

Technische Spezifikation

Fennec Racer

Autor: Lukas Evers, Umut Uzunoglu, Son Khue Nguyen, Hien Anh Nguyen Manh
Letzte Änderung: 21.07.21 22:31
Dateiname: Technische_Spezifikation_Fennec_Bot_Sprint_3.pdf

Version: 1.0

Inhaltsverzeichnis

1	Vorhandene Dokumente	4
2	Prozessüberblick	5
2.1	Fachlicher Workflow	5
2.2	Technischer Workflow	6
3	Technische Spezifikation SW	8
3.1	Überblick Komponenten	8
3.2	Beschreibung der Implementierung	10
3.2.1	Funktion 1	10
3.2.2	Funktion 2	12
3.3	System Infrastruktur	13
4	Technische Spezifikation Konstruktion	14
4.1	Baugruppen	14
4.2	Einzelteile	15
4.3	Berechnungen	15
5	Offene Fragen	16
6	Modul Abhängigkeiten	17

Abbildungsverzeichnis

Abbildung 1: Nodes, Topics und Messages	5
Abbildung 2: Fachlicher Workflow	7
Abbildung 3: Komponentendiagramm	8
Abbildung 4: AMCL	11
Abbildung 5: ROS Navigation	12
Abbildung 6: Ballerkennung	14
Abbildung 7: Rviz	16
Abbildung 8: Torerkennung	18
Abbildung 9: Torerkennung in Rviz	19
Abbildung 10: Systeminfrastruktur	22
Abbildung 11: Technische Zeichnung	23

Tabellenverzeichnis

Tabelle 1: Vorhandene Dokumente	5
Tabelle 2: Komponente Erfasste Funktionen	9
Tabelle 3: Funktion 1	10
Tabelle 4: Funktion 2	12
Tabelle 5: ROS Navigation	12

Copyright

© Team Fennec Bot

Die Weitergabe, Vervielfältigung oder anderweitige Nutzung dieses Dokumentes oder Teile davon ist unabhängig vom Zweck oder in welcher Form untersagt, es sei denn, die Rechteinhaber/In hat ihre ausdrückliche schriftliche Genehmigung erteilt.

Version Historie

Version:	Datum:	Verantwortlich	Änderung
0.1	14.07.2021	Lukas Evers	Fachlicher Workflow, Überblick der Komponenten, Systeminfrastruktur/ ROS Multiple Machines Setup
0.2	20.07.2021	Umut Uzunoglu	Beschreibung der Implementierung, Erklärung wichtiger Begriffe
1.0	21.07.2021	Lukas Evers, Umut Uzunoglu, Son Khue Nguyen, Hien Anh Nguyen Manh	Ergänzungen und Korrektur

1 Vorhandene Dokumente

Alle für die vorliegende Spezifikation ergänzenden Unterlagen müssen hier aufgeführt werden

Tabelle 1: Vorhandene Dokumente

Dokument	Autor	Datum
Lastenheft_Fennec_Bot.pdf	Team Fennec Bot	02.05.21
Pflichtenheft_FennecBot.docx.pdf	Team Fennec Bot	19.05.21
Qualitätssicherung_FennecBot.docx.pdf	Team Fennec Bot	17.06.21

1.1 Erklärung wichtiger Begriffe

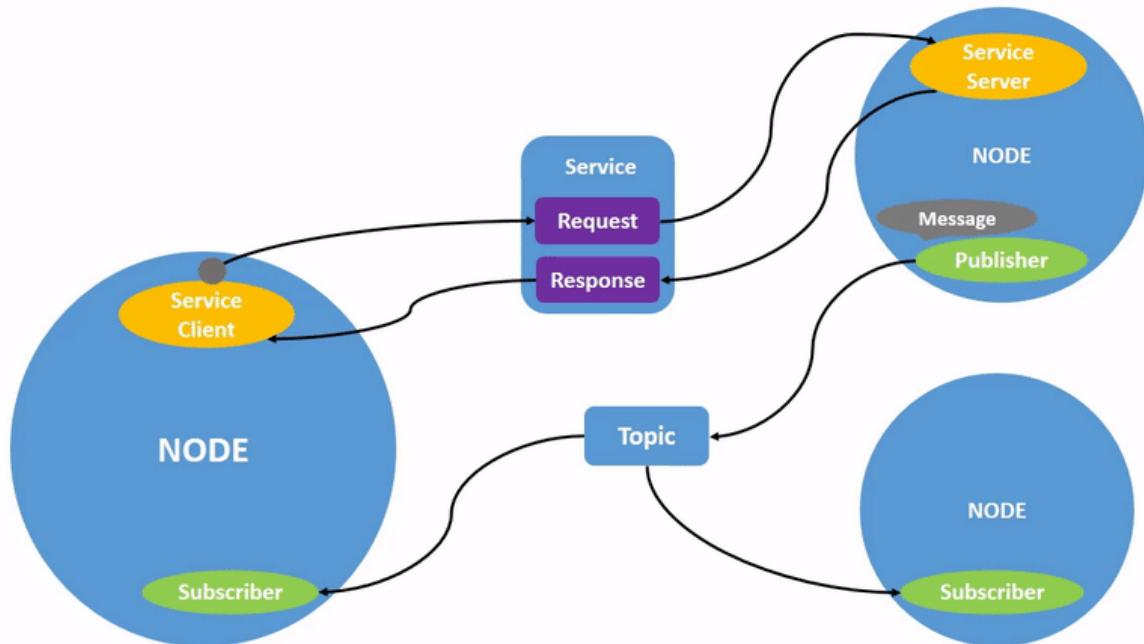


Abbildung 1: Nodes, Topics und Messages

Node: Nodes in ROS werden über Python- und C++- Programme gestartet. Im Grunde genommen sollen Nodes bestimmte Funktionen erfüllen, wie zum Beispiel Objekterkennung oder Steuerung des Roboters. Dabei können Nodes auch mit anderen Nodes kommunizieren, wobei die Programmiersprache (C++ oder Python) nicht entscheidend ist. Dies geschieht über Topics.

Topic: Die Aufgabe von Topics ist es, gewisse Daten, die eine Node erstellt, über einen Publisher zu publishen. Diese Daten werden auch Messages genannt. Nun ist es für eine andere Node möglich, diese Messages über einen Subscriber abzugreifen und diese zu verarbeiten.

Message: Messages sind nichts anderes als Daten, die von Topics gepubliziert werden. Wenn Messages von Nodes und Topics aus verschiedenen Programmiersprachen gepubliziert werden, dann muss darauf geachtet werden, dass diese Messages in einem ROS "neutralen" Typen gepubliziert werden. Beispielsweise ist "Twist" sowohl in C++ als auch in Python ein Typ den man einer Callback- Methode übergeben kann. Das heißt, dass man über einen Subscriber auf diesen Messagetypen von anderen Nodes aus zugreifen kann.

