Technische Spezifikation

Fennec Racer

Autor:
Letzte Änderung:
Dateiname:

Version:

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Copyright

Version Historie

Version:	*Datum:*	*Verantwortlich*	*Änderung*
0.1	14.06.2021	Lukas Evers	Fachlicher Workflow, Überblick der Komponenten, Systeminfrastruktur/ ROS Multiple Machines Setup
0.2	15.06.2021	Umut Uzunoglu	Beschreibung der Implementierung, Erklärung wichtiger Begriffe
0.3			
0.4			
0.5			
1.0			
1.1			
1.2			

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

1.1 Erklärung wichtiger Begriffe

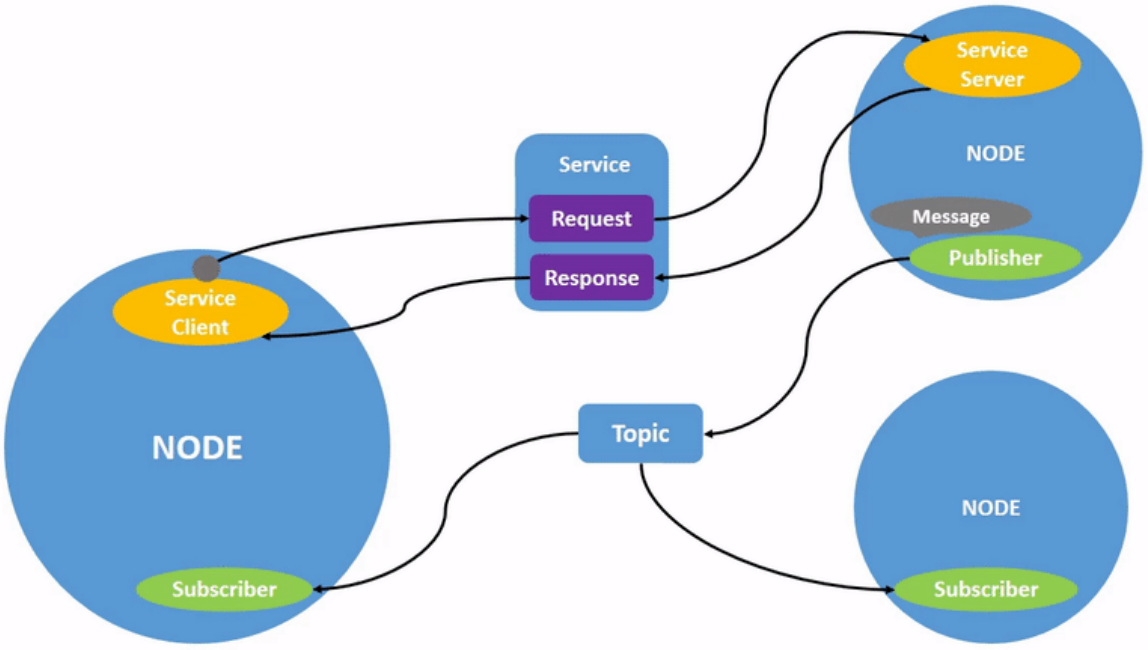

Abbildung 1: Nodes, Topics und Messages

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 gepublisht werden. Wenn Messages von Nodes und Topics aus verschiedenen Programmiersprachen gepublisht werden, dann muss darauf geachtet werden, dass diese Messages in einem ROS "neutralen" Typen gepublisht 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 gepublisht, 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

Hier soll der übergeordnete Überblick des gesamten Prozesses graphisch dargestellt werden. Dabei können zwei Ebenen unterschieden werden

2.1 Fachlicher Workflow

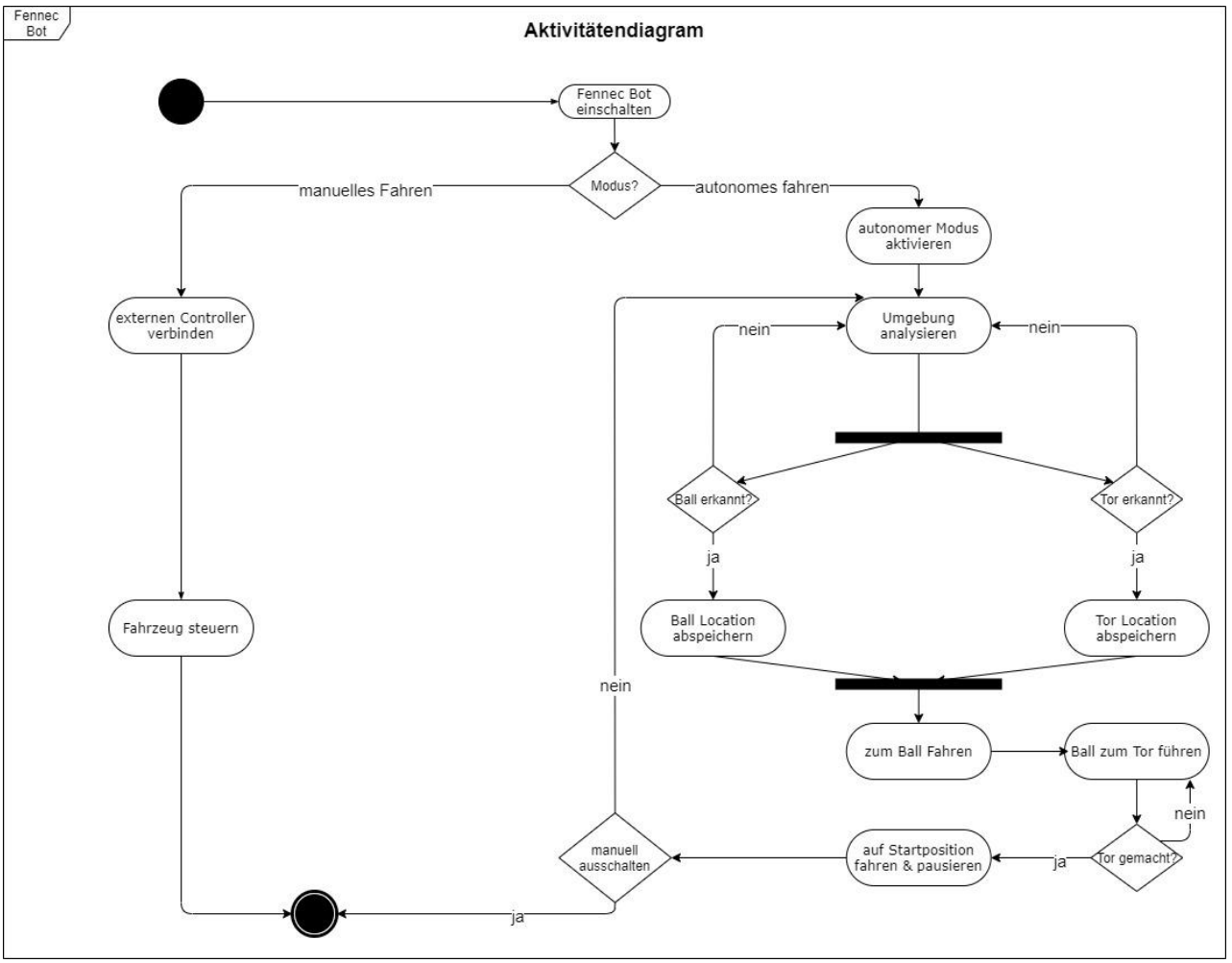

Abbildung 2: Fachlicher Workflow

3 Technische Spezifikation SW

3.1 Überblick Komponenten

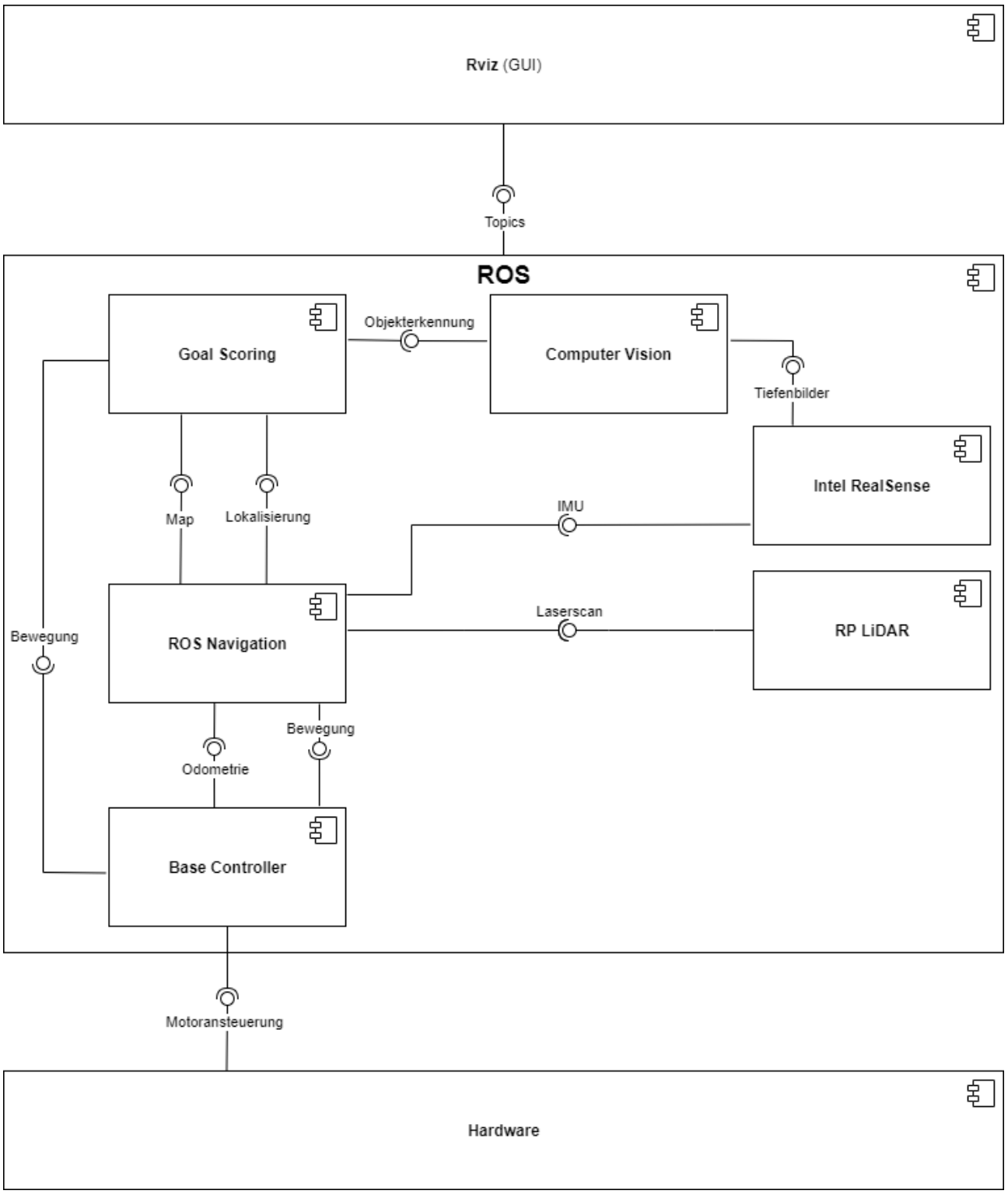

Abbildung 3: Komponentendiagramm

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 gepublisht.
T3	AMCL	Die gepublishte Karte vom Gmapping kann dann AMCL übergeben werden, wodurch der Fennec auf der Karte lokalisiert wird.
T4	ROS Navigation Stack	ROS Navigation Stack ist für die Pfadplanung und das Umgehen von Objekten verantwortlich

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 gepublisht (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)

Nach dem 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 gepublisht wird. Dabei wird die Karte immer wieder aktualisiert und gepublisht, 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 dem /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.

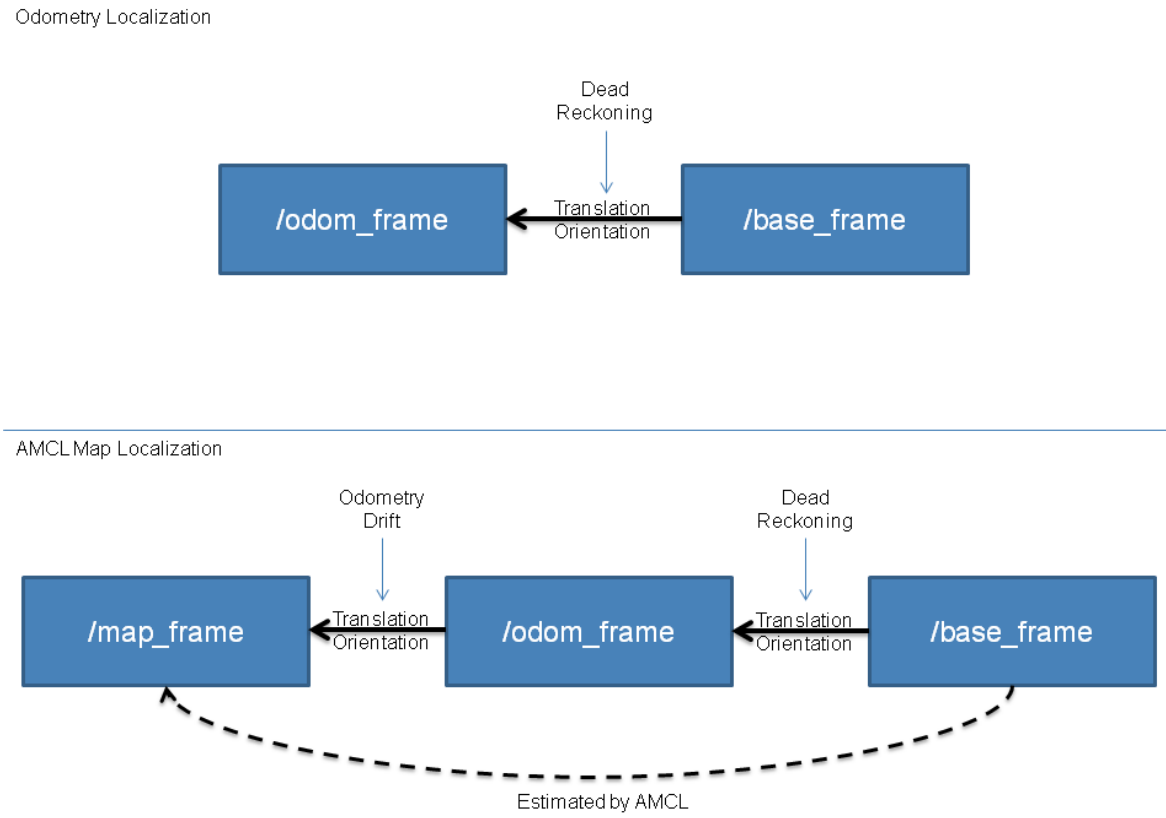

Abbildung 4: AMCL http://wiki.ros.org/amcl

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 7 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 gepublisht. Gleichzeitig werden aber auch Odometrie Messages in Form von “nav_msgs/Odometry”, die die Geschwindigkeit des Roboters neben der “tf”-Transformation publisht, gepublisht.

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.

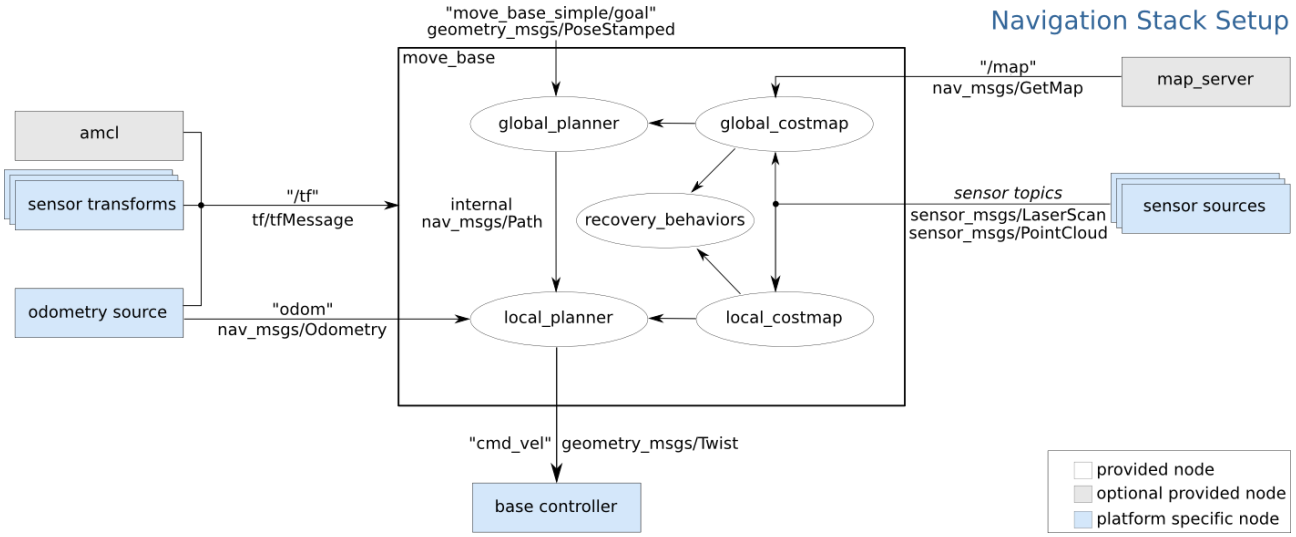

Abbildung 5: ROS Navigation navigation/Tutorials/RobotSetup - ROS Wiki

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 gepublisht. Diese Messages werden dann der Hardware übergeben und der Fennec setzt sich in Bewegung.

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.

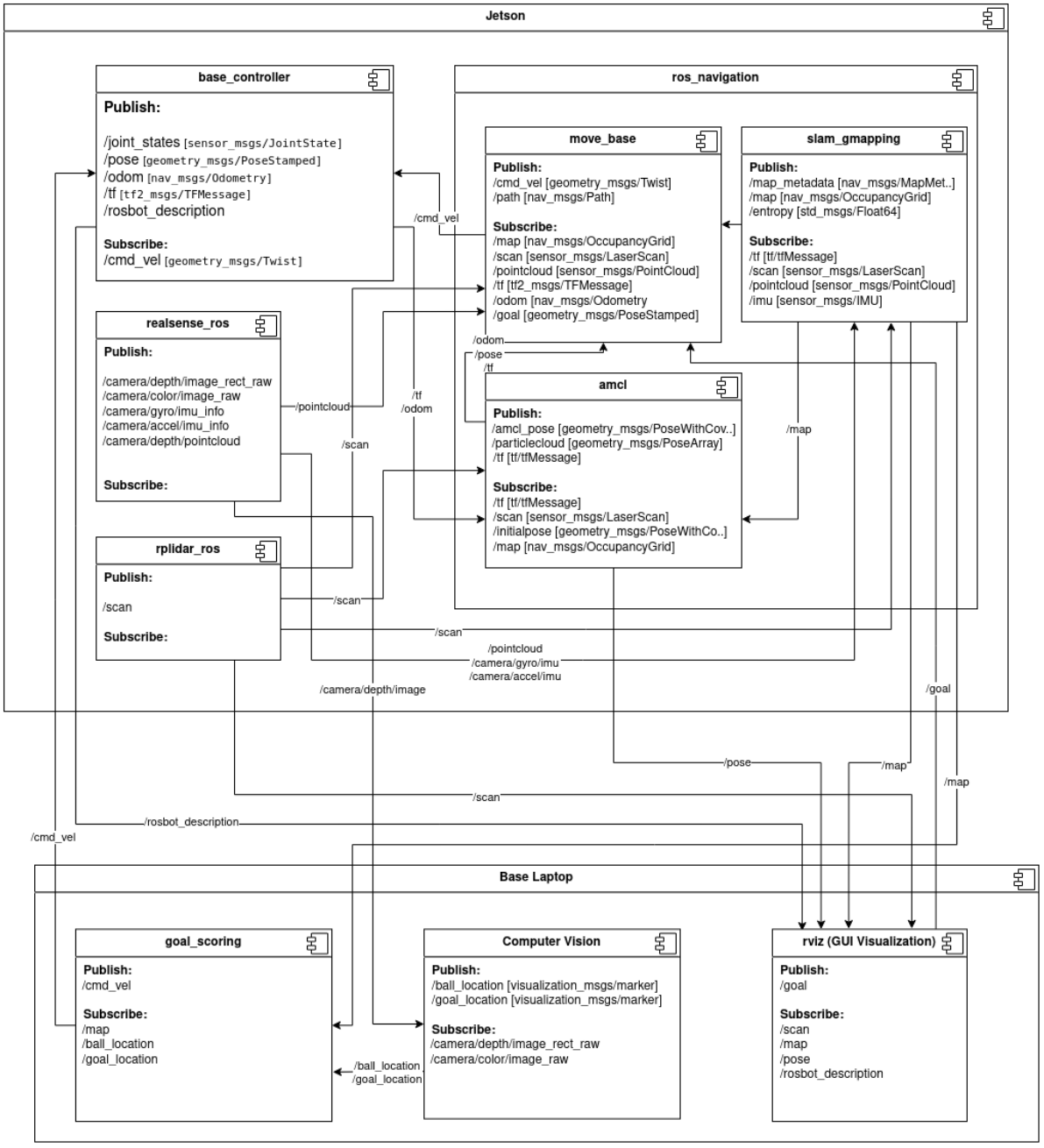

Abbildung 6: Systeminfrastruktur

4 Technische Spezifikation Konstruktion

Das Chassis ist einem "Fennec"-Rennwagen nachempfunden und wird später neben dem Laserscanner auch die Intel RealSense-Kamera aufnehmen können. In seinem aktuellen Zustand dient das 3D-Druckergebnis als grobe Form des Endprodukts, das für die weitere ROS-Programmierung dient.

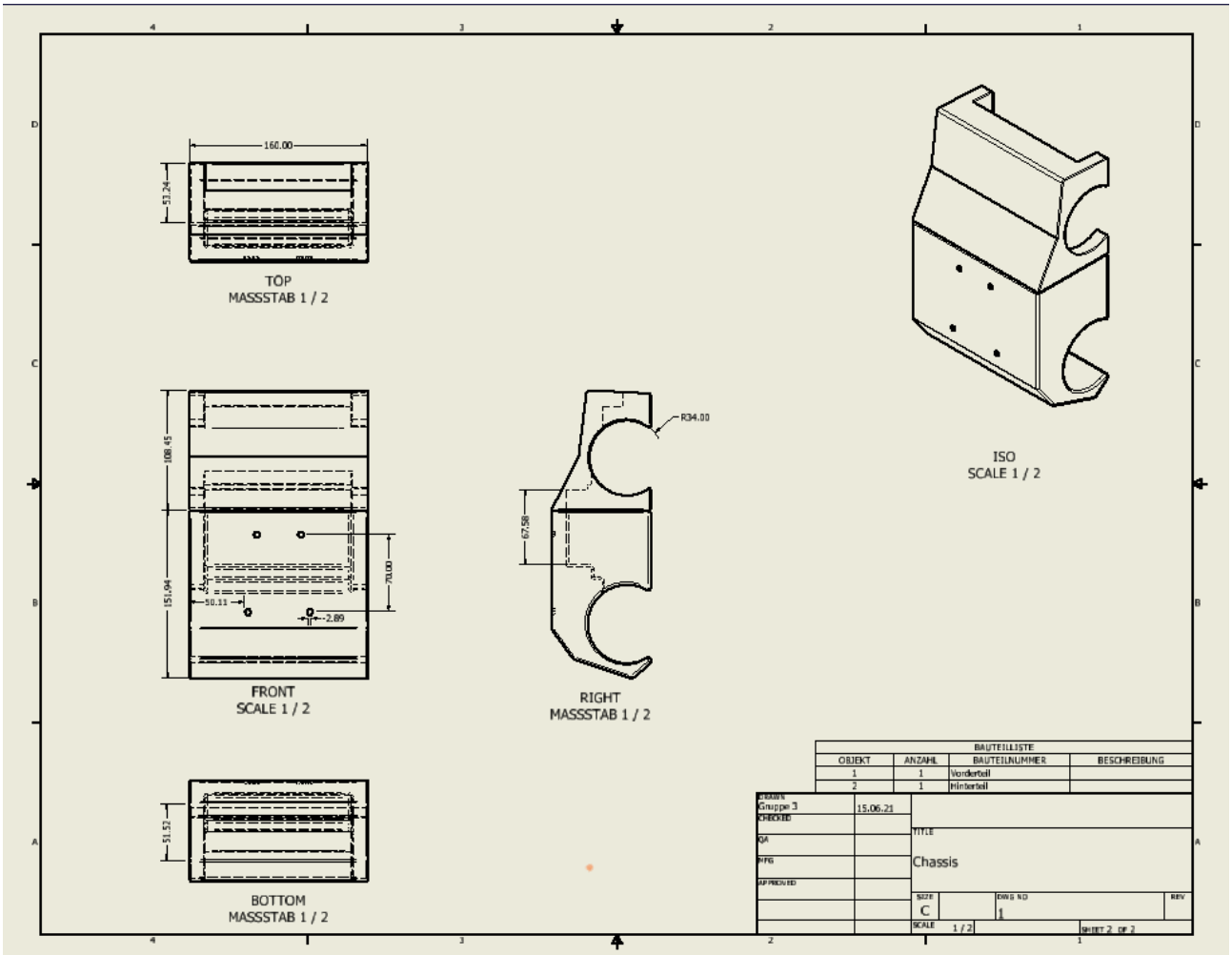

Abbildung 7: 2D-Zeichnung des Chassises