrie (dank der verbesserten Proposal Distribution), scannt GMmapping viel öfters Hindernisse auf den gleichen Zellen der Karte. So ist sich GMmapping schneller sicher, wo ein Hindernis und wo kein Hindernis ist und kann die Karte anpassen.

RPLidar hat beim «Testraum» (Siehe RPLidar/Offene Fensterfront) bereits nach zwei Minuten und 10 Sekunden praktisch dieselbe Karte erkennt, die es auch zum Schluss ausgibt. Beim «Testraum ohne Fensterfront» lässt sich das Selbe ebenfalls nach exakt zwei Minuten beobachten. Somit benötigt GMmapping mit RPLidar als Tiefensensor für unseren «Testraum» von den gegebenen fünf Minuten nur ungefähr zwei Minuten.

5.3.5 Vergleich vom RPLidar und der KinectOne

Die Tests zeigen eindeutig, dass die Karten des RPLidar Sensors besser sind, als die Karten der KinectOne, was nach der Tiefenscan-Evaluation (Siehe 5.1 Tiefenscan-Sensoren Evaluation) nicht weiter überrascht – bessere Sensordaten führen meist zu besseren Ergebnissen.

Das Problem an der KinectOne ist das eingeschränkte Blickfeld. GMmapping verkleinert die Unsicherheit der Odometrie des Roboters in der Proposal Distribution mit dem letzten Laserscan. Ist im Laserscan keine Wand sichtbar, oder nur ein kleiner Ausschnitt einer Wand, kann das Scan-Matching nur bedingt der Odometrie helfen.

5.3.6 Fazit

Unsere Tests zeigen

- GMmapping funktioniert deutlich besser mit dem RPLidar als mit der KinectOne.
- Mit dem RPLidar findet GMmapping eine wahrheitsgetreue Karte für kleine Räume
- GMmapping erkennt Fensterfronten nicht als Wände, da der Sensor diese nicht erkennt und GMmapping keine Annahmen darüber trifft.
- Wenig Fenster im Raum verwirren GMmapping nicht, und alles außer der Fensterfront wird weiterhin realitätsgetreu erkannt.

5.4 Navigation

5.4.1 Einleitung

IANAs Navigation ist dafür verantwortlich auf der vom SLAM Algorithmus erstellten Map einen Weg von A nach B zu finden und die nötigen Befehle an Kobuki zu senden, sodass IANA das Ziel erreicht. In diesem Test wollen wir überprüfen, ob der eingesetzte Navigations-Algorithmus auch bei Szenarien mit engen Passagen einen Weg zum Ziel findet und wie gross die Abweichung der erreichten Koordinaten von den eigentlichen Zielkoordinaten ist.

5.4.2 Testszenario

Als Raum dient der «Testraum mit Hindernissen und ohne Fensterfront». IANA hat bereits eine Karte des Raumes und steht im Punkt 0. Sie bekommt nun nacheinander 3 Ziele, die sie erreichen muss, wobei das letzte Ziel wieder dem Startpunkt entspricht. Erwartet wird, dass sie für alle Ziele einen Weg findet. Am Schluss werden für jeden der 3 Ziele die Differenz zwischen erreichter Position und tatsächlichem Ziel gemessen.

Die Navigationskomponente ist wie im Kapitel (Siehe 0

Navigation) beschrieben konfiguriert. Wichtig für diesen Test ist, dass die mit der verwendeten Konfiguration auch bei einer Abweichung von bis zu 21.2 cm von den Zielkoordinaten das Ziel als erreicht gilt. Erwartet wird, dass sich die Abweichung bei den Tests innerhalb dieses Bereichs befinden oder maximal leicht überschreiten.

Testraum: Testraum mit Hindernissen und ohne Fensterfront

Relevante Testwerte: Differenz zwischen Ziel und gemessenen Positionen

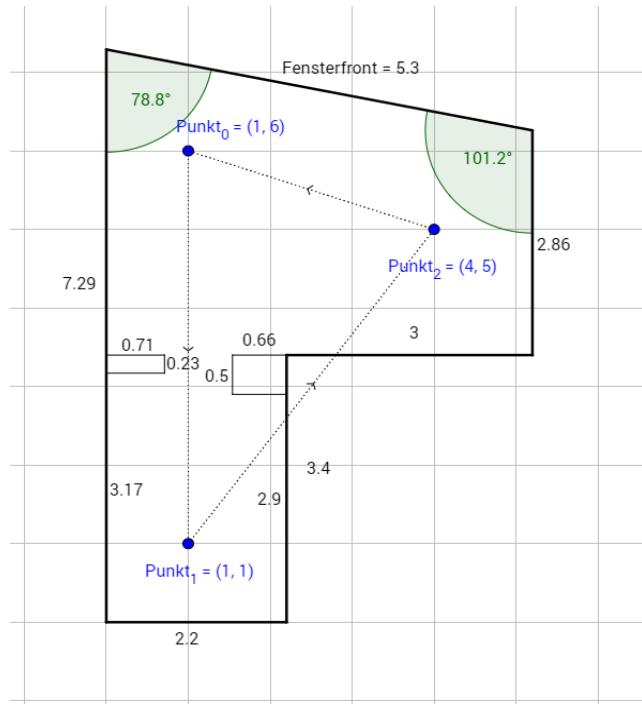


Abbildung 64: Navigation Testszenario

5.4.3 Ergebnis

Zielkoordinaten	Abweichung
(0, 0)	0 cm
(5, 0)	23 cm
(1, 3)	3 cm
(0, 0)	24 cm

5.4.4 Fazit

Für alle Ziele wurde ein Weg gefunden. Die Abweichungen der erreichten Positionen von den tatsächlichen Zielen liegen alle entweder innerhalb der erlaubten 21.2 cm oder leicht darüber. Die Überschreitungen der erlaubten Abweichung kann mit leichten Odometrie-Messfehlern erklärt werden. Da sie sich jedoch in einem sehr kleinen Rahmen bewegen, sehen wir keinen Handlungsbedarf.

5.5 Explorer

5.5.1 Einleitung

Mit den Explorer Tests wollen wir ermitteln wie sich die beiden in dieser Arbeit implementierten Explorer messen. Insbesondere wollen wir testen, ob der Rational Explorer tatsächlich effizienter vorgeht und schwierig zugängliche Bereiche zu erkunden versucht. In den Tests verwenden wir den Testraum mit Hindernissen und ohne Fensterfront. Dieser Raum verfügt über eine verengte Stelle, die den Raum in zwei Bereiche aufteilt. Wir erwarten, dass der Random Explorer mit dem passieren dieser Stelle Mühe bekundet.

