IROBOT-Projekt

Autonome Navigation, Trajektoriengenerierung und Verhaltenskoordination

Benutzerhandbuch – Oktober

Autor: Alvaro Fernández Garcia

Kontakt: fernandezgalvaro@uniovi.es

Datum: 9. Oktober 2019

1 Zusammenfassung des ROS-Workspace-Sets

Die ROS-Umgebung ist hauptsächlich durch Arbeitsbereiche oder *Arbeitsbereiche organisiert.* Diese enthalten den in Paketen organisierten Quellcode, weitere Informationen finden Sie im Anhang, der den Grundkonzepten von ROS gewidmet ist, und nach der Kompilierung die generierten ausführbaren Dateien und Bibliotheken.

Das Endergebnis der Arbeit war der Einfachheit halber die Erstellung von zwei Arbeitsbereichen: *ws* und *irobot_qt_ws.* Die erste davon umfasst hauptsächlich autonome Navigationsroutinen und die zweite, eher darauf ausgerichtet, mit Hilfe von QtCreator bearbeitet, kompiliert und ausgeführt werden zu können, solche zur Generierung von Trajektorien.

In diesem Abschnitt wird eine kurze Beschreibung der bemerkenswertesten Pakete, Knoten und Bibliotheken des Projekts gegeben.

1.1 Arbeitsbereich ws Der

Arbeitsbereich ws besteht aus folgenden Paketen:

1.1.1 action_server_package Dieses

Paket enthält die selbst erstellten Dienste, die während des Navigationsverhaltens aufgerufen werden.

1.1.2 aux_controllers Paket,

das orientierte Bibliotheken enthält, die hauptsächlich dem Senden von Nachrichten an den Roboter und dem Koppeln von Werkzeugen in der Simulation gewidmet sind.

1.1.3 flexbe_ap, flexbe_behavior_engine und generic_flexbe_states Diese

Pakete sind dafür verantwortlich, dass das FlexBe-Koordinierungstool funktioniert. Für den Download und die Installation können Sie die Webseite REF konsultieren.

1.1.4 irobot_move_base Dieses

Paket enthält die .yaml-Konfigurationsdateien, die die autonome Navigation des Roboters anpassen.

1.1.5 irobot_sm_behaviors Enthält

von FlexBE generierten Code, der aus der Erstellung von Zustandsmaschinen und selbst erstellten Zuständen in Python abgeleitet wurde.

1.1.6 my_rviz_tools

Dieses Paket enthält die Bibliotheken, die für unsere eigenen Tools in Rviz erstellt wurden. Kurz gesagt, *my_goal_tool* implementiert das Tool, mit dem Sie auf die Rviz-Karte klicken und diese Position im *my_2d_goal*- Thema veröffentlichen können, das von jedem Knoten gelesen werden kann. Die *repair_tool* -Klasse hat eine ähnliche Funktionalität, indem sie im rviz_defect- *Thema* veröffentlicht wird, für temporäre Funktionalität, die eine sehr einfache Simulation einer Reparatur ermöglicht.

1.1.7 slam_launchers Paket

enthält. starten, um gemeinsam und mit den geeigneten Parametern das 3D-Mapping und den SLAM-Algorithmus auszuführen.

1.1.8 spawn_package Paket,

das sich vervielfacht. Startdateien, die intern zum Aufrufen von Modellen in verschiedenen Gazebo-Welten verwendet werden.

1.1.9 summitxl_metapackage Paket,
das die Pakete enthält, die für die Verwendung von SummitXL im Gazebo-Simulator erforderlich sind. Zusätzliche Dateien wurden hinzugefügt, z. B. ein Modell mit dem RGBD-Sensor.

1.1.10 velodyne_simulator
Dieses Paket, erhältlich unter [1], enthält die für die 3D-Sensorsimulation notwendigen Dateien. Änderungen am Originalcode befinden sich in den Ordnern velodyne_description/urdf und velodyne_description/launch. Der erste enthält die Parameter, mit denen der 3D-Sensor modifiziert werden kann (Anzahl der Laser, Reichweite, Blende usw.). In diesem Ordner finden Sie Simulationen handelsüblicher Sensoren wie MRS1000, MRS6000, IntelRealSense_D435 etc.

Wenn Sie FILE als Referenz verwenden, finden Sie die Parameter, die Sie in der urdf ändern möchten, in Zeile 4. Darin können Sie den Namen des *Themas* ändern , in dem Sie veröffentlichen, die Anzahl der Laser, die Frequenz, das Rauschen und das Minimum Reichweite und Maximum.

```
1 <?xml-version="1.0"?>
2 <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="VLP-16">
3     <xacro:property name="M_PI" value="3.1415926535897931" />
4     <xacro:makro name="VLP-16" params="*origin parent:=base_link name:=velodyne topic:=/
      velodyne_points hz:=10 lasers:=16 samples:=1875 Collision_range:=0.3 min_range:=0.9 max_range:=130.0 noise:=0.008
      min_angle:=- ${M_PI} max_angle:=${M_PI} gpu:=false">
5
```