Frames: Frames in ROS bezeichnen eine Transformation im Raum bezogen auf einen anderen Frame. Wenn wir die Basis des Fennecs als eigenen Frame bezeichnen und den Nullpunkt der Welt in der Visualisierung auch als eigenes Frame bezeichnen, kann ROS über eine Transformation die Position und die Orientierung der Basis festlegen. Die Transformation erfolgt auch zu Frames, die nicht direkt mit dem Fennec verbunden sind.

/cmd_vel: /cmd_vel ist für die eigentliche Bewegung des Roboters verantwortlich. Auf diesem Topic werden Twist Messages gepubliziert, die der Hardware übergeben werden. Die wichtigsten Bestandteile der Message für einen Roboter, der sich auf Räder fortbewegt sind die linearen Werte auf der x- Achse, für Vorwärts- und Rückwärtsbewegungen und die angulare Werte auf der z- Achse, die für die Rotation der Roboter von Bedeutung sind.

tf: tf ist eine Transformationen von einem Frame zum Anderen. Dadurch besitzen alle Frames eine relative Pose (Position und Orientierung) und eine Absolute. Die relative Transformation ist im eigenen Frame, wobei die absolute im Bezug zur Welt ist.

2 Prozessüberblick

2.1 Fachlicher Workflow

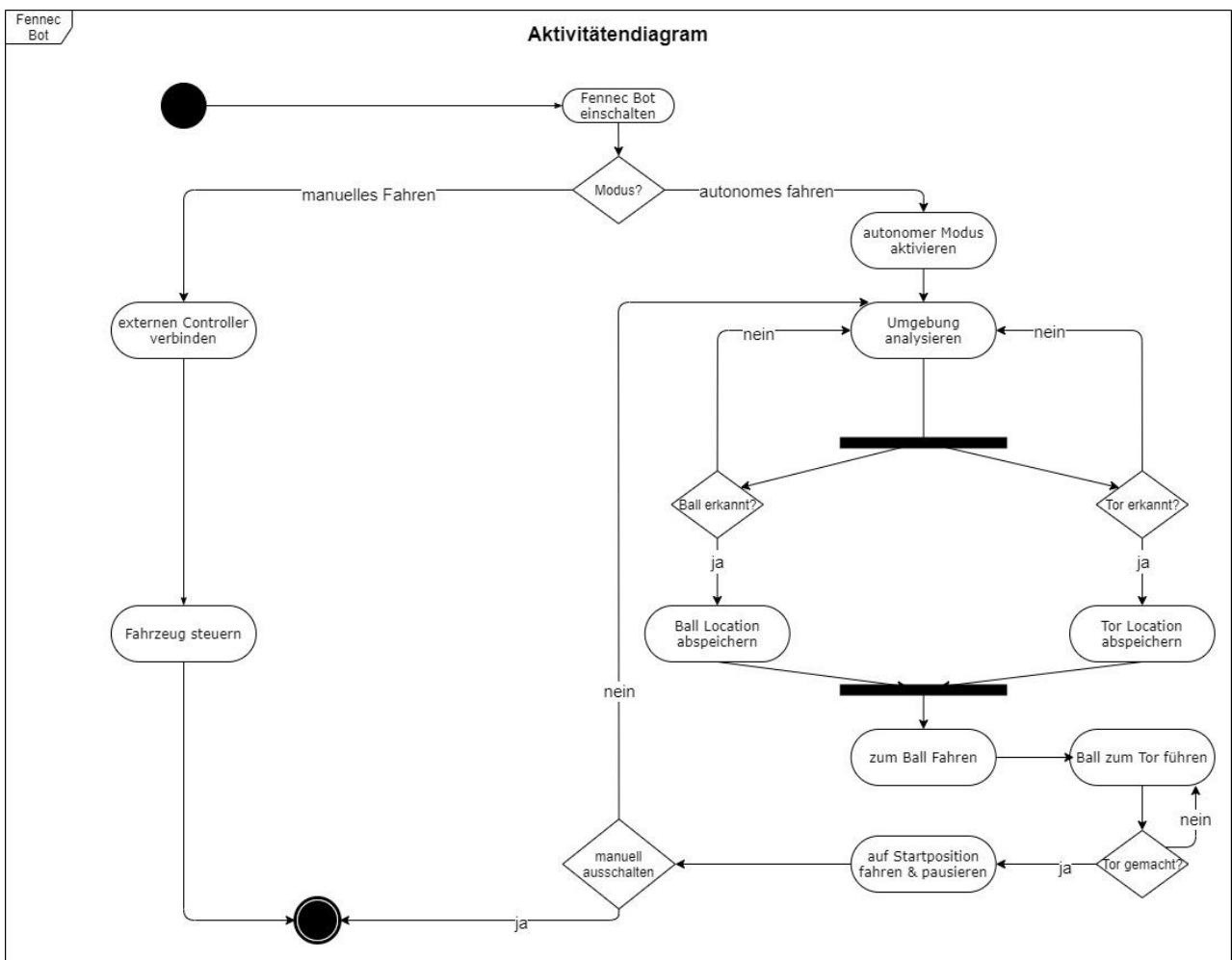


Abbildung 2: Fachlicher Workflow

3 Technische Spezifikation SW

3.1 Überblick Komponenten

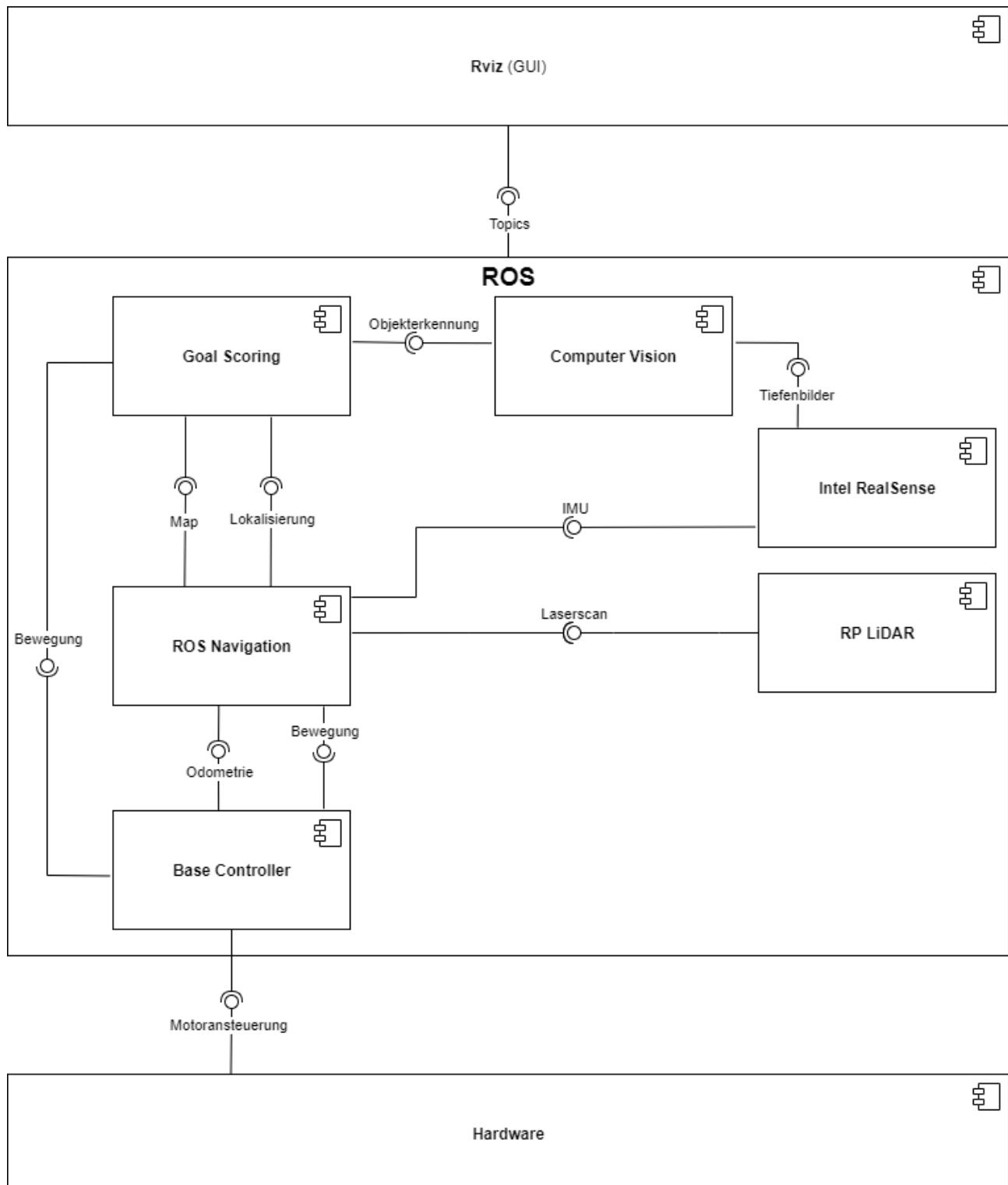


Abbildung 3: Komponentendiagramm

Tabelle 2: Komponenten Erfasste Funktionen

Komponente	Erfasste Funktion aus dem Pflichtenheft
Base Controller	Funktion 3: Gegen Ball fahren und schießen Funktion 4: Hindernisse umfahren Funktion 7: Fernsteuerung
Hardware	Funktion 5: Automatisch abschalten
LiDAR	Funktion 1: Umgebung scannen Funktion 2: Ball autonom suchen und erkennen Funktion 4: Hindernisse umfahren
Intel RealSense	Funktion 1: Umgebung scannen Funktion 2: Ball autonom suchen und erkennen Funktion 4: Hindernisse umfahren Funktion 6: Tore erkennen und auseinanderhalten
ROS Navigation	Funktion 1: Umgebung scannen Funktion 3: Gegen Ball fahren und schießen Funktion 4: Hindernisse umfahren
Computer Vision	Funktion 2: Ball autonom suchen und erkennen Funktion 6: Tore erkennen und auseinanderhalten
Goal Scoring	Funktion 3: Gegen Ball fahren und schießen Funktion 6: Tore erkennen und auseinanderhalten