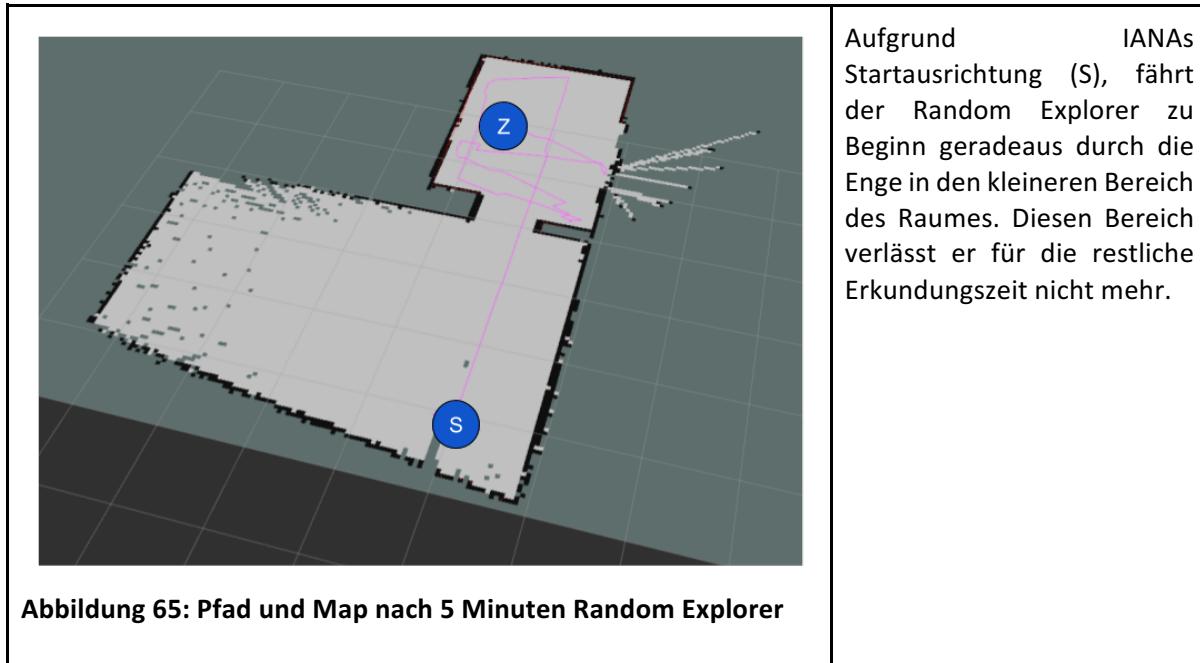
Testdauer: 5 Minuten

Testraum: Testraum mit Hindernissen und ohne Fensterfront

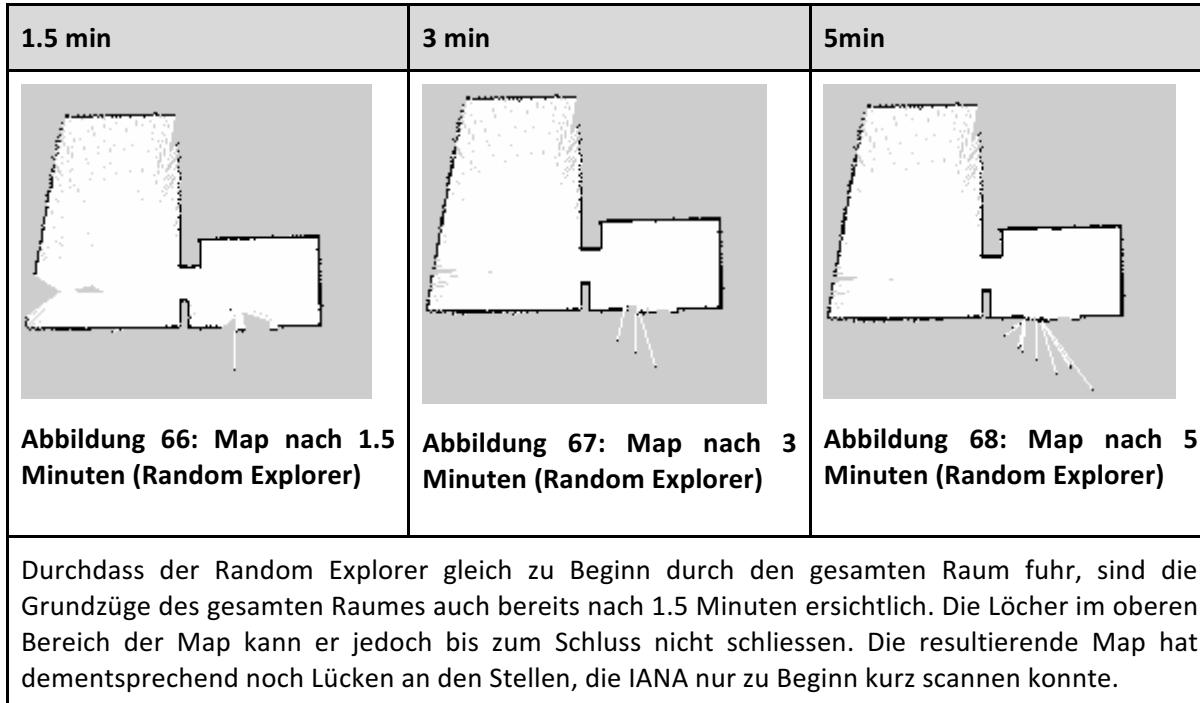
Relevante Testwerte: Pfad, Entwicklung der Map, resultierende Map

5.5.2 Random Explorer

5.5.2.1 Pfad



5.5.2.2 Map Entwicklung und resultierende Map



5.5.3 Rational Explorer

5.5.3.1 Pfad

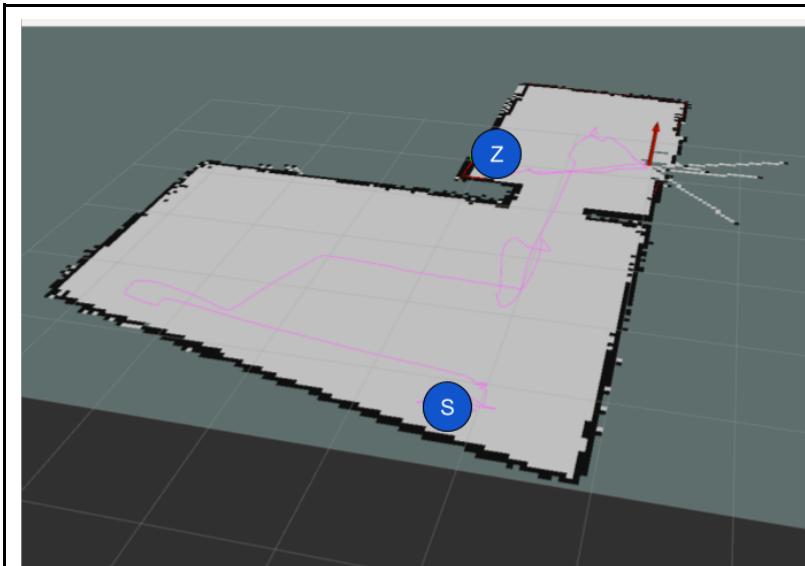


Abbildung 69: Pfad und Map nach 5 Minuten Rational Explorer

Der Rational Explorer erkundet zunächst den grossen Bereich in dem sich auch die Startposition (S) befindet. Anschliessend navigiert er IANA erfolgreich durch die Raumenge und erkundet auch diesen Bereich.

5.5.3.2 Map Entwicklung und resultierende Map

1.5 min	3 min	5min

Abbildung 70: Map nach 1.5 Minuten (Rational Explorer)

Abbildung 71: Map nach 3 Minuten (Rational Explorer)

Abbildung 72: Map nach 5 Minuten (Rational Explorer)

Die Map entwickelt sich regelmässig, nach 3 Minuten ist bereits die gesamte Map erkundet. In der resultierenden Map ist die Ausrichtung des rechten Teilbereiches noch etwas korrigiert. Ausser beim kleinen Fenster sind keine Fronten mehr vorhanden.

5.5.4 Fazit

Der Test zeigt, dass der Rational Explorer zielstrebig vorgeht beim Erkunden der Umgebung, im Gegensatz zum Random Explorer, der fast über die gesamten 5 Minuten in einem Teilbereich festgehangen hat. Das Resultat nach 5 Minuten Erkundung durch Rational Explorer ist dementsprechend vollständiger, sogar bereits nach 3 Minuten war die Karte beinahe vollständig erkundet.

Um genauere Aussagen über die Beschleunigung der Erkundung treffen zu können, müssten jedoch auch Tests in grösseren Räumlichkeiten durchgeführt werden.

5.6 Person Detection

5.6.1 Gesichtsdetektion

Wir wollen unseren neuen HaarFaceDetector und unsere veränderten DLibFaceDetector vergleichen. Dazu wiederholen wir die FaceDetectionValidation [81] vom IP5 Projekt mit den neuen FaceDetectoren. Zusätzlich messen wir noch die Performance.

	DLibFaceDetector		HaarFaceDetector	
	IP5	IP6	haarcascade frontalface model	lbpcascade frontalface model
Bilder	2842	2845	2845	2845
Gesichter	5168	5171	5171	5171
TP	3972	3697	3272	2381
FP	19	5	69	4
FN	1196	1474	1899	2790
Zeit in Sekunden (Durchschnitt von 10 Runs)	Kein Wert	~ 94.98	~ 48.97	~ 24.49
Zeit in Sekunden auf IANA (Durchschnitt von 10 Runs)	Kein Wert	~ 1008.67	~ 158.54	~ 53.46

5.6.2 Fazit

In diesem Test ist der qualitativ beste Gesichtsdetektor der DLibFaceDetector vom IP5 Projekt. Der DLibFaceDetector vom IP6 Projekt ist ihm ähnlich, nur skaliert er die Bilder im Pre-processing nicht mehr hoch. Ein Nachteil in diesem Test, im ROS System haben wir aber HD-Bilder, die wir vor der Gesichtsdetektion herunterskalieren.