Im unteren Teil der Datei können Sie die maximalen und minimalen Winkel ändern, sowohl horizontal (Zeile 110 und 111) als auch vertikal (Zeile 116 und 117).

```
100         <xacro:unless value="${gpu}">
101             <sensor type="ray" name="${name}-VLP16">
102                 <einstellen>0 0 0 0 0 0</einstellen>
103                 <visualize>false</visualize>
104                 <update_rate>${hz}</update_rate>
105                 <Strahl>
106                     <scannen>
107                         <horizontal>
108                             <Beispiele>${Beispiele}</Beispiele>
109                             <Auflösung>1</Auflösung>
110                             <min_angle>${min_angle}</min_angle> <max_angle>$
111                             {max_angle}</max_angle>
112                         </horizontal>
113                         <vertikal>
114                             <Beispiele>${Laser}</Beispiele>
115                             <Auflösung>1</Auflösung>
116                             <min_angle>-${15.0*M_PI/180.0}</min_angle> <max_angle> ${15.0*M_PI/
117                             180.0}</max_angle>
118                         </vertikal>
119                     </scan>
120                     <Bereich>
121                         <min>${collision_range}</min>
```

```
                        <max>${max_range+1}</max>
                        <Auflösung>0.001</Auflösung>
                    </range>
                    <Lärm>
                        <type>Gaußscher</type>
                        <Mittelwert>0,0</Mittelwert>
                        <stddev>0.0</stddev>
                    </Rauschen>
                </ray>
<plugin name="gazebo_ros_laser_controller"
    filename="libgazebo_ros_velodyne_laser.so"> <topicName>${topic}</
                    topicName>
                    <frameName>${name}</frameName>
                    <min_range>${min_range}</min_range> <max_range>$
                    {max_range}</max_range> <gaussianNoise>${noise}</
                    gaussianNoise>
                </Plugin>
            </sensor>
        </xacro:es sei denn>
```

1.2 Arbeitsbereich irobot_qt_ws *1.2.1*

action_server_package_qt Paket, das die
Dienste und Server für die Roboterbewegungsaktionen und die *Rahmenerstellung enthält.*

1.2.2 gazebo_sim_movement Paket,
das verwendet wird, um Trajektorien in Gazebo zu simulieren, ohne kinetische Nachrichten zu senden.

1.2.3 irobot_fcpp_pckg
Enthält Bibliotheken für Trapezzerlegung, Trajektoriengenerierung und andere Hilfsfunktionen.

1.2.4 irobot_fcpp_simul
Paket, das die Dienste und Server der Trapezzerlegungsaktionen, Trajektorienberechnung und den Hauptdienst des Automaten enthält.

1.2.5 irobot_inspection_pckg
Paket, das die Dienste und Server der Inspektionsaktionen enthält.

2 Autonome Navigationsumgebung

In diesem Abschnitt werden die Schritte erläutert, die zum Implementieren des autonomen Navigationsverhaltens erforderlich sind.

2.1 Konfiguration der autonomen Navigation

Wie bereits erwähnt, befinden sich die Konfigurationsdateien der autonomen Navigation im *Paket irobot_move_base.*

Die Parameter für den TEB-Scheduler befinden sich in der *Datei base_local_params.yaml.* Diese Werte werden vom installierten Paket *teb_local_planner ausgelesen,* die Erklärung der wichtigsten findet sich im ROS-Wiki [2]. Die Datei für das MPO700 basiert größtenteils auf dieser Datei, die in einem offiziellen Neobotix-Repository vorhanden ist [3].

Die restlichen Dateien enthalten die Parameter, die sich auf die lokale, globale und statische Kostenkarte beziehen.

Zur Navigation wird die ROS- *Navigationsbibliothek* [4] verwendet . Es ist das umfangreichste Paket zur Steuerung und Navigation mobiler Roboter. Es wird dafür verantwortlich sein, die von der Karte und dem Trajektorienplaner erhaltenen Informationen zu koordinieren, um Befehle an die mobile Basis zu senden. Die Erklärung kann in die Merkmale der Kostenkarten und die Konfiguration des Schedulers selbst unterteilt werden.

Kostenkarten sind von der Bibliothek *costmap_2d* [5] bereitgestellte Strukturen , die es ermöglichen, mit belegten, freien oder unbekannten Zonen zu arbeiten.

Die Kostenkarten in diesem Paket können in verschiedene Ebenen organisiert werden, wobei die gebräuchlichsten die statische Kartenebene, die Hindernisebene und die Inflationsebene sind. Eine übliche Lösung besteht darin, eine statische Karte für die globale Karte und eine Hindernisschicht zu verwenden, die dynamisch aus den Sensoren für die lokale Karte erstellt wird.

Dank *Octomap* wird die in unserer Arbeit erhaltene Karte bereits ein dynamisches Verhalten aufweisen, das dem Roboter eine sichere Umgebung für die Navigation garantiert. Auf diese Weise werden die lokale und die globale Karte aus den gleichen Arten von Layern bestehen: einem *Staticmap[6]-Layer* und darüber hinaus einem *Inflations[7]-Layer.*

Die statische Schicht ist einfach das Lesen der projizierten 3D-Karte, die so konfiguriert ist, dass sie auf Aktualisierungen wartet.

Die Inflationsebene wiederum ermöglicht es Ihnen, die Navigationssicherheit zu erhöhen, indem Sie der Karte Gefahrenwerte um erkannte Hindernisse geben. In der Umsetzung wird ein Inflationsradius von 0,75 Metern festgelegt. Dieser Wert kann in der letzten Zeile der Datei irobot_move_base/yaml/costmap_common_params geändert werden.