3.2 Beschreibung der Implementierung

3.2.1 Funktion 1: Umgebung scannen

Tabelle 3: Funktion 1

#	Komponentendetail	Erforderliche Arbeiten
T1	RPLiDAR-A1(LiDAR Laserscanner)	Die LiDAR muss auf das Gehäuse des Fennecs gesetzt werden. Dadurch wird der LiDAR die Möglichkeit gegeben, die Umgebung um 360° zu scannen.
T2	SLAM Gmapping	Die Scandaten vom LiDAR werden dann weiter an das SLAM Gmapping- Paket von ROS übergeben, wodurch eine Karte der Umgebung erstellt wird. Die Karte wird dann im /map- Topic als Message gepublisiert.
T3	AMCL	Die gepublisierte Karte vom Gmapping kann dann AMCL übergeben werden, wodurch der Fennec auf der Karte lokalisiert wird.

T1: LiDAR

Um den Laserscanner an ROS anzubinden ist das ROS- Paket "rplidar_ros" notwendig. Über eine Node (Python/C++ Skripte) ist es möglich, die Laserscandaten abzugreifen und in ROS Messages umzuwandeln. Diese Messages werden über einen ROS Topic gepubliziert (Publisher). Das heißt, andere Nodes, die auf diese Daten zugreifen möchten, müssen einen "Subscriber" beinhalten, der dann den Laserscanner- Topic abonniert (subscribed).

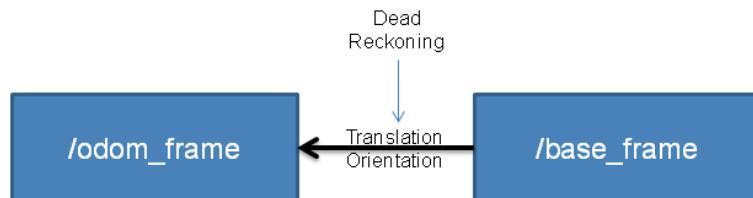
T2: SLAM Gmapping (Simultaneous Localization and Mapping)

Nachdem der Laserscanner über eine Node gestartet wurde, wird das SLAM Gmapping- Paket gestartet. Dabei abonniert die Node vom Gmapping- Paket dem Topic vom Laserscanner. Die Messages vom Laserscanner- Topic werden dann zur Erstellung einer Karte benutzt, welche anschließend gepubliziert wird. Dabei wird die Karte immer wieder aktualisiert und gepubliziert, da sich der Fennec bewegt. Dadurch wird die Karte ausgebaut und der Fennec weiß immer wo er gerade ist.

T3: AMCL (adaptive Monte Carlo localization)

AMCL abonniert das /map-Topic und schätzt eine ungefähre Pose im Bezug zur laserbasierten Karte. Dabei beachtet AMCL, ob es einen Drift in der Pose basierend auf der Odometrie gibt. Falls es einen Drift geben sollte, dann wird ein Transform zwischen dem Odometrie- Frame und dem Map- Frame berechnet, der dann die echte Pose, also zwischen dem Map- Frame und dem Base_Link- Frame des Fennecs feststellt. Base_Link- Frame ist der Frame an dem die Basis des Fennecs gebunden ist.