Ein Haar Cascade basierter Ansatz ist einem HoG basierten Ansatz bei einer Gesichtsdetektion unterlegen, weshalb wir uns im IP5 Projekt für den HoG Ansatz entschieden (Siehe «Gesichtsdetektion» im separates Dokument «IANA – IMVS Autonomous Neural Assistant»). Daher überrascht es nicht, dass ein HaarFaceDetector schlechter abschneidet als ein DlibFaceDetector (HoG basiert).

In der Geschwindigkeit schneidet die beiden HaarFaceDetector allerdings besser ab, was das Ziel der Entwicklung dieser Detektoren war.

Das «lbpcascade frontalface model» nutzt nur Integer Arithmetik und ist daher deutlich schneller als das «haarcascade frontalface model». Allerdings kommt dieser Vorteil zu einem hohen Preis, im Schnitt wurden 17.23% weniger korrekte Gesichter erkannt.

Der HaarFaceDetector mit «haarcascade frontalface model» ist im Schnitt doppelt so schnell wie der DLibFaceDetector vom IP6 und erkennt 8.22% korrekte Gesichter weniger. Allerdings muss erwähnt

werden, dass er dafür 69 falsche Gesichter erkannte, was 13.8 Mal mehr Fehler sind, als im DLibFaceDetector IP6. Auf IANA ist der Performance Unterschied noch deutlich grösser, da der HaarFaceDetector die Grafikkarte ausnutzen kann.

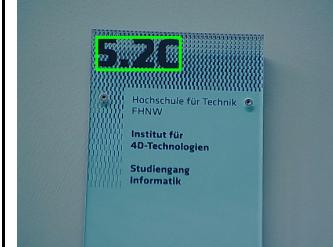
Schlussendlich haben wir uns für den HaarFaceDetector mit «haarcascade frontalface model» entschieden, da er eine Balance zwischen Geschwindigkeit und Qualität liefert, die für IANA mit ihrem Videostream für die Gesichtsdetektion geeignet ist.

5.7 Text Recognition

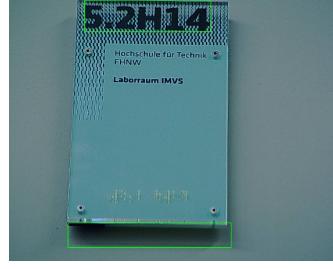
5.7.1 Testdaten

Unsere Texterkennung ist zugeschnitten auf die FHNW Türschilder, daher gibt es keine sinnvollen öffentlichen Testdaten dafür. Um die Texterkennung dennoch auszuwerten, haben wir 8 Bilder aus einer vordefinierten Distanz gesammelt (2 Meter Entfernung). Dieser kleine Datensatz deckt mögliche Bilder der FHNW Türschilder ab, die IANA während der Fahrt begegnen könnte.

5.7.2 Text Detection

Name	Bild	Nummer-Erkenner	Namens-Erkenner
01.jpg			
	Abbildung 73: Originalbild von 01.jpg	Abbildung 74: Nummer-Erkenner in 01.jpg	Abbildung 75: Namens-Erkenner in 01.jpg
02.jpg			
	Abbildung 76: Originalbild von 02.jpg	Abbildung 77: Nummer-Erkenner in 02.jpg	Abbildung 78: Namens-Erkenner in 02.jpg

03.jpg			
	Abbildung 79: Originalbild von 03.jpg	Abbildung 80: Nummer-Erkenner in 03.jpg	Abbildung 81: Namens-Erkenner in 03.jpg
04.jpg			
	Abbildung 82: Originalbild von 04.jpg	Abbildung 83: Nummer-Erkenner in 04.jpg	Abbildung 84: Namens-Erkenner in 04.jpg
05.jpg			
	Abbildung 85: Originalbild von 05.jpg	Abbildung 86: Nummer-Erkenner in 05.jpg	Abbildung 87: Namens-Erkenner in 05.jpg
06.jpg			
	Abbildung 88: Originalbild von 06.jpg	Abbildung 89: Nummer-Erkenner in 06.jpg	Abbildung 90: Namens-Erkenner in 06.jpg

07.jpg			
	Abbildung 91: Originalbild von 07.jpg	Abbildung 92: Nummer-Erkenner in 07.jpg	Abbildung 93: Namens-Erkenner in 07.jpg

08.jpg			
	Abbildung 94: Originalbild von 08.jpg	Abbildung 95: Nummer-Erkenner in 08.jpg	Abbildung 96: Namens-Erkenner in 08.jpg

5.7.3 Text Recognition

Bild	Erwartetes Resultat	Tatsächliches Resultat
01.jpg		
	«5.2C»	«5.2C»
Abbildung 97 Nummer-Erkenner in 01.jpg		
	<p>«Hochschule für Technik FHNW»</p> <p>«Institut für 4D-Technologien»</p> <p>«Studiengang Informatik»</p>	<p>«Hochschule für Technik FHNW»</p> <p>«Institut für 4D-Technologien»</p> <p>«Studiengang Informatik»</p>
Abbildung 98: Namens-Erkenner in 01.jpg		

02.jpg



«5.2H12»

Detektion fehlgeschlagen wegen dem Schatten des Schildes:



Abbildung 99: Nummer-Erkenner in 01.jpg



«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DS/SGI»

«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DS/SGI»

Abbildung 100: Namens-Erkenner in 02.jpg

03.jpg



«5.2H12»

Detektion fehlgeschlagen wegen dem Schatten des Schildes:



Abbildung 101: Nummer-Erkenner in 03.jpg



«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DS/SGI»

«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DS/SGI»

Abbildung 102: Namens-Erkenner in 03.jpg

04.jpg



«5.2H12»

Detektion fehlgeschlagen wegen dem Schatten des Schildes:



Abbildung 103: Nummer-Erkenner in 04.jpg



«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DS/SGI»

«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DS/SGI»

Abbildung 104: Namens-Erkenner in 04.jpg

05.jpg



«5.2H12»

Detektion fehlgeschlagen wegen dem Schatten des Schildes:



Abbildung 105: Nummer-Erkenner in 05.jpg



«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DS/SGI»

«Hochschule für Technik FHNW»
«Media-Labor IMVS/I4DSISGI»

Abbildung 106: Namens-Erkenner in 05.jpg

06.jpg



«5.2H12»

Detektion fehlgeschlagen wegen dem Schatten des Schildes:



Abbildung 107: Nummer-Erkenner in 06.jpg



«Hochschule für Technik
FHNW»
«Media-Labor
IMVS/I4DS/SGI»

«Hochschule für Technik
FHNW»
«Media-Labor IMVS/I4DS~~S~~GI»

Abbildung 108: Namens-Erkenner in 06.jpg

07.jpg



«5.2H14»

»5.1H~~1~~4«

Abbildung 109: Nummer-Erkenner in 07.jpg



«Hochschule für
Technik FHNW»
«Laborraum IMVS»

«Hochschule~~F~~ für Technik
FHNW»
«Laborraum IMVS»

Abbildung 110: Namens-Erkenner in 07.jpg

08.jpg



Abbildung 111: Nummer-Erkenner in 08.jpg

«5.2H14»

«5.2H14»



Abbildung 112: Namens-Erkenner in 08.jpg

«Hochschule für
Technik FHNW»
«Laborraum IMVS»

«Hochschule für Technik
FHNW»
«Laborraum IMVS»

5.7.4 Fazit

Die Textdetektion und die Texterkennung funktionieren mit den starken Annahmen über die FHNW Türschilder und deren Grösse. Bei der Detektion können Schattenwürfe zu False Positives führen. Problematisch wird es bei der Nummer-Erkennung, wenn der Schatten in den Text verstrichen wird, und deshalb der Text nicht erkannt wird.

Die Texterkennung funktioniert teilweise und erkennt immer wieder einzelne Buchstaben falsch.

Um verwendet zu werden, müsste die Texterkennung noch verbessert werden. Man könnte die erkannten Texte mit möglichen Wörtern vergleichen, um so Fehler zu erkennen.

5.8 Ressourcen Verbrauch

5.8.1 Einleitung

IANA soll im Stande sein, auch grössere Bürolandschaften zu erkunden und kartographieren. In diesem Test wollen wir ermitteln, wie sehr der Ressourcenverbrauch (Laufzeit, Speicher) bei wachsender Kartengrösse zunimmt.