Der Hauptunterschied zwischen der lokalen Karte und der globalen Karte wird in ihrer Größe liegen. Die globale Karte erhält in diesem Fall eine maximale Größe von 100 x 100 Metern (änderbarer Wert in irobot_move_base/yaml/global_costmap) und wird beim Erstellen der 3D-Karte vervollständigt. Es wird dazu dienen, die vollständige anfängliche Flugbahn zu planen. Die lokale Karte wird jedoch ein 6x6-Meter-Fenster sein (Wert modifizierbar in irobot_move_base/yaml/local_costmap), auf der Karte, die sich zusammen mit dem Roboter bewegt, Abbildung 1. Auf dieser Karte werden die verschiedenen Zwischenposen in einer Form eingeführt respektieren Sie die Gesamtflugbahn so weit wie möglich und vermeiden Sie Kollisionen mit Hindernissen.

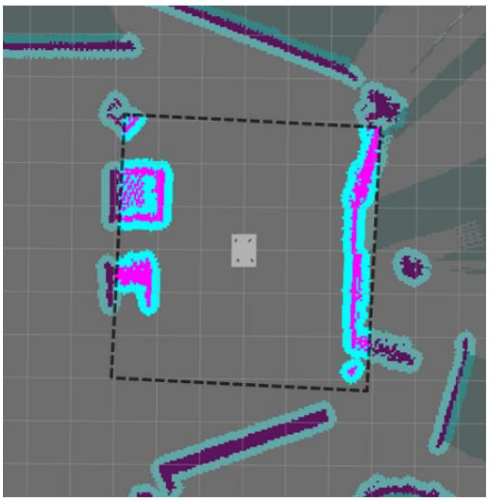

Abbildung 1: Darstellung der lokalen Kostenkarte. Die globale Kostenkarte wird weiter ausgebreitet und ist im Bild mit geringerer Deckkraft zu sehen.

2.2 Aktionen und Server

Der FlexBE-Code ist im Gegensatz zum C++ der selbst entwickelten Pakete in Python geschrieben. Um diesen Code zu nutzen, wird die Kommunikation über Aktionen verwendet, die in Anhang I erläutert wird. Auf diese Weise werden die Zustände der Maschine in Python geschrieben und enthalten die Clients, die die Aufrufe der Aktionen durchführen. Der C++-Code implementiert Server, die ständig laufen und die die der Aktion zugeordnete Funktion ausführen, wenn der Client sie anfordert.

Die Server, die speziell für die Simulation der autonomen Navigation implementierte Aktionen ausführen, sind in Tabelle 1 zu sehen. Die Bewegungen des Roboters werden mit ausgeführt
Methoden der *RobotController-Klasse.*

Server - .Aktion	**Beschreibung**
baserotation_server.cpp **–** BaseRotation.action	Ausführen einer Drehung der Basis um eine bestimmte Gradzahl.
moverobotcoord_server.cpp – MoveRobotCoord.action	Geradlinige Bewegung des Roboters zu einer Referenzposition in einem bestimmten *Rahmen* .
completerectanglecoverage_server.cpp – CompleteRectangleCoverage.action	Planung und Ausführung der komplexen Prüfstrecke auf Basis der Blech- und Roboterparameter.

Tabelle 1: Satz von Aktionen, die im Automaten verwendet werden

Andere Operationen werden viel schneller ausgeführt, weshalb sie über Dienste statt über Aktionen realisiert werden, Tabelle 2. Dies ist der Fall bei der Veröffentlichung eines Hilfskoordinatensystems und der Abfrage der Lage der Blätter.

Server - .srv	**Beschreibung**
inspektionsrahmen_server.cpp – SetFrame.srv	Veröffentlichung eines tf -*Rahmens in* Entfernung und Orientierung eines Bezugssystems. Der Dienst ist Sie ist als Methode einer Klasse implementiert, bekommt also Speicher in Bezug auf ein paar Variablen.
chapastorage_server.cpp – ChapaStorage.srv	Abhängig von der Anfrage gibt die Anzahl zurück gespeicherte Tafeln oder die Abmessungen und Eckkoordinaten einer ausgewählten Tafel.

Tabelle 2: Satz von Diensten, die im Automaten verwendet werden

2.3 Implementierung der Zustandsmaschine und Parameter Nachdem die Konzepte erklärt wurden, wird die implementierte Maschine detailliert beschrieben. Zunächst ist anzumerken, dass die Server der Aktionen und Dienste parallel laufen müssen, wenn die Zustandsmaschine gestartet wird.

Neben diesen selbst entwickelten Programmen müssen auch die Knoten der Pakete laufen, die für SLAM, 3D-Mapping und Navigation zuständig sind, sowie natürlich diejenigen, die die Simulation des Roboters und der Sensoren bereitstellen.

Schließlich können vor der Ausführung der Maschine die Variablen angepasst werden, die ihr Verhalten konfigurieren, Tabelle 3.