Odometry Localization



AMCL Map Localization

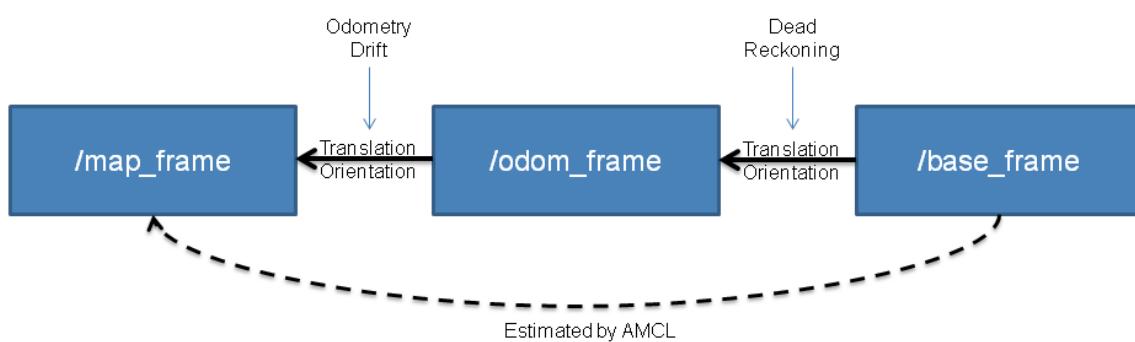


Abbildung 4: AMCL <http://wiki.ros.org/amcl>

T4: ROS Navigation Stack

Die Zusammenarbeit der obigen Komponenten ermöglicht dem Navigation Stack die Pfadplanung, somit auch das Publishen von Twist Messages auf dem /cmd_vel Topic. Dadurch, dass der Fennec dank AMCL und der gegebenen Karte, die durch den Laserscanner erstellt wurde, kann der Pfadplaner vom Navigation Stack den Fennec durch die Welt führen. Der Laserscanner aktualisiert die Karte regelmäßig, um ROS Navigation neue Informationen von der Umwelt zu übergeben. AMCL hilft dabei, dass ROS Navigation die ungefähre Pose des Roboters zu finden.

In der Abbildung 5 kann man die einzelnen Nodes, die miteinander kommunizieren genauer betrachten. Die weißen Nodes sind vom Navigation stack schon gegeben oder werden vorher schon durch Konfigurationsdateien festgelegt/konfiguriert (die Costmaps zum Beispiel). Die blauen Nodes müssen selber implementiert werden und die grauen Nodes müssen nicht implementiert werden damit ROS Navigation funktionsfähig ist.

Die Costmaps werden zum Speichern von Informationen, wie zum Beispiel Hindernisse, benutzt. Dabei speichert "global_costmap" langzeit Informationen, wie zum Beispiel Wände und "local_costmap" Informationen die direkt vor dem Fennec geschehen. Diese werden dann zum Umgehen von Hindernissen benutzt.

Die Node "sensor sources" bezieht sich auf den Laserscanner und der Intel Realsense, die es überhaupt möglich machen, Hindernisse zu identifizieren und auch den Fennec auf der Karte zu lokalisieren.

Wichtig für das Lokalisieren ist auch die "odometry source"- Node. ROS Navigation benutzt "tf" (Transforms) um die Pose des Basis des Fennecs festzustellen. Jedoch beinhaltet "tf" keine Geschwindigkeit. Dafür ist auch der "odom"- Frame und Node da. Die "tf" - Transformationen werden über diesen Frame gepublisiert. Gleichzeitig werden aber auch Odometrie Messages in Form von "nav_msgs/Odometry", die die Geschwindigkeit des Roboters neben der "tf"- Transformation publisht, gepublisiert.

Die "Planner" nutzen all die "tf" und Odometrie Informationen um dem "base_controller" eine Twist Message zu übergeben. Dies geschieht über das /cmd_vel- Topic.

Der "base_controller" übergibt die Messages weiter an die Hardware.

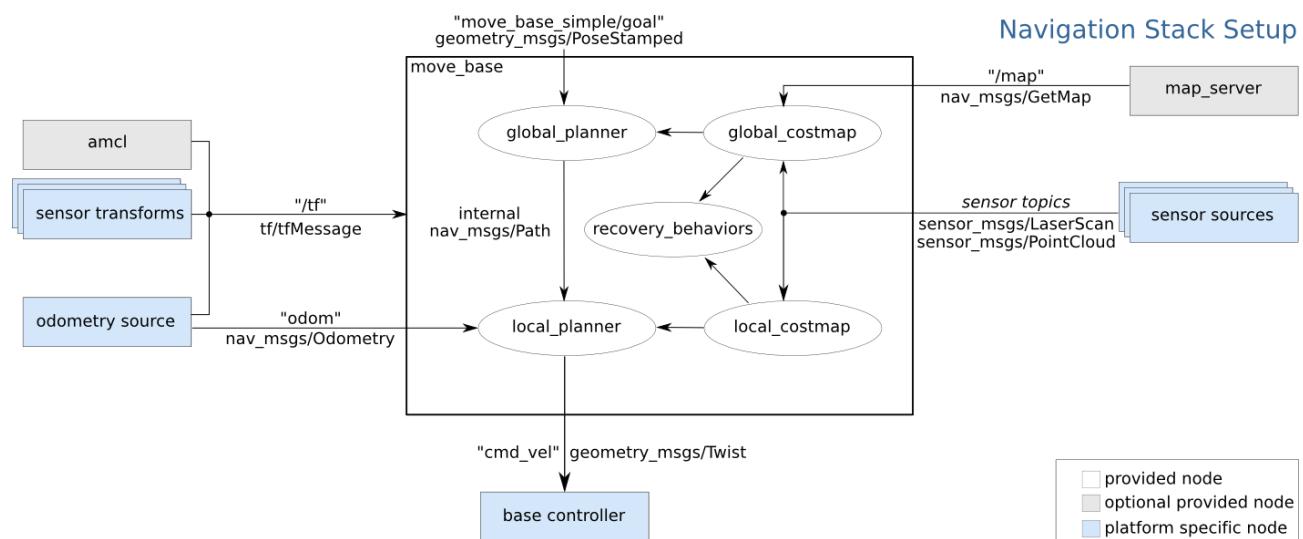


Abbildung 5: ROS Navigation [navigation/Tutorials/RobotSetup - ROS Wiki](#)

3.2.2 Funktion 2: Fernsteuerung

Tabelle 4: Funktion 2

#	Komponentendetail	Erforderliche Arbeiten
T1	Wireless Controller	Der Wireless Controller muss zunächst über Bluetooth an den Jetson Nano im Fennec verbunden werden. Die Hauptaufgabe des Controllers ist es, den User Input an das Betriebssystem zu schicken.
T2	ROS- Paket Joy	Nachdem der User eine Taste/Button/Stick betätigt, wird der Input in ROS Messages umgewandelt, die der Hardware übergeben wird.

T1: Wireless Controller

Die Verbindung des Controllers an den Jetson Nano erfolgt auf simple Art und Weise. Der User muss den Controller erstmal einschalten in dem er den Schalter auf der Rückseite betätigt und anschließend auf den Home- Button drückt. Die Hauptaufgabe des Controllers ist, jede Art von Bewegung (in unserem Fall die Sticks) an ROS zu schicken.