5.8.2 Testszenario

IANA soll einen sehr kleinen und einen sehr grossen Raum erkunden. Bei beiden messen wir Mapgrösse und Ressourcenauslastung, nachdem die Kartographierung grösstenteils abgeschlossen ist. Der grosse Raum liegt etwa im Grössenbereich des IMVS.

Kleiner Raum: ca. 15m²

Grosser Raum: ca. 450m²

5.8.3 Ergebnis

Mapgrösse	Raum-grösse	Laufzeit	Arbeitsspeicher-auslastung	Prozessorauslastung (auf 4 Kerne verteilt)
Leerlauf			~ 1.1 GB (27.1%)	~ 14.3 %
3'642 Pixel	ca. 15 m ²	30 s	~ 2.3 GB (60,3%)	~ 45.5 %
68'430 Pixel	ca. 450 m ²	30 min	~ 2.9 GB (74.3%)	~ 69.7 %

5.8.4 Fazit

Die Ressourcenauslastung bei zunehmender Kartengrösse nimmt sowohl beim Arbeitsspeicher als auch bei den Prozessoren merklich zu. Die Tests haben jedoch gezeigt, dass selbst beim Kartographieren eines sehr grossen Raumes, wie beispielsweise den Räumlichkeiten des IMVS, noch Ressourcen verfügbar bleiben.

6 Ergebnisse und Fazit

6.1 Source Code

Der gesamter Source Code von IANA mit allen ROS Packages, Prototypen und weiteren Komponenten befindet sich auf Git Labs unter dem geschützten Link:

https://gitlab.fhnw.ch/christian.scheller/iana_ip6

6.2 Roboter

IANAs Roboterhardware bestehend aus Kobuki, Jetson TX1-Board, Arduino UNO-Board, Pan&Tilt-Vorrichtung, RPLidar-Laserscanner, ZED-Kamera und D-Link Highspeed Kamera sind zusammengebaut, angeschlossen und mit der nötigen Stromversorgung versehen. Treiber und Ansteuerungssoftware sind programmiert, installiert und konfiguriert. Alle Komponenten können mittels von uns programmierte oder installierte ROS-Nodes individuell gesteuert werden.

6.3 Roboter durch Bürolandschaft bewegen

IANA erkundet autonom mit einem Rational Explorer Driver oder einer Random Explorer ihre Umgebung. Dabei erstellt sie mittels GMapping SLAM-Algorithmus eine zweidimensionale Grid Map und bestimmt, welche Kacheln frei und welche besetzt sind. Wir haben gezeigt, dass der verwendete GMapping Algorithmus mit dem RPLidar Sensor realitätsnahe Karten für kleine Räume ohne Fensterfronten findet. Hat der Raum eine Fensterfront, entdeckt der Algorithmus alles außer dieser Fensterfront realitätsgerecht.

Die Erkennung der FHNW Türschilder stellte sich im Projekt als grösste Aufgabe heraus als zu Beginn angenommen. Wir brauchten eine extra Kamera mit optischem Zoom, um die Texte ausreichend zu erkennen. Dies hat zur Folge, dass der Roboter sich vor dem Türschild platzieren müsste, um den Text zuverlässig zu erkennen. Dieses Verhalten benötigte einen Türschilddetektor, damit der Roboter weiß, wo und wie er sich ausrichten muss. Weder der Türschilddetektor, noch das Ausrichten wurden umgesetzt. Die Texterkennung mit Textdetektion auf dem Bild der Kamera wurde umgesetzt und spärlich getestet. Die Einbindung der erkannten Texte in die Karte des SLAM Algorithmus fehlt, aus Zeitgründen konnten wir sie nicht mehr in der von uns gewünschten Qualität umsetzen.

Über die Benutzerschnittstelle kann man dem Roboter GoTo-Befehle schicken und ihn in der erkannten Karte herumfahren lassen. Wir haben gezeigt, dass die GoTo-Befehle bis auf wenige Zentimeter genau sind.

6.4 Interaktion mit Personen

Die Gesichtserkennung aus dem vorangegangenen Projekt wurde übernommen und zu ROS umgeschrieben. Um die Performance der gesamten Gesichtserkennung zu verbessern, wurde das Bottleneck Gesichtsdetektion durch eine neue Implementierung ersetzt. Die neue Gesichtsdetektion wurde mit der bestehenden auf Robustheit und Performance verglichen. Sie ist schneller, allerdings zu kosten der Robustheit, die Anzahl False Positives stieg, die Anzahl True Positives sank.

Verbessert wurde die Robustheit der Gesichtserkennung, indem das Bild nicht zu Beginn für die ganze Pipeline herunterskaliert wird, sondern nur für die Gesichtsdetektion. Zudem hilft der Face Tracker ein Gesicht länger zu erkennen.

Weiter wurde das in IP5 bereits vorgesehene jedoch noch nicht Umgesetzte Aktualisieren der gespeicherten Gesichts Feature-Vektoren bekannter Personen implementiert. Somit aktualisiert IANA

ständig ihr Wissen über wiedererkannte Personen, was ihr helfen soll, auch Veränderungen von Gesichtern (Alterung, Gesichtsbehaarung, etc.) zu registrieren und zu berücksichtigen.

An einer Benutzerschnittstelle kann eine erkannte Person sich IANA vorstellen und neu gibt es ein Preview-Bild, was zeigt, welche Person erkannt wurde. Zudem wird die Frage nach dem Namen nun abgebrochen, wenn die Person das Blickfeld IANAs verlässt. Die Ausnutzung des gesamten Originalbilds für die Gesichtserkennung und der Einfluss vom Face Tracker wurden aus Zeitgründen nicht getestet und nicht validiert.

6.5 Tests

In unserem «Testraum» mit den vier Settings zeigten wir, dass IANA keine Probleme mit kleinen Räumen ohne Fensterfronten hat. Fensterfronten können selbst nicht erkannt werden, stören aber die restliche Erkundung nur gering. Im Anhang (Siehe 8.2 Grenzen strapazieren) haben wir zudem einen grösseren Test in den Gängen der FHNW dokumentiert. Dieser Test zeigt, dass IANA grössere Räume nur teilweise erkennen kann und auch der Rational Explorer eine Erweiterung benötigt (Siehe 7.3 Frontier-Auswahl verbessern).

Wir zeigten nicht vollumfänglich, «dass IANA über längere Zeit in der Lage ist, bereits angetroffene Personen wiederzuerkennen und unbekannte Personen zu erfassen» (Siehe Aufgabenstellung). Aus Zeitgründen haben wir es nicht mehr geschafft, die dafür nötigen systematischen Tests zur Erkennung bereits bekannter Personen über längere Zeit durchzuführen. Wie verweisen hier auf das Kapitel «Weiteren Schritte», wo wir unseren Ansatz für diese Tests beschreiben (Siehe 7.1 Validierung der Erkennung bereits bekannter Personen über längere Zeit). Ergebnisse aus dem IP5 zeigen jedoch, wie sich die Fähigkeit bereits angetroffene Personen wiederzuerkennen und unbekannte Personen zu erfassen bei steigender Anzahl Personen entwickelt (Siehe separates Dokument «IANA – IMVS Autonomous Neural Assistant», Kapitel Ergebnisse). Damit zeigen wir auch bereits, wie sich die Erkennung unbekannter Personen über die Zeit entwickelt, denn dafür sind lediglich die Anzahl Personen im Korpus der bekannten Personen ausschlaggebend, nicht aber deren Alterung über die Zeit.

7 Weitere Schritte

IANA befindet sich in einem vielversprechenden Zustand, doch gibt es noch viele Möglichkeiten und Ideen IANA zu verbessern. Hier wollen wir einige unserer Vorschläge auflisten, wie man IANA in Zukunft erweitern könnte:

7.1 Validierung der Erkennung bereits bekannter Personen über längere Zeit

Ein von uns nicht erreichtes Ziel ist die Validierung der Gesichtserkennung über längere Zeit. Dazu bräuchte man ein Datensatz mit Gesichtern und Alter gruppiert nach der Person. Ein möglicher Datensatz wäre das «IMDB-WIKI» dataset [82], welches aus gelabelten Bildern inklusive Altersangaben besteht, wobei jede Person mit mehreren Gesichtern vertreten ist. Dieser Datensatz wurde für Arbeiten im Bereich automatischer Schätzung des Alters von Gesichtern [83] [84] entwickelt und müsste für unsere Zwecke noch aufbereitet werden. Nachteil dieses Datensatzes ist, dass die Gesichter nur auf Jahre genau gelabelt sind, für unsere Tests wären jedoch Wochen oder Monatsangaben besser geeignet.