Name	**Wert**	**Beschreibung**
topic_rviz_goal	= „/my_2d_goal“	<cte.> *Thema ,* aus dem objektive Positionen stammen.
Inspektionsrahmen	= „inspection_frame“	<Fortsetzung> Hilfskoordinatensystem für die Inspektion.
Grad	= 180,0	<Konstante> Gradzahl für den ersten Erkundung.
dist_laserbot	= 1	<Fortsetzung> Abstand zwischen der Basis des Roboters und dem Prüfstrahl.
width_laser	= 2	<konstant> Breite des Prüfstrahls.
step_path	= 4	<Fortsetzung> Maximaler Abstand zwischen zwei Zwischenpositionen einer Blechbahn.
vel_pfad	= 0,5	<Konstante> Geschwindigkeit für die obige Fahrt des Blattes
rviz_goal_posestamped = [ ]		<vble.> Hilfsvariable, die a speichert Nachricht vom Typ *PoseStamped.*

rviz_goal_pose2d = [ ]	<vble.> Hilfsvariable, die eine Nachricht vom Typ *Pose2D speichert.*

Tabelle 3: Konstanten und Variablen der Maschine in FlexBE

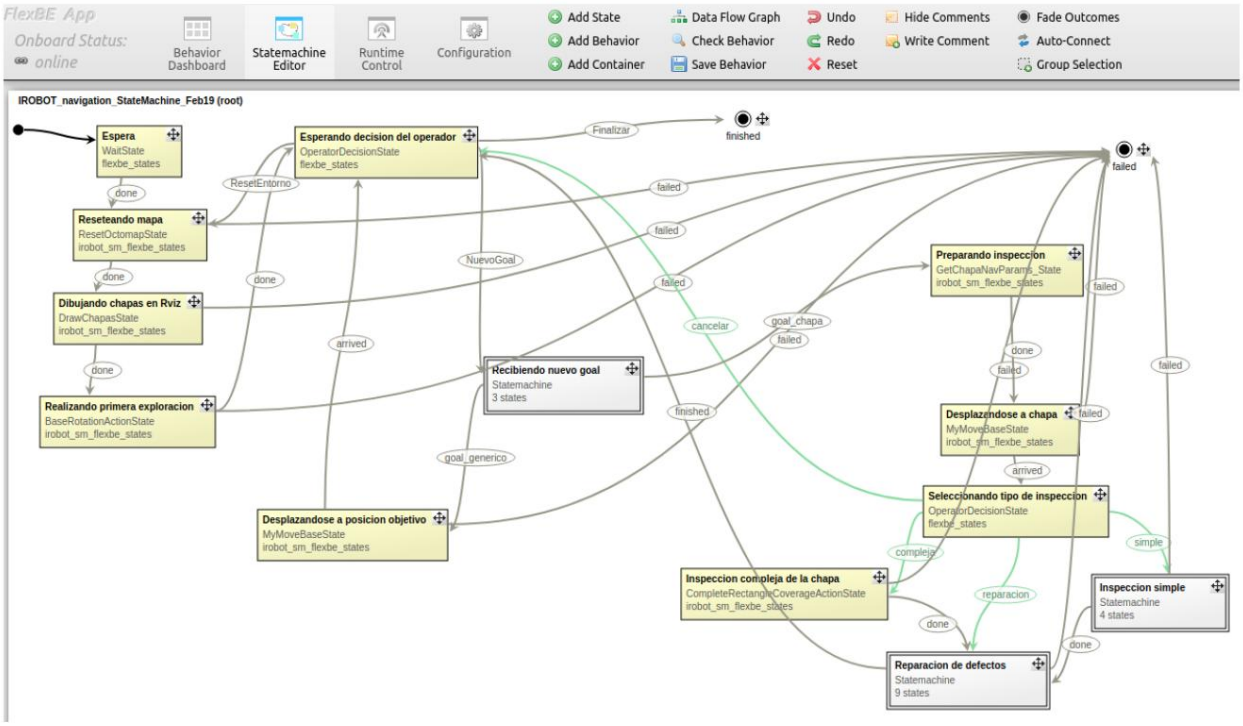

Abbildung 2: Übersicht über die Zustandsmaschine in FlexBE

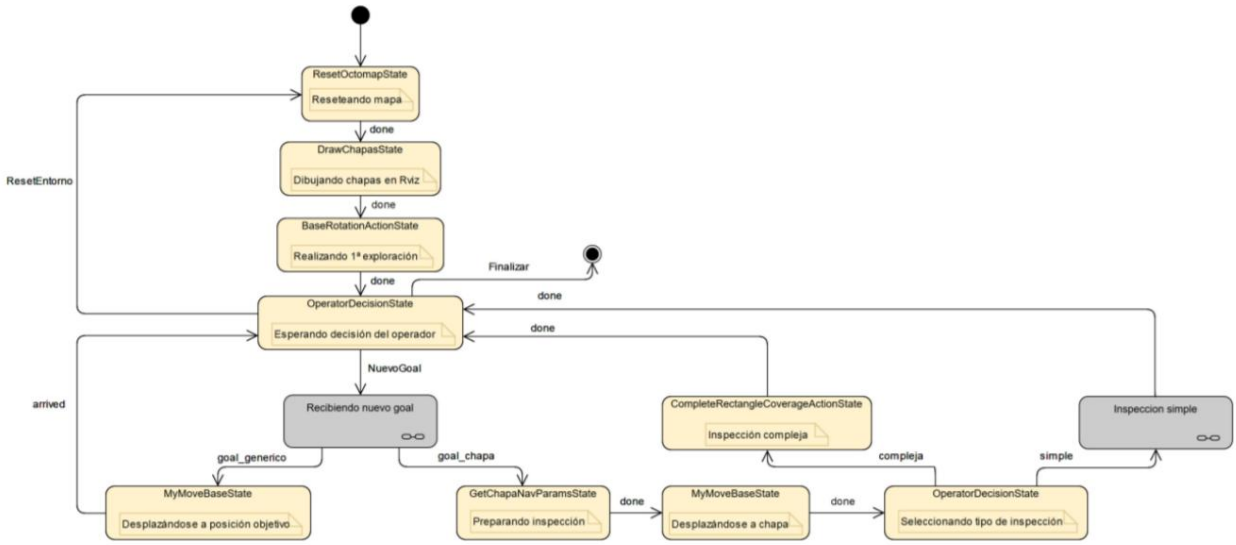

Abbildung 3: Überblick über eine Darstellung des Zustandsautomaten.