T2: ROS- Paket Joy

Dieses Paket ermöglicht es Teleoperation durchzuführen. Im Grunde genommen heißt Teleoperation, den Fennec fernzusteuern. Der User Input wird an das Betriebssystem via Bluetooth geschickt. Diese werden dann vom Paket in ROS Messages umgewandelt und auf dem /cmd_vel Topic gepubliziert. Diese Messages werden dann der Hardware übergeben und der Fennec setzt sich in Bewegung.

3.2.3 Funktion 3: Gegen Ball fahren und schießen

Tabelle 5: Funktion 3

#	Komponentendetail	Erforderliche Arbeiten
T1	Intel Realsense	Die Intel Realsense wird vorne am Fennec festgeschraubt und hilft dabei den Ball vor der Kamera zu erkennen.
T2	Ballchasing Paket	Übernimmt Informationen von OpenCV und gibt dem Fennec Steuerbefehle weiter

T1: Intel Realsense

Die Intel Realsense wird hier als simples Tool zur Bilderkennung verwendet. Dabei setzen wir die Kameraauflösung auf 640x480 Pixel. Dadurch spart man Ressourcen. Falls ein, von uns vorgegebener Ball, vor der Kamera steht, wird in der GUI ein roter Kreis um den Ball gezeichnet. Dieser signalisiert die Position im Bild. Die Pixelkoordinaten werden als x-y- Werte dem Fennec übergeben.

T2: Ballchasing Paket

Die Pixelkoordinaten von **T1: Intel Realsense** werden dann in links und rechts Werte unterteilt. Dadurch versucht der Fennec den Ball zentral zu treffen und korrigiert im Fall von Abweichungen. Die Befehle, die übergeben werden, werden als Twist Messages übergeben.

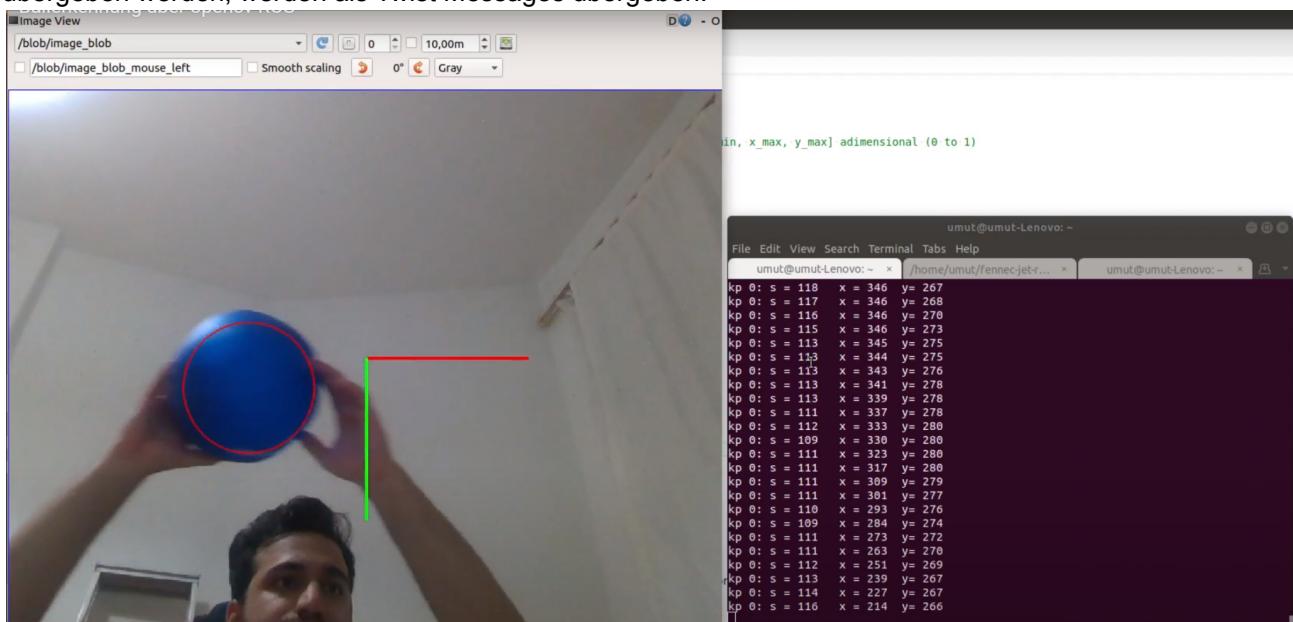


Abbildung 6: Ballerkennung

3.2.4 Funktion 4: Hindernisse umfahren

Tabelle 6: Funktion 4

#	Komponentendetail	Erforderliche Arbeiten
T1	rf2o_laser_odometry Paket	Erzeugt Odometrie über Laserscandaten
T2	RP LiDAR Laserscanner	Sendet Laserscandaten ständig an das Gmapping Tool
T3	SLAM Gmapping	Kartiert die Umwelt dank den übergebenen Laserscandaten des RP LiDARs
T4	TEB Local Planner	Übernimmt die Pfadplanung für einen gegebenen Punkt in der Karte
T5	ROS Navigation Stack	ROS Navigation Stack ist für die Pfadplanung und das Umgehen von Objekten verantwortlich

T1: rf2o_laser_odometry

Das rf2o- Paket kann durch Änderungen der Position der Laserscandaten die Odometrie berechnen und auf dem /odom- Topic veröffentlichen. Gleichzeitig wird auch auf dem /odom- Frame die Translation vom /base_link- Frame zum /odom- Frame veröffentlicht. Dadurch wird Drift beim Kartieren minimal gehalten.

T2: RP LiDAR Laserscanner

Der Laserscanner übernimmt das Veröffentlichen von Laserscandaten auf dem /scan- Topic. Dadurch können andere Komponente, wie zum Beispiel **T1**, auf diese Daten zugreifen. Außerdem nimmt auch das Gmapping Tool (SLAM) diese Werte auf. Wichtig hierbei ist, dass die vom Laserscan erkannten Wände/Gegenstände von ROS- Navigation als Hindernis erkannt werden, die vom Pfadplaner umgangen werden. Außerdem hat der Laserscanner ein eigenes Frame, welches zeigt, dass der Laserscanner auf dem Fennec sitzt und nicht im Fennec.

T3: SLAM Gmapping

Damit das Kartieren und Lokalisieren durch SLAM brauchbar werden können, braucht das Gmapping Tool Laserscandaten und Odometriedaten. Diese werden von den Komponenten **T1** und **T2** geliefert. Falls Odometriedaten nicht vorhanden sind, wird nur der erste Laserscan zu Nutzen gemacht, dadurch Fehlt die Aktualisierung der Karte, um Hindernisse zu umfahren. Falls ein Objekt vor dem Laserscanner gestellt wird, wird dieses Objekt in Rviz als Hindernis/Wand dargestellt.