Mit einem geeigneten Datensatz würden wir die Streuung der OpenFace Vektoren einer Person im steigenden Alter betrachten und mit der Streuung der OpenFace Vektoren einer Person mit konstantem Alter vergleichen. So können wir feststellen, ob und wie stark die Abweichung in OpenFace wegen des Alters stattfindet. Anschliessend können wir einen systematischen Test designen, der unserem System chronologisch Bilder von Personen speist. Diesen Test können wir mit und ohne unsere Aktualisierungsfunktion (Siehe 3.9.2.3 Aktualisierung der Gesichts Feature-Vektoren) laufen lassen und die Ergebnisse miteinander vergleichen. So kann gezeigt werden, ob die Abweichung im Gesicht über die Zeit von der Aktualisierungsfunktion erfasst werden kann.

7.2 Texterkennung in Karte einbauen

Ein von uns nicht erreichtes Ziel ist den Text in die Karte einzubauen. Zur Zeit der Erkundung hat IANA noch keine sichere Karte, was zur Folge hat, dass die Karte sich verändern kann. Wir raten daher davon ab, sich die Positionen der Türschilder relative zur aktuellen Position des Roboters zu merken. Ein vielversprechender Ansatz wäre es, die erkannten Texte in den SLAM Algorithmus aufzunehmen. GMapping verwaltet bei uns intern 80 Partikel. Man müsste sich für jedes Partikel die Texte relative zur Position des Roboters in diesem Partikel merken. So können sich die Texte mit der Karte zusammen verändern, wird ein anderes Partikel das Vielversprechendste.

7.3 Frontier-Auswahl verbessern

Zurzeit haben wir eine rein zufällige Frontier-Auswahl. Problematisch dabei ist, dass der Roboter in einem langen Gang ständig hin und her fahren würde, zu den beiden jeweils offenen Enden. Dies könnte man beheben, indem man die gefundenen Frontiers nach ihrer Nähe zum Roboters gewichtet – Frontiers die dem Roboter nahe sind, haben eine grössere Wahrscheinlichkeit selektiert zu werden. Dadurch kann der Roboter nicht in das ewige Hin und Her von der nächsten Frontier-Auswahl (Siehe 3.8.3.3 Auswahl des Zielpunktes) geraten, sucht die Frontiers aber klüger aus, als die zufällige Frontier-Wahl.

7.4 Erkennung von Türschilder und Ausrichtung des Roboters

Die D-Link Highspeed Kamera mit Zoom-Objektiv muss in einer bestimmten Distanz und in einem bestimmten Winkel zum Türschild stehen, sodass dieses entzifferbar im Bild ist. Dies ist ein Problem, denn es ist sehr unwahrscheinlich, dass IANA während dem Erkunden zufällig im korrekten Abstand und Winkel kurz anhält und dabei den Text im Türschild erkennt. Um zuverlässig die FHNW Türschilder

zu erkennen, benötigt IANA zwei zusätzliche Komponenten. Die eine Komponente richtet sich während dem Explore Task vor dem Schild aus, wenn eines erkannt wird. Die zweite Komponente erkennt die FHNW Türschilder im ZED-Bild und veröffentlicht deren Position an die erste Komponente.

Das Ausrichten des Roboters ist dank des Navigation Stack kein Problem.

Die Berechnung der Position vom Türschild anhand der ZED-Bilder kann man mit einem Trick umgehen: In der Karte hat man wahrscheinlich die Wand bereits erkannt, auf welcher sich das Türschild befindet, also muss man nur noch herausfinden, wo an dieser Wand das Türschild befestigt ist, was deutlich einfacher ist. Die Position, die IANA einnehmen muss, kann dann relativ zur entsprechenden Stelle an Wand auf der 2D Karte berechnet werden.

Die Erkennung von Türschildern in den ZED-Bildern ist die schwierigste der drei Aufgaben. Man müsste einen Detektor für dieses Objekt entwickeln, aber wie wir das im Detail machen würden, haben wir nicht entworfen.

8 Anhang

8.1 Definition Rationalität

Im Buch «Künstliche Intelligenz – Ein moderner Ansatz» hängt Rationalität von vier Punkten ab:

«

- der Leistungsbewertung, die das Erfolgskriterium definiert
- dem Vorwissen des Agenten über die Umgebung
- den Aktionen, die der Agent ausführen kann
- der bisherigen Wahrnehmungsfolge des Agenten

» [85]

Rationalität ist anders wie Perfektion: «Die Rationalität maximiert die erwartete Leistung, während die Perfektion die tatsächliche Leistung maximiert.» [86]

8.2 Grenzen strapazieren

Nach unseren Tests im nachvollziehbaren «Testraum» und seinen Settings, haben wir IANA auf den Gängen der Fachhochschule ausprobiert. IANA ist für 35 Minuten mit dem Erweiterten Rational Explorer gefahren, und versuchte die Karte zu erkennen. Der Test ist nicht systematisch durchgeführt, gibt allerdings eine Impression darüber, wie IANA sich in grossen Räumen schlagen kann.

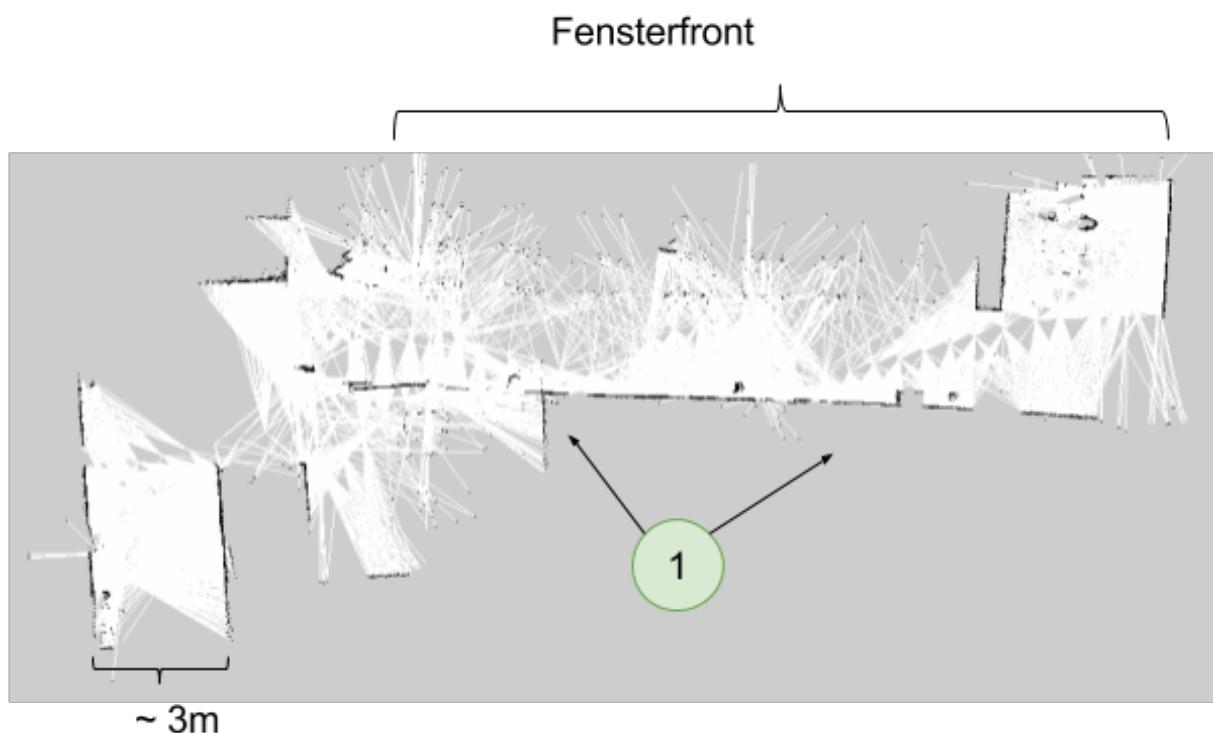


Abbildung 113: IANA in Gang der FHNW

Das Problem im Raum scheint vor allem die lange Fensterfront zu sein, dort gibt es noch viele Unbekannte Bereiche (grau). Die Odometrie hat sich zudem leicht verschoben (1) was zu einer Verschiebung in der Karte führt. Die Grösse des Raumes macht IANA unsicherer, wahrscheinlich, weil unserer Laserscanner mit 5.5-6 Meter Reichweite an Präzision verliert, ein besserer Laserscanner könnte helfen.