Fokussierung auf die im Automaten implementierte Logik, Abbildung 2 und Abbildung 3 Abbildung 3: Gesamtansicht einer Darstellung des Zustandsautomaten.

, Der erste zu aktivierende Zustand ist die Neuinitialisierung des *Octomap.* Für die Löschung aller Knoten, die die Octrees-Struktur bilden, wird *der Reset* -Dienst des Knotens *octomap_server* aufgerufen.

Der folgende Status wird verwendet, um die Position der Blätter in *Rviz darzustellen.* Dazu wird auf den Server, der die Blechkoordinaten speichert, zugegriffen und eine Nachricht vom Typ *geometry_msgs::PolygonStamped* [8] veröffentlicht, die *jeweils das Topic sheet_idX* basierend auf der Kennung des gezeichneten Blechs erzeugt.

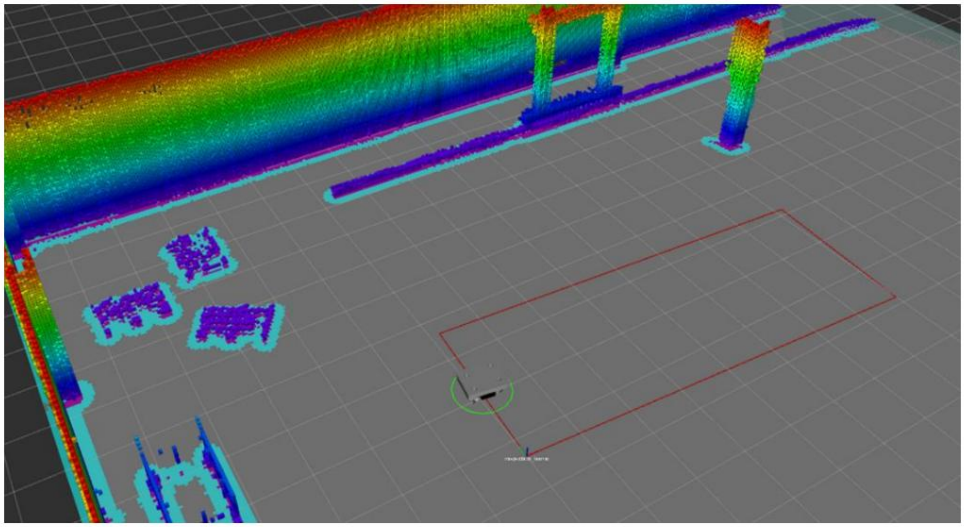

Abbildung 4: Bild mit Roboter im gezeichneten Blatt

Im nächsten Schritt dreht sich die Basis des Roboters um sich selbst. Auf diese Weise erhält der Roboter vor der Initialisierung des Verhaltens eine vollständige Karte seiner nächsten Umgebung. Um die Drehung durchzuführen, wird die *Serveraktion baserotation_server aufgerufen ,* in der die Gradzahl durch eine beim Erstellen des Automaten konfigurierte Variable definiert wird.
Die Aktion verwendet die *turn_rel*- Methode der RobotController-Klasse .

Sobald diese Anfangsphasen abgeschlossen sind, wird der *OperatorDecisionState erreicht,* ein von *FlexBE* entwickelter Raum , der es dem Benutzer ermöglicht, den nächsten Schritt auszuwählen. Es werden drei mögliche Ausgänge eingerichtet: *Finish, ResetEntorno* und *NuevoGoal.* Der erste erlaubt dem Automaten zu enden und der zweite in den Anfangszustand zurückzukehren, falls ein Fehler auf der Karte aufgetreten ist. Schließlich ermöglicht Ihnen *NuevoGoal* den Zugriff auf die Status im Zusammenhang mit dem Senden von Zielpositionen.

Dieser letzte Ausgang führt zum Superstate *Receiving new goal ,* der sich intern aus drei Zuständen zusammensetzt, die nacheinander ausgeführt werden, Bild 5. Der erste ist ein *SubscriberState* aus der *FlexBE*- Bibliothek selbst, der auf die Veröffentlichung einer Nachricht in einem bestimmten *Thema* wartet . Der Zustand ist so konfiguriert, dass er das *Thema* liest, das das Positionsauswahltool im Viewer veröffentlicht, und eine Ausgabevariable mit der angeklickten Position hat. Der nächste Zustand, *PoseStamped_Pose2D_ConversionState,* wird verwendet, um den Nachrichtentyp von *PoseStamped* zu *Pose2D zu ändern.* Dadurch können Sie mit dem Status fortfahren, der auswertet, ob sich die ausgewählte Position innerhalb oder außerhalb eines Blatts befindet. Die Überprüfung erfolgt durch Zugriff auf die im *Dienst chapastorage_server gespeicherten Koordinaten.* Der Zustand hat zwei Ausgänge, die er mit dem Superzustand teilt: *goal_generic,* falls sich die Position nicht innerhalb einer Stahlplatte befindet, und *goal_chapa ,* falls dies der Fall ist.