T4: TEB Local Planner

TEB (kurz für Timed-Elastic-Band) ist ein Pfadplaner, der dabei hilft, den berechneten Pfad zu optimieren. Mit den Beispieldpfadplanern ist es nicht möglich, dass der Fennec zur Korrektur rückwärts fährt, jedoch macht das TEB möglich. Auch erlaubt es TEB erkannte Objekte in der Costmap (Lokale Karte durch den Laserscanner) zu umfahren.

T5: ROS Navigation Stack

Das ROS Navigation Stack spielt die Hauptrolle beim Umfahren der Hindernisse. Hier werden die anderen Komponenten **T1**, **T2**, **T3** und **T4** vereint. Alle Frames und Topics, die in den anderen Komponenten erwähnt worden sind, sind nötig, damit die Pfadplanung über TEB Local Planner ermöglicht werden kann. Wenn die anderen Komponenten problemlos funktionieren, kann über Rviz ein "Navigation Goal" gesetzt werden und der TEB Local Planner plant, in Abhängigkeit von vielen gesetzten Parametern, wie zum Beispiel base_footprint und min_obstacle_dist, den Weg und die Geschwindigkeit zum Ziel. Am Ende wird, falls das Ziel erreicht wird "Goal reached" ausgegeben. Falls nicht, steht die entsprechende Fehlermeldung im Terminal.

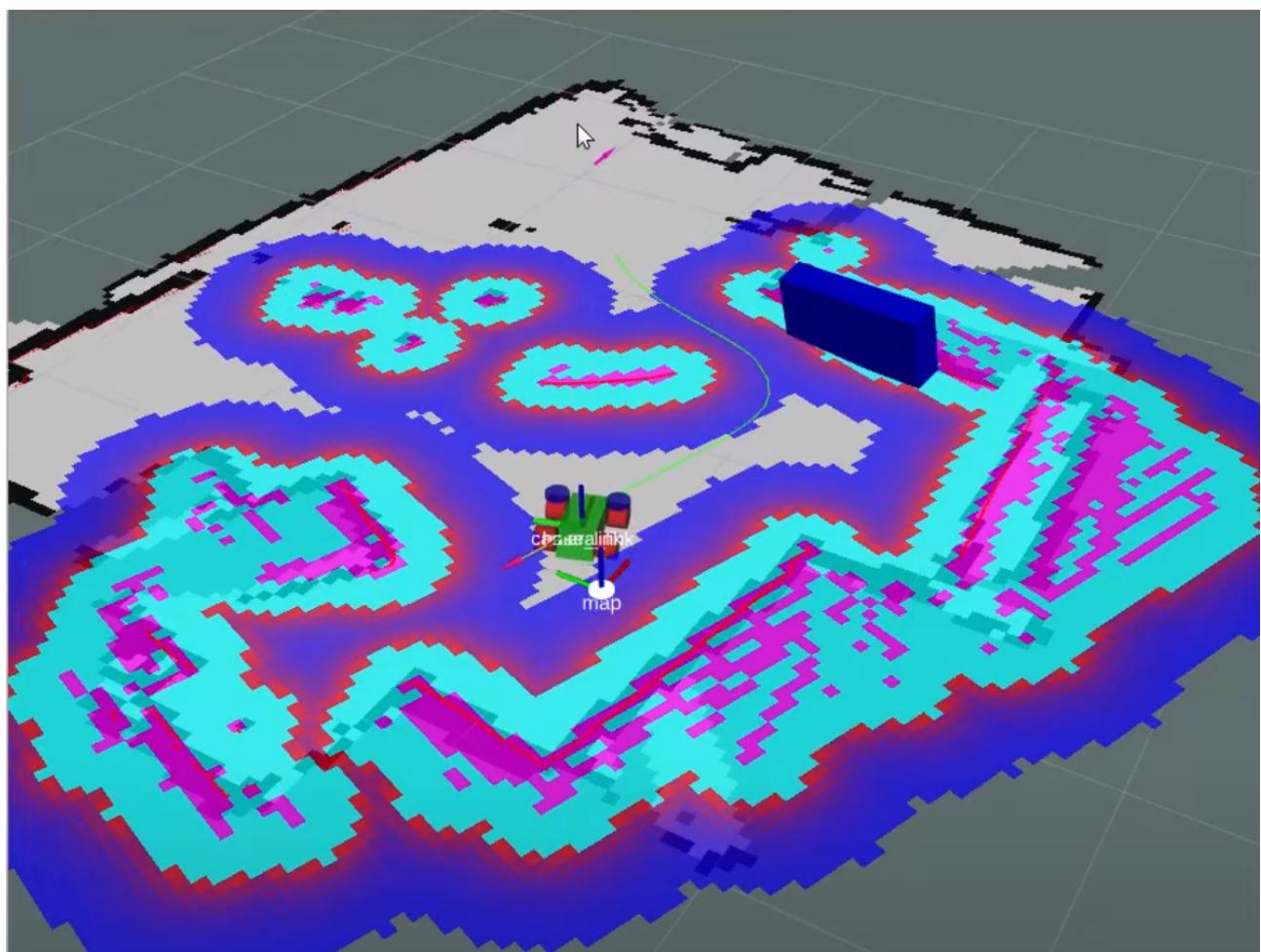


Abbildung 7: Rviz

3.2.5 Funktion 5: Automatisch abschalten

Tabelle 7: Funktion 5

#	Komponentendetail	Erforderliche Arbeiten
T1	Jetson Nano	Abschalten bei Überhitzung

T1: Jetson Nano

Der Jetson Nano selbst hat die benötigten Komponente, um die eigene CPU und GPU Temperatur auszulesen. Mit dem befehl “sudo jtop” werden die technischen Daten des Systems angezeigt. Darunter befindet sich auch die momentane Temperatur. Falls es zur Überhitzung kommen sollte, was bis jetzt nicht der Fall, schaltet sich der Jetson Nano ohne jeglichen Einfluss selbst aus.

3.2.6 Funktion 6: Tore erkennen und auseinanderhalten

Tabelle 8: Funktion 6

#	Komponentendetail	Erforderliche Arbeiten
T1	Intel Realsense	Durch die Farbbilder und Tiefendaten der Intel Realsense ist es möglich die Position der Tore zu finden
T2	find_2d/3d_object Paket	Bilderkennung für die Tore und darstellen der Torposition in Abhängigkeit der Kamera und der Karte durch TF-Transformation

T1: Intel Realsense

Die Intel Realsense hat lediglich die Aufgabe Farbbilder und Tiefendaten an **T2** zu schicken.

T2: find_2d/3d_object

Dieses Paket erlaubt es uns Bilder von Objekten zu machen und zu speichern. Diese Objekte kriegen als Namen "object" als Prefix und dahinter die Nummer, z. B. "object_1." Die Tiefendaten der Intel Realsense werden dann zum Messen der Distanz zum Objekt in Abhängigkeit des Kameraframes, welches an den Fennec angebunden ist. Falls das Frame existiert, wird an der Stelle in Rviz ein Rechteck, welches das Tor darstellen soll, platziert.