Während des Tests zeigte sich, dass der Rational Explorer noch überarbeitet werden muss. Oft fand er unerreichbare Frontiers und Frontiers die sehr weit entfernt waren, was zu einem Hin und Her geführt hätte.

Der Test deutet darauf hin, dass IANA in einem grossen Raum mit grosser Fensterfront nicht wünschenswert funktioniert.

9 Literaturverzeichnis

- [1] «ROS-Wiki Navigation,» [Online]. Available: <http://wiki.ros.org/navigation>. [Zugriff am 18 08 2017].
- [2] U. F. G. G. Cyrill Stachniss, «OpenSLAM.org,» [Online]. Available: <http://openslam.org/>. [Zugriff am 18 08 2017].
- [3] «Kobuki Turtlebot 2,» [Online]. Available: <http://kobuki.yujinrobot.com/about2/>. [Zugriff am 18 08 2017].
- [4] «Robot Locomotion,» [Online]. Available: http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html. [Zugriff am 18 08 2017].
- [5] «Nvidia Jetson TX1,» [Online]. Available: <http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html#section3>. [Zugriff am 18 08 2017].
- [6] «ZED,» [Online]. Available: <https://www.stereolabs.com/zed/specs/>. [Zugriff am 18 08 2017].
- [7] «ROS Wiki: Depthimage to Laserscan,» [Online]. Available: http://wiki.ros.org/depthimage_to_laserscan. [Zugriff am 18 08 2017].
- [8] «RPLidar a1,» Slamtec, [Online]. Available: <http://www.slamtec.com/en/lidar/a1>. [Zugriff am 18 08 2017].
- [9] «Lynxmotion Large Pan/Tilt,» [Online]. Available: <http://www.lynxmotion.com/p-892-large-pantilt-black-anodized-no-servos.aspx>. [Zugriff am 18 08 2017].
- [10] «Computar Varifocal Lens TG10Z0513AFCS,» Computar, [Online]. Available: <https://computar.com/product/547/TG10Z0513AFCS>. [Zugriff am 18 08 2017].
- [11] «TurtleBot 2,» TurtleBot, [Online]. Available: <http://www.turtlebot.com/turtlebot2/>. [Zugriff am 18 08 2017].
- [12] «Jetson Jetpack,» NVIDIA, [Online]. Available: <https://developer.nvidia.com/embedded/jetpack>. [Zugriff am 18 08 2017].
- [13] «Linux For Tegra R28.1,» NVIDIA, [Online]. Available: <https://developer.nvidia.com/embedded/linux-tegra>. [Zugriff am 18 08 2017].
- [14] «ROS Wiki: Kinetic,» [Online]. Available: <http://wiki.ros.org/kinetic>. [Zugriff am 18 08 2017].
- [15] «NVIDIA TensorRT,» NVIDIA , [Online]. Available: <https://developer.nvidia.com/tensorrt>. [Zugriff am 18 08 2017].
- [16] «NVIDIA cuDNN,» NVIDIA , [Online]. Available: <https://developer.nvidia.com/cudnn>. [Zugriff am 18 08 2017].
- [17] «NVIDIA VisionWorks,» NVIDIA, [Online]. Available: <https://developer.nvidia.com/embedded/visionworks>. [Zugriff am 18 08 2017].

- [18] «OpenVX,» Khronos, [Online]. Available: <https://www.khronos.org/openvx>. [Zugriff am 18 08 2017].
- [19] «Wikipedia: Robot Operating System,» [Online]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System. [Zugriff am 18 08 2017].
- [20] «ROS Wiki: TCPROS,» [Online]. Available: <http://wiki.ros.org/ROS/TCPROS>. [Zugriff am 18 08 2017].
- [21] «ROS Wiki: Catkin,» [Online]. Available: <http://wiki.ros.org/catkin>. [Zugriff am 18 08 2017].
- [22] «ROS Wiki: Rviz,» [Online]. Available: <http://wiki.ros.org/rviz>. [Zugriff am 18 08 2017].
- [23] «ROS Wiki: Navigation,» [Online]. Available: <http://wiki.ros.org/navigation>. [Zugriff am 18 08 2017].
- [24] «ROS Wiki: TF,» [Online]. Available: <http://wiki.ros.org/tf>. [Zugriff am 18 08 2017].
- [25] «ROS Community Metrics Report,» 07 2016. [Online]. Available: <http://download.ros.org/downloads/metrics/metrics-report-2016-07.pdf>. [Zugriff am 18 08 2017].
- [26] . B. Gerkey, «ROS, the Robot Operating System, Is Growing Faster Than Ever, Celebrates 8 Years,» 09 12 2015. [Online]. Available: <http://spectrum.ieee.org/automaton/robotics/robotics-software/ros-robot-operating-system-celebrates-8-years>. [Zugriff am 18 08 2017].
- [27] «ROS: Testimonials,» [Online]. Available: <http://www.ros.org/testimonials/>. [Zugriff am 18 08 2017].
- [28] «ROS Wiki: Nodes,» [Online]. Available: <http://wiki.ros.org/Nodes>. [Zugriff am 18 08 2017].
- [29] «ROS Wiki: Roscore,» [Online]. Available: <http://wiki.ros.org/roscore>. [Zugriff am 18 08 2017].
- [30] «ROS Wiki: Master API,» [Online]. Available: http://wiki.ros.org/ROS/Master_API. [Zugriff am 18 08 2017].
- [31] «ROS Wiki: Parameterserver,» [Online]. Available: <http://wiki.ros.org/Parameter%20Server>. [Zugriff am 18 08 2017].
- [32] «ROS Wiki: Rosout,» [Online]. Available: <http://wiki.ros.org/rosout>. [Zugriff am 18 08 2017].
- [33] «ROS Wiki: Messages,» [Online]. Available: <http://wiki.ros.org/msg>. [Zugriff am 18 08 2017].
- [34] «ROS Wiki: Message Grundtypen,» [Online]. Available: http://wiki.ros.org/msg#Field_Types.
- [35] «Wikipedia: Request-response,» [Online]. Available: <https://en.wikipedia.org/wiki/Request%E2%80%93response>. [Zugriff am 18 08 2017].
- [36] «ROS Docs: Class ServiceProxy,» [Online]. Available: http://docs.ros.org/api/rospy/html/rospy.impl.tcpros_service.ServiceProxy-class.html. [Zugriff am 18 08 2017].
- [37] «ROS Wiki: ActionLib,» [Online]. Available: <http://wiki.ros.org/actionlib/DetailedDescription> . [Zugriff am 18 08 2017].

- [38] «ROS Wiki: Action Lib - Detailed Description,» [Online]. Available: <http://wiki.ros.org/actionlib/DetailedDescription>. [Zugriff am 18 08 2017].
- [39] «OpenCV,» [Online]. Available: <http://opencv.org/>. [Zugriff am 18 08 2017].
- [40] «DLib,» [Online]. Available: <http://dlib.net/>. [Zugriff am 18 08 2017].
- [41] «OpenFace,» [Online]. Available: <https://cmusatyalab.github.io/openface/>. [Zugriff am 18 08 2017].
- [42] D. K. J. P. Florian Schroff, «FaceNet: A Unified Embedding for Face Recognition and Clustering,» 12 05 2015. [Online]. Available: <https://arxiv.org/abs/1503.03832>. [Zugriff am 18 08 2017].
- [43] «Torch,» [Online]. Available: <http://torch.ch/>. [Zugriff am 18 08 2017].
- [44] «Kobuki User Guide,» 15 03 2017. [Online]. Available: https://docs.google.com/document/export?format=pdf&id=15k7UBnYY_GPmKzQCjzRGCW-4dlP7zl_R_7tWPLM0zKI. [Zugriff am 18 08 2017].
- [45] «Wikipedia: Occupancy grid mapping,» [Online]. Available: https://en.wikipedia.org/wiki/Occupancy_grid_mapping. [Zugriff am 18 08 2017].
- [46] «Wikipedia: Monte Carlo localization,» [Online]. Available: https://en.wikipedia.org/wiki/Monte_Carlo_localization. [Zugriff am 18 08 2017].
- [47] C. Stachniss, «Introduction to Robot Mapping,» [Online]. Available: <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam01-intro.pdf>. [Zugriff am 18 08 2017].
- [48] C. Stachniss , «Grid Maps,» [Online]. Available: <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam10-gridmaps.pdf>. [Zugriff am 18 08 2017].
- [49] S. Kohlbrecher, O. von Stryk, J. Meyer und U. Klingauf, «A Flexible and Scalable SLAM System with Full 3D Motion Estimation,» November 2011. [Online]. Available: http://www.sim.informatik.tu-darmstadt.de/publ/download/2011_SSRR_KohlbrecherMeyerStrykKlingauf_Flexible_SLAM_System.pdf. [Zugriff am 18 08 2017].
- [50] G. Grisetti, C. Stachniss und W. Burgar, «GMapping,» [Online]. Available: <http://openslam.org/gmapping.html>. [Zugriff am 18 08 2017].
- [51] C. Stachniss, «Robot Mapping - WS 2013 / 14,» [Online]. Available: <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/>. [Zugriff am 18 08 2017].
- [52] «Wikipedia: Sequenzielle Monte-Carlo-Methode,» [Online]. Available: https://de.wikipedia.org/wiki/Sequenzielle_Monte-Carlo-Methode. [Zugriff am 18 08 2017].
- [53] «Wikipedia: Fitness proportionate selection,» [Online]. Available: https://en.wikipedia.org/wiki/Fitness_proportionate_selection. [Zugriff am 18 08 2017].
- [54] «Stochastic universal sampling,» [Online]. Available: https://en.wikipedia.org/wiki/Stochastic_universal_sampling. [Zugriff am 18 08 2017].
- [55] «Wikipedia: Particle Filter,» [Online]. Available: https://en.wikipedia.org/wiki/Particle_filter. [Zugriff am 18 08 2017].