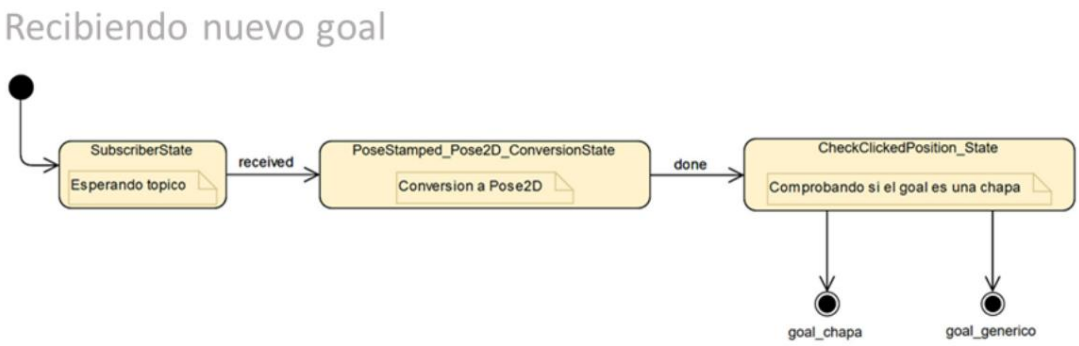

Abbildung 5: Innerhalb des Superzustands „Neues Ziel erhalten“

Wenn eine beliebige Position ausgewählt wurde, bewegt sich der Automat in den Zustand *Moving to target position.* Dies wird von *MyMoveBaseState implementiert,* das eine leichte Variation von *MoveBaseState* ist, das bereits in der Bibliothek enthalten ist. Die Hauptänderung bezieht sich auf den standardmäßig mitgelieferten *Referenzrahmen .* Der Automat bleibt in diesem Zustand, bis der Roboter die gewünschte Position erreicht und die *angekommene* Ausgabe aktiviert wird, wobei er in den Zustand zurückkehrt, der durch die Entscheidung des Bedieners gesteuert wird.

Befindet sich die angeklickte Pose innerhalb eines Blattes, wird auf den Zustand zugegriffen, der die Inspektion vorbereitet. Dies nimmt als Eingabe die *ID* der Platte, die den vorherigen Superzustand bereitstellt. Mit diesen Informationen kann *GetChapaNavParamsState* die Koordinaten und Abmessungen der Stahlbleche vom Dienst erhalten. Damit können Sie die Position des *Inspection_Frame* aus dem *SetFrame* -Dienst variieren, der als Methode der *FrameService-Klasse ausgeführt wird.* Der Staat kann auch die Zwischenstellung zwischen den Ecken *A* und *B* des Bogens auswählen. Diese Pose wird an einen neuen Zustand gesendet, der das *move_base* -Paket verwendet, damit die Verschiebung stattfinden kann.

Sobald sich der Roboter korrekt in einer Anfangsposition für die Inspektion befindet, wird ein neuer Zustand erreicht, in dem die möglichen Ausgaben darin bestehen, die Inspektion abzubrechen, eine komplexe Inspektion und eine einfache Inspektion des Bogens. Für den Fall, dass keine Einsichtnahme gewünscht wird , kehrt die Stornomöglichkeit wieder in den Status *„Warten auf Entscheidung des Betreibers“* zurück.

Die komplexe Prüfung wird in einem Zustand durchgeführt, der die Abmessungen des zu prüfenden Blechs erhält. Mit diesen Parametern und denen, die die Konfiguration des Roboters ausmachen, *kann die Aktion CompleteRectangleCoverage aufgerufen werden.* Diese Aktion umfasst das Planen der verschiedenen zu erreichenden Positionen und das Senden von Befehlen an den Roboter, um sie auszuführen. Sobald die Inspektion beendet ist, kehrt es zu dem Zustand zurück, in dem über die nächste Operation entschieden wird.

Falls eine einfache Inspektion ausgewählt wird, wird auf den Container Einfache Inspektion zugegriffen , der aus vier aufeinanderfolgenden Zuständen besteht, Abbildung 6. Der erste ermöglicht es, die drei erforderlichen Posen für diese Inspektion zu erhalten: eine Pose in einem bestimmten Abstand von der Platte Um es abzuschließen, eine Rückkehr zu einer Position zwischen den Koordinaten *A* und *B* und schließlich zu einer Position zwischen den Koordinaten *C* und *D ,* wodurch die Inspektion beendet wird, wie in **Error! Referenzquelle nicht gefunden.** Die folgenden Zustände sind für das Senden einfacher Befehle an den Roboter über die *Aktion „MoveRobotCoord“ verantwortlich.* Dazu erhält die Aktion als Ziele den Vektor, der von einer *Move* -Funktion der *RobotController* -Klasse ausgeführt wird. Wenn diese Inspektion endet, wird der Entscheidungsstatus des Bedieners zurückgegeben.

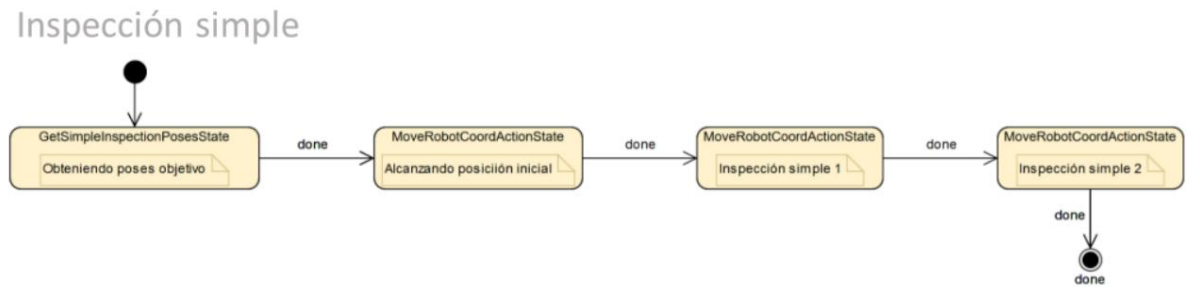

Abbildung 6: Innenansicht des Superstates „Einfache Inspektion“

Sobald die Erklärung abgeschlossen ist, ist es erwähnenswert, dass die überwiegende Mehrheit der Zustände einen *fehlgeschlagenen* Ausgang hat , der zum Ende des Automaten mit demselben Namen führt und der aktiviert wird, wenn irgendein Element im Zustand einen Fehler auslöst. Es wurde in der allgemeinen Erläuterung der Übersichtlichkeit halber weggelassen.

Schließlich sehen Sie in Abbildung 7, Abbildung 8 und Abbildung 9 ein Diagramm zum Senden von Daten durch die Zustandsmaschine.

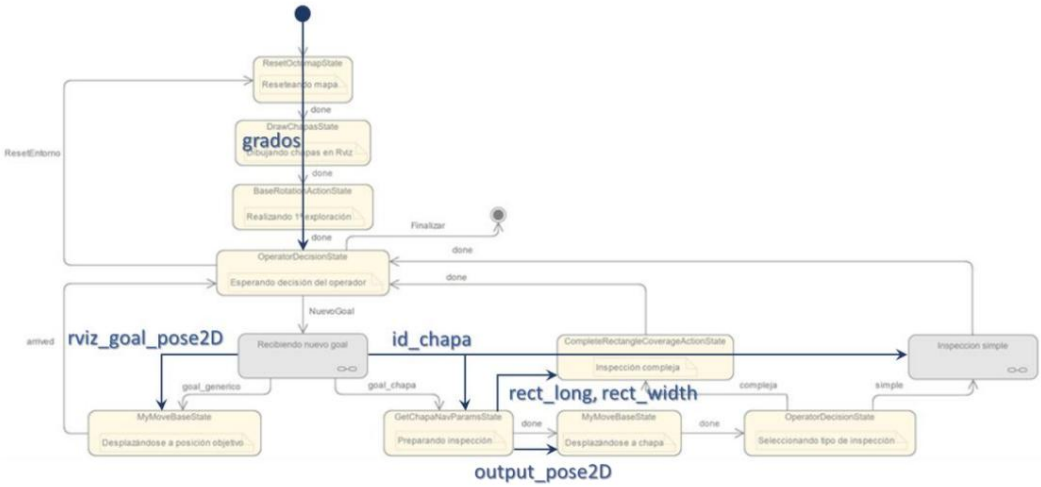

Abbildung 7: Allgemeiner Datenfluss in der SPS

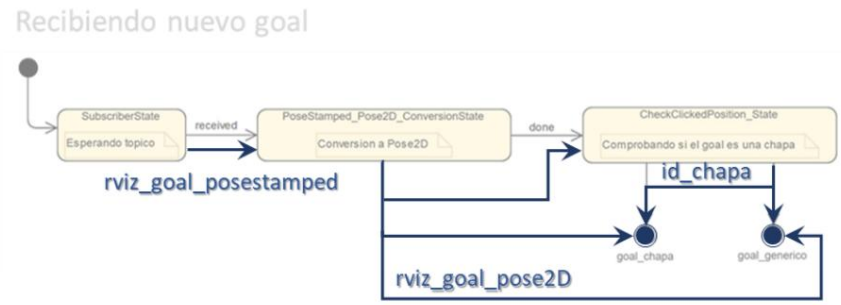

Abbildung 8: Datenfluss innerhalb des Superzustands „Neues Ziel erhalten“.

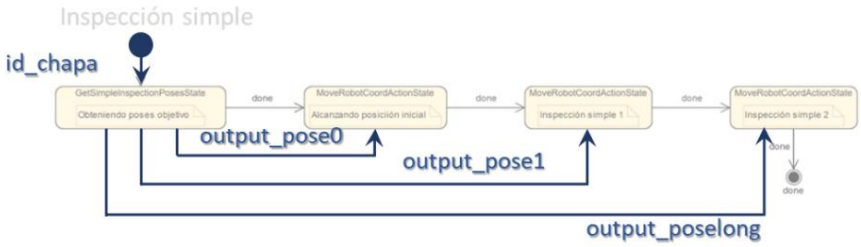

Abbildung 9: Datenfluss innerhalb des Superstates „Simple Inspection“.

2.4 Ausführung Um

die Umgebung und alle Server zu starten, genügt es, sie in einem Terminal auszuführen

roslaunch spawn_package mpo700_complete_slam_*

Wobei * je nach gewünschter Umgebung epilab, example oder factory sein kann.

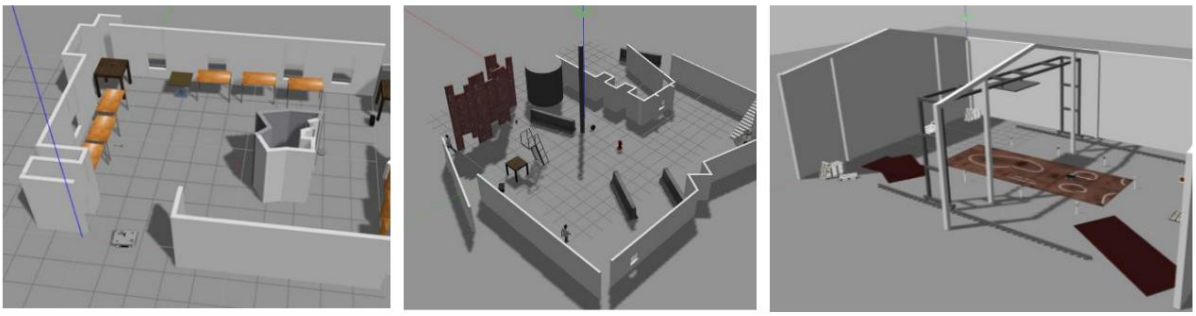

Abbildung 10:

Um den Zustandsautomaten auszuführen, muss die FlexBE-App ausgeführt werden.

roslaunch flexbe_app flexbe_full

Und wählen Sie das *Verhalten* namens IROBOT_navigation_StateMachine_Feb19.S

2.5 Struktur in ROS

Als Abschluss des Kapitels und als allgemeine Zusammenfassung ist Abbildung 11 enthalten, in der Sie die wichtigsten Knoten und *Themen* des Projekts sehen können, wobei einige dieser Elemente zur besseren Erläuterung weggelassen wurden.

An erster Stelle sind rechts die Knoten zu sehen , die die Verbindungen des Roboters zu *tf* kommunizieren. In der Abbildung sehen Sie auch die Knoten, die auf Gazebo für die Kopplung der Sensoren wirken, und wie die von ihnen generierten *Themen* zu den SLAM- und 3D-Mapping-Algorithmen führen. Unten sehen Sie auch den *Movebase* -Knoten mit seinen wichtigsten *Themen* und verbunden mit der Karte, den Koordinatensystemen und dem *Verhalten.* Auch das *Rviz*- Tool und die Aktionsserver sind mit dem Automaten verwandt .

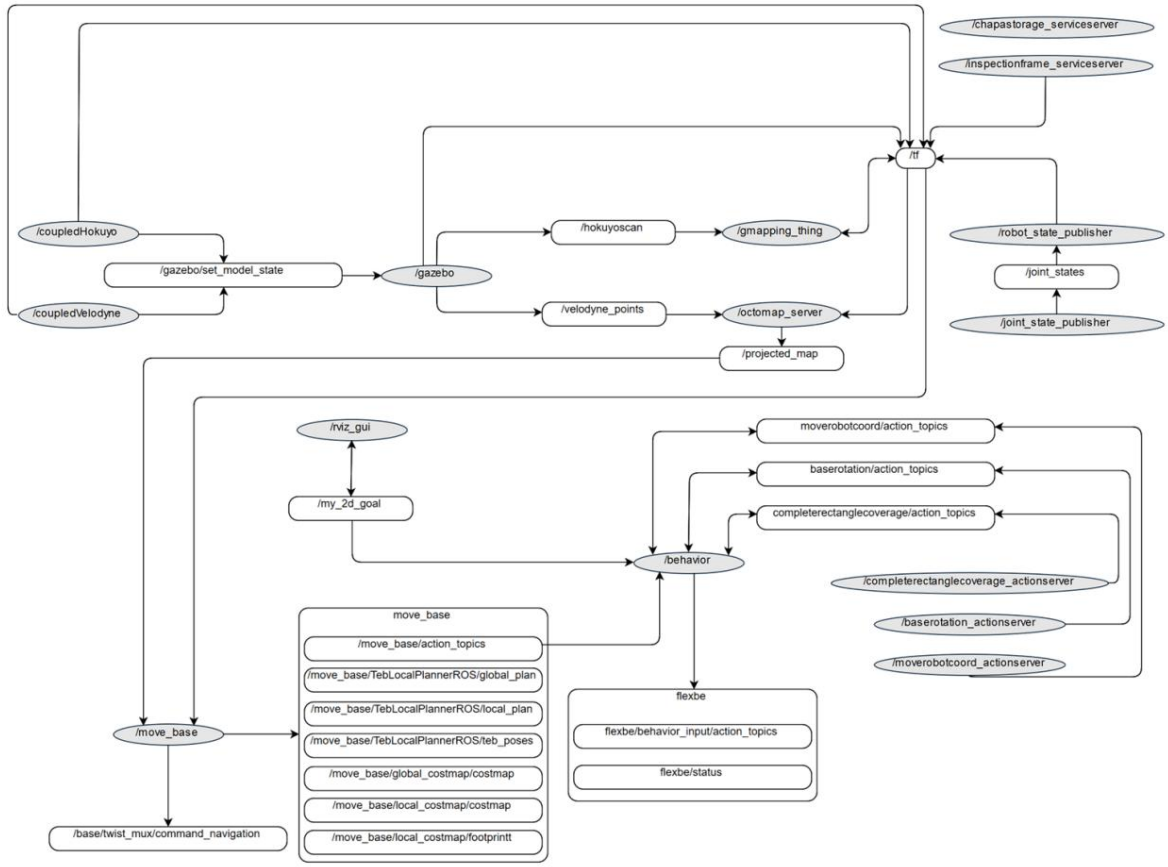

Abbildung 11: Grafische Darstellung der Hauptknoten, dargestellt durch Ellipsen, und *Themen* , die an der Arbeit beteiligt sind

3 Inspektionspfaderzeugungsumgebung