Abbildung 8: Torerkennung

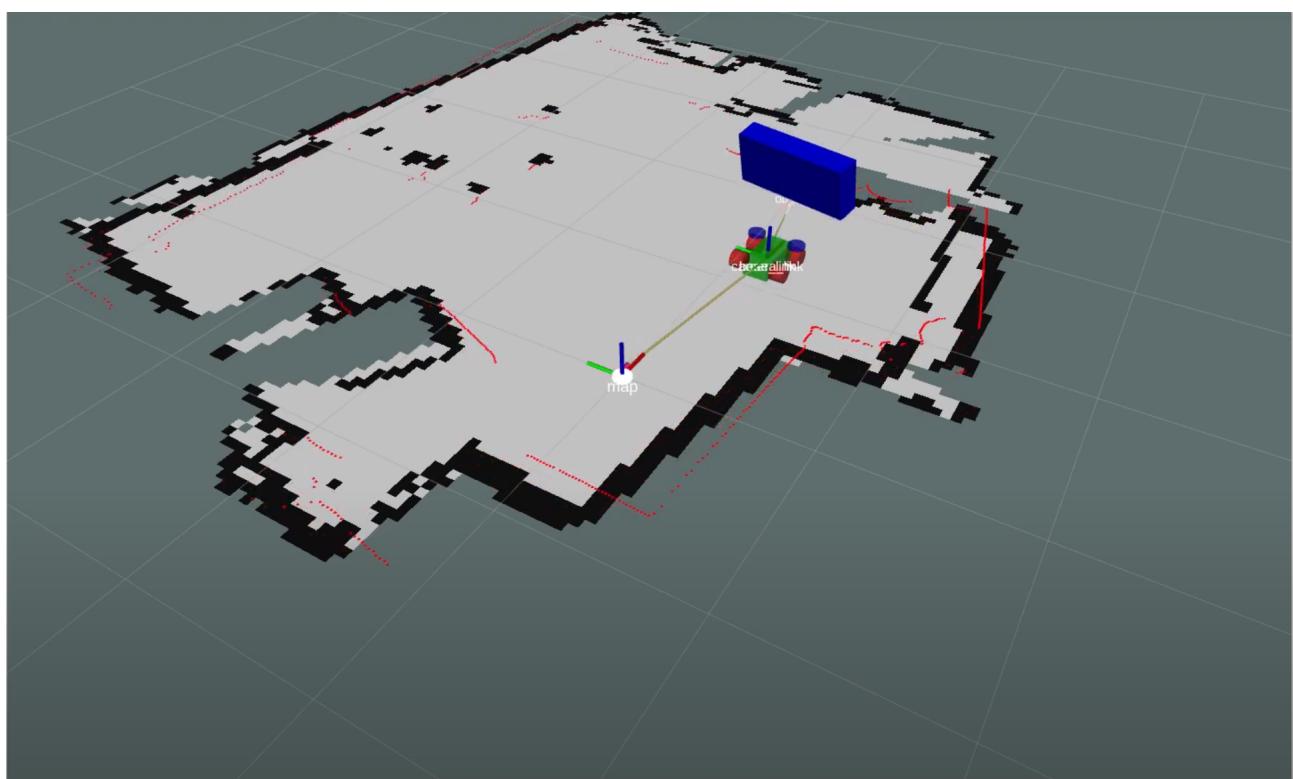


Abbildung 9: Torerkennung in Rviz

3.2.7 Funktion 7: Einen Ball autonom suchen und erkennen

Tabelle 9: Funktion 7

#	Komponentendetail	Erforderliche Arbeiten
T1	ROS Navigation	Erlaubt das Erkunden von unbekannten Karten. Dadurch kann der Fennec autonom fahren
T2	Intel Realsense	Farbbilder weiter an T3 übergeben
T3	opencv Paket	Der Ball wird erkannt und Pixelkoordinaten werden dann veröffentlicht.

T1: ROS Navigation

Siehe **Funktion 4: Hindernisse umfahren**

T2: Intel Realsense

Die Intel Realsense gibt, wie in anderen Fällen, Farbbilder und Tiefendaten weiter an ROS. In diesem Fall benötigen wir erstmal keine Tiefendaten. Die Farbbilder reichen aus, um die Pixelkoordinaten des Balles innerhalb einer Aufnahme auszumachen.

T3: opencv Paket

Wie bei der **Funktion 3 Gegen Ball fahren und schießen** werden die Pixelkoordinaten, die von der Intel Realsense übergeben werden verwertet und ausgegeben. Dabei wird die Bildgröße beachtet. Auch hier beträgt die Bildgröße 640x480 Pixel.

3.3 System Infrastruktur / ROS Multiple Machine Setup

Für den Betrieb der Anwendung benötigen wir neben dem Jetson Nano Mikrocontroller wie in Abbildung 6 zu sehen noch einen Laptop, welcher zum Visualisieren der erstellten Karten und Szenen benutzt wird. Außerdem können dort rechenintensive Aufgaben wie z.B. die Bildverarbeitung ausgelagert werden. Das ist nützlich, wenn der Jetson trotz seiner Grafik - Rechenleistung Verzögerungen bei der Berechnung in einzelnen Nodes hat.

Durch die verteilte Struktur von ROS können einzelne Pakete oder ganze Module auf verschiedener Hardware laufen und miteinander weiterhin kommunizieren.

Die einzigen Packages, welche zwingendermaßen auf dem Jetson laufen müssen sind der base_controller, realsense_ros und rplidar_ros, da sie alle in direktem Kontakt zu Hardware sind und die Schnittstelle zwischen ROS und der Hardware sind.

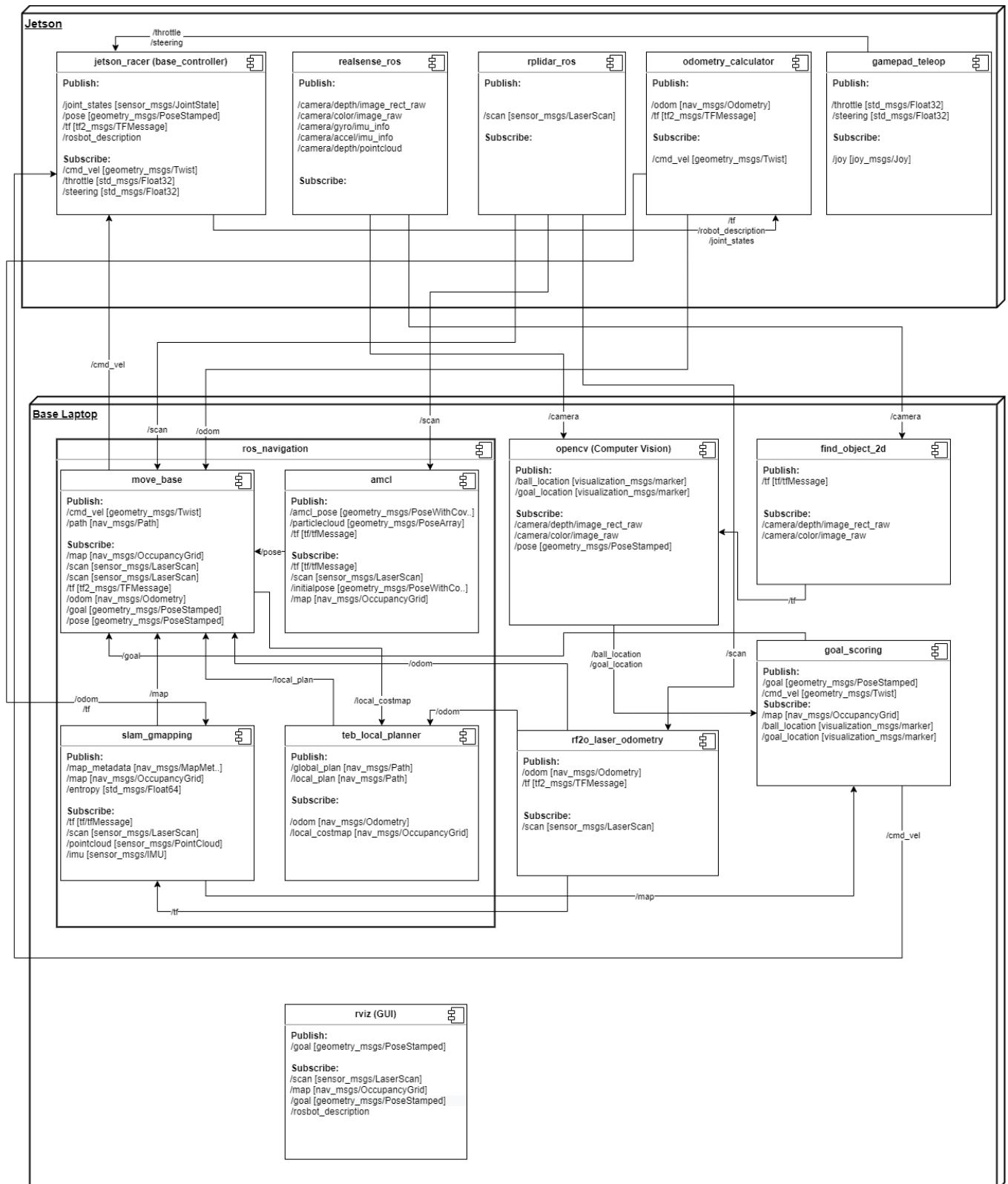


Abbildung 10: Systeminfrastruktur

4 Technische Spezifikation Konstruktion

Das Chassis ist locker nach dem "Fennec"-Rennwagen geformt und wurde in 4 verschiedenen Teilen im 3D-Druck gedruckt, die später zusammengeklebt wurden. Neben kosmetischen Zwecken dient es als Schutzhülle für den JetRacer und bietet Platz zum Montage der Laserscanner und Intel RealSense-Kamera.

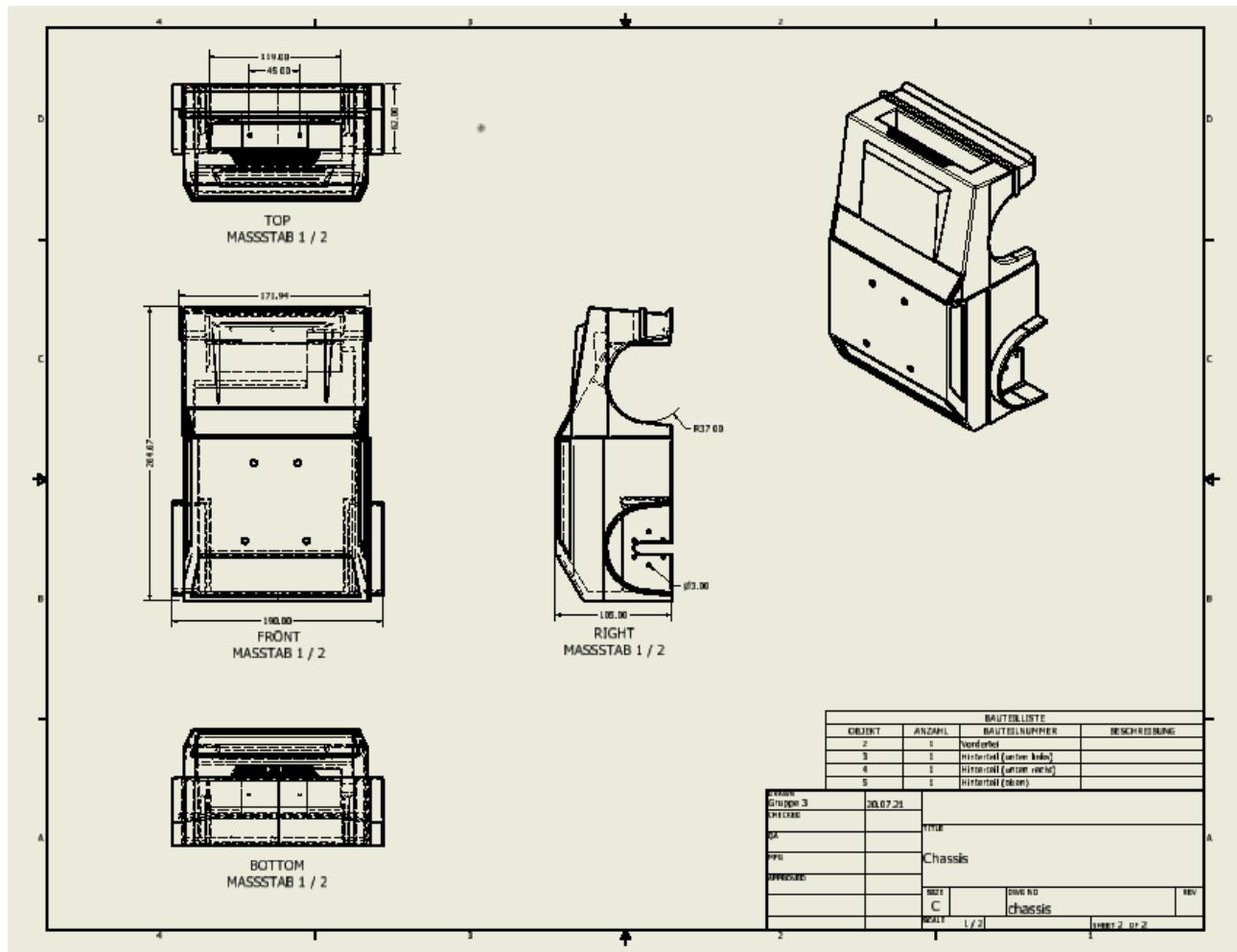


Abbildung 11: 2D-Zeichnung des Chassis