- [56] C. Stachniss, «Grid-Based FastSLAM,» [Online]. Available: <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam13-gridfastslam.pdf>. [Zugriff am 18 08 2017].
- [57] G. Grisetti, C. Stachniss und W. Burgard, «Improving Grid-based SLAM with Rao-Blackwellized Particle,» 01 2005. [Online]. Available: <http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti05icra.pdf>. [Zugriff am 18 08 2017].
- [58] «ROS Wiki: Robots using Navigation Stack,» [Online]. Available: <http://wiki.ros.org/navigation/RobotsUsingNavStack> . [Zugriff am 18 08 2017].
- [59] «ROS Wiki: Carrot Planner,» [Online]. Available: http://wiki.ros.org/carrot_planner. [Zugriff am 18 08 2017].
- [60] «ROS Wiki: Navfn,» [Online]. Available: <http://wiki.ros.org/navfn>. [Zugriff am 18 08 2017].
- [61] «Wikipedia: Von Neumann Neighborhood,» [Online]. Available: https://en.wikipedia.org/wiki/Von_Neumann_neighborhood. [Zugriff am 18 08 2017].
- [62] «Wikipedia: Moore neighborhood,» [Online]. Available: https://en.wikipedia.org/wiki/Moore_neighborhood. [Zugriff am 18 08 2017].
- [63] «ROS Wiki: Eband local planner,» [Online]. Available: http://wiki.ros.org/eband_local_planner . [Zugriff am 18 08 2017].
- [64] K. Zheng, «ROS Navigation Tuning Guide,» 2 September 2016. [Online]. Available: <http://zkytony.com/documents/navguide.pdf>. [Zugriff am 18 08 2017].
- [65] «ROS Wiki: Rotate Recovery,» [Online]. Available: http://wiki.ros.org/rotate_recovery. [Zugriff am 18 08 2017].
- [66] «ROS Wiki: Clear costmap recovery,» [Online]. Available: http://wiki.ros.org/clear_costmap_recovery. [Zugriff am 18 08 2017].
- [67] B. Yamauchi, «A frontier-based approach for autonomous exploration,» 06 August 2002. [Online]. Available: https://www.cs.cmu.edu/~motionplanning/papers/sbp_papers/integrated1/yamauchi_frontiers.pdf. [Zugriff am 18 08 2017].
- [68] «Wikipedia: Pulsweitenmodulation,» [Online]. Available: <https://de.wikipedia.org/wiki/Pulsweitenmodulation>. [Zugriff am 18 08 2017].
- [69] «Arduino Libraries,» [Online]. Available: <https://www.arduino.cc/en/Reference/Libraries>. [Zugriff am 18 08 2017].
- [70] J. Zhang, «Text Detection in the Wild,» 07 2016. [Online]. Available: http://mcl.usc.edu/wp-content/uploads/2016/07/Junting_Text-Detection-in-the-Wild.pdf. [Zugriff am 18 08 2017].
- [71] «Stackoverflow: Extracting text OpenCV,» [Online]. Available: <https://stackoverflow.com/questions/23506105/extracting-text-opencv>. [Zugriff am 18 08 2017].
- [72] «Text Skew Correction in OpenCV,» [Online]. Available: <http://www.pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/>. [Zugriff am 18 08 2017].

- [73] «ROS Wiki: GMapping,» [Online]. Available: <http://wiki.ros.org/gmapping>. [Zugriff am 18 08 2017].
- [74] «Turtlebot: GMapping Launch File,» [Online]. Available: https://github.com/turtlebot/turtlebot_apps/blob/indigo/turtlebot_navigation/launch/includes/gmapping/gmapping.launch.xml . [Zugriff am 18 08 2017].
- [75] «ROS Wiki: Rosparam - YAML Format,» [Online]. Available: http://wiki.ros.org/rosparam#YAML_Format. [Zugriff am 18 08 2017].
- [76] «Turtlebot: Navigation Parameter,» [Online]. Available: https://github.com/turtlebot/turtlebot_apps/blob/indigo/turtlebot_navigation/param/. [Zugriff am 18 08 2017].
- [77] «ROS Wiki: Navigation Tuning Guide,» [Online]. Available: <http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide>. [Zugriff am 18 08 2017].
- [78] «ROS Wiki: Kobuki Node,» [Online]. Available: http://wiki.ros.org/kobuki_node. [Zugriff am 18 08 2017].
- [79] «Autobahn Libraries,» Crossbar.io, [Online]. Available: <http://crossbar.io/autobahn/>. [Zugriff am 18 08 2017].
- [80] «Bootstrap,» [Online]. Available: <http://getbootstrap.com/>. [Zugriff am 18 08 2017].
- [81] «Gitlab: IANA IP5 - FaceDetectionValidation,» [Online]. Available: <https://gitlab.fhnw.ch/daniel.kroeni/iana/tree/master/FaceDetectionValidation>. [Zugriff am 18 08 2017].
- [82] R. Rothe, R. Timofte und L. Van Gool, «IMDB-WIKI – 500k+ face images with age and gender labels,» 2015. [Online]. Available: <https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/>. [Zugriff am 18 08 2017].
- [83] R. Rothe, R. Timofte und L. Van Gool, «Deep expectation of real and apparent age from a single image without facial landmarks,» *International Journal of Computer Vision (IJCV)*, 07 2016.
- [84] R. Rothe, R. Timofte und L. Van Gool, «DEX: Deep EXpectation of apparent age from a single image,» in *IEEE International Conference on Computer Vision Workshops (ICCVW)*, Santiago, Chile, 2015.
- [85] S. Russell und P. Norvig, «Kapitel 2.2.1 Rationalität,» in *Künstliche Intelligenz - Ein moderner Ansatz, 3. Auflage*, Pearson, 2011, p. 64.
- [86] S. Russell und P. Norvig, «Kapitel 2.2.2 Allwissenheit, Lernen und Autonomie,» in *Künstliche Intelligenz - Ein moderner Ansatz, 3. Auflage*, Pearson, 2011, p. 65.
- [87] «RobotSetup,» [Online]. Available: <http://wiki.ros.org/navigation/Tutorials/RobotSetup> . [Zugriff am 18 08 2017].

10 Abbildungsverzeichnis

Abbildung 1: Kobuki Detailansicht	11
Abbildung 2: Control Panel von Kobuki mit den 4 Stromanschlüssen, dem USB- und Serial-Port.....	11
Abbildung 3: Aufnahme eines FHNW Türschildes mit der ZED	12
Abbildung 4: Aufnahme eines FHNW Türschildes mit der D-Link Highspeed Kamera mit Objektiv.	13
Abbildung 5: Catkin Workspaces.....	15
Abbildung 6: Kommunikation zwischen verteilten ROS Nodes über das Netzwerk.....	15
Abbildung 7: Ein Beispiel einer ROS Kommunikation über eine Message.	16
Abbildung 5: Kommunikation zwischen Action Client und Action Server (Quelle: [38]).....	17
Abbildung 9: IANAs Komponenten und deren Interaktion	18
Abbildung 10: Fehler des Gyro-Sensors pro Umdrehung bei unterschiedlichen Geschwindigkeiten (Quelle: Kobuki User Guide).....	21
Abbildung 11: Messergebnisse des Gyro-Fehlers nach 5 Umdrehungen.....	22
Abbildung 9: Interaktion der SLAM Komponente (GMapping) mit der ROS-Ausenwelt.....	23
Abbildung 10: Aufbau der Navigation Stack Komponente [87].....	25
Abbildung 14: Suboptimaler Weg bei Verwendung des A* mit Von Neumann Neighborhood (Quelle: http://wiki.ros.org/global_planner#A.2A)	26
Abbildung 15: Übersicht über die Controller Komponente.....	27
Abbildung 16: Task Zustandsdiagramm	28
Abbildung 17: Übersicht über die IANA Navigation Komponente	30
Abbildung 18: Übersicht über die Collision Detector Komponente	30
Abbildung 19: Übersicht über die Random Explorer Komponente	32
Abbildung 20: IANA Random Explorer Zustandsdiagramm.....	32
Abbildung 21: Übersicht über die Rational Explorer Komponente	35
Abbildung 22: Grid-Map als Input des Detect Frontiers Algorithmus	36
Abbildung 23: Resultat des Detect Frontiers nach Anwendung auf die Grid-Map aus Abbildung 22...36	36
Abbildung 24: Erweiterter Rational Explorer Zustandsdiagramm.....	38
Abbildung 25: Übersicht über die Person Detection Komponente	38
Abbildung 26: Ansteuerung der Servos via Kobukis Digital Output Pins.....	40
Abbildung 27: Das Signal auf 2 und 1 Millisekunde ist nicht konstant und schwankt stark (rot markiert)	41
Abbildung 28: Ansteuerung der Servos via Arduino Board	41
Abbildung 29: Schematische Darstellung der Pan-Ausrichtung	43

Abbildung 30: Übersicht über Benutzerschnittstelle Komponente.....	43
Abbildung 28: Aufnahme FHNW Türschild mit DLink Kamera	44
Abbildung 32: Das Text Detection Prinzip in der Übersicht	45
Abbildung 33: Beispiel für die Erkennung von Namen auf einem FHNW Türschild	46
Abbildung 34: Ausrichtung eines erkannten Wortes	46
Abbildung 35: Verbinden von zusammengehörenden Zeilen	46
Abbildung 36: Beispiel für die Erkennung von Raumnummer auf einem FHNW Türschild	47
Abbildung 37: Umwandlung zu Binärbild und Ausrichtung.....	47
Abbildung 38: Beispiel für die Ausrichtung des Textes «FHNW»	48
Abbildung 39: Abonnierte und publizierte ROS Topics der SLAM Komponente	49
Abbildung 40: IANA Benutzerschnittstelle	59
Abbildung 41: Popup mit Vorschaubild und Inputfeld zur Eingabe des Namens.....	59
Abbildung 42: Anzahl Zeilen Code unseres Git Repositories (mit „ <code>colc \$(git ls-files)</code> “ berechnet)	64
Abbildung 43: Tiefenscan-Sensoren Evaluation Testszenario 1 mit IANA.....	65
Abbildung 44: ZED Tiefenscan	66
Abbildung 45: KinectOne Tiefenscan	66
Abbildung 46: RPLidar Tiefenscan	66
Abbildung 47: ZED Map	66
Abbildung 48: KinectOne Map	66
Abbildung 49: RPLidar Map	66
Abbildung 50: Tiefenscan-Sensoren Evaluation Testszenario 2 mit IANA.....	67
Abbildung 51: ZED Tiefenscan	67
Abbildung 52: KinectOne Tiefenscan	67
Abbildung 53: RPLidar Tiefenscan	67
Abbildung 54: Grundrissplan des Testraums.....	68
Abbildung 55: Grundrissplan des Testraums mit Hindernissen	69
Abbildung 56: Kinect SLAM Map mit offener Fensterfront.....	70
Abbildung 57: Kinect SLAM Map mit geschlossener Fensterfront (1).....	70
Abbildung 58: Kinect SLAM Map mit geschlossener Fensterfront (2), nach 11 Minuten	71
Abbildung 59: Kinect SLAM Map mit geschlossener Fensterfront (2), nach 5 Minuten	71
Abbildung 60: RPLidar SLAM Map mit offener Fensterfront.....	71
Abbildung 61: RPLidar SLAM Map mit geschlossener Fensterfront.....	72

Abbildung 62: RPLidar SLAM Map im Testraum mit Hindernissen ohne Fensterfront	72
Abbildung 63: Entwicklung der SLAM Karte im Verlauf der Zeit (KinectOne vs. RPLidar).....	73
Abbildung 64: Navigation Testszenario.....	74
Abbildung 65: Pfad und Map nach 5 Minuten Random Explorer.....	76
Abbildung 66: Map nach 1.5 Minuten (Random Explorer)	76
Abbildung 67: Map nach 3 Minuten (Random Explorer).....	76
Abbildung 68: Map nach 5 Minuten (Random Explorer).....	76
Abbildung 69: Pfad und Map nach 5 Minuten Rational Explorer.....	77
Abbildung 70: Map nach 1.5 Minuten (Rational Explorer).....	77
Abbildung 71: Map nach 3 Minuten (Rational Explorer).....	77
Abbildung 72: Map nach 5 Minuten (Rational Explorer).....	77
Abbildung 73: Originalbild von 01.jpg	79
Abbildung 74: Nummer-Erkenner in 01.jpg	79
Abbildung 75: Namens-Erkenner in 01.jpg.....	79
Abbildung 76: Originalbild von 02.jpg	79
Abbildung 77: Nummer-Erkenner in 02.jpg	79
Abbildung 78: Namens-Erkenner in 02.jpg.....	79
Abbildung 79: Originalbild von 03.jpg	80
Abbildung 80: Nummer-Erkenner in 03.jpg	80
Abbildung 81: Namens-Erkenner in 03.jpg.....	80
Abbildung 82: Originalbild von 04.jpg	80
Abbildung 83: Nummer-Erkenner in 04.jpg	80
Abbildung 84: Namens-Erkenner in 04.jpg.....	80
Abbildung 85: Originalbild von 05.jpg	80
Abbildung 86: Nummer-Erkenner in 05.jpg	80
Abbildung 87: Namens-Erkenner in 05.jpg.....	80
Abbildung 88: Originalbild von 06.jpg	80
Abbildung 89: Nummer-Erkenner in 06.jpg	80
Abbildung 90: Namens-Erkenner in 06.jpg.....	80
Abbildung 91: Originalbild von 07.jpg	81
Abbildung 92: Nummer-Erkenner in 07.jpg	81
Abbildung 93: Namens-Erkenner in 07.jpg.....	81

Abbildung 94: Originalbild von 08.jpg	81
Abbildung 95: Nummer-Erkenner in 08.jpg	81
Abbildung 96: Namens-Erkenner in 08.jpg.....	81
Abbildung 97 Nummer-Erkenner in 01.jpg	81
Abbildung 98: Namens-Erkenner in 01.jpg.....	81
Abbildung 99: Nummer-Erkenner in 01.jpg	82
Abbildung 100: Namens-Erkenner in 02.jpg.....	82
Abbildung 101: Nummer-Erkenner in 03.jpg	82
Abbildung 102: Namens-Erkenner in 03.jpg.....	82
Abbildung 103: Nummer-Erkenner in 04.jpg	83
Abbildung 104: Namens-Erkenner in 04.jpg.....	83
Abbildung 105: Nummer-Erkenner in 05.jpg	83
Abbildung 106: Namens-Erkenner in 05.jpg.....	83
Abbildung 107: Nummer-Erkenner in 06.jpg	84
Abbildung 108: Namens-Erkenner in 06.jpg.....	84
Abbildung 109: Nummer-Erkenner in 07.jpg	84
Abbildung 110: Namens-Erkenner in 07.jpg.....	84
Abbildung 111: Nummer-Erkenner in 08.jpg	85
Abbildung 112: Namens-Erkenner in 08.jpg.....	85
Abbildung 113: IANA in Gang der FHNW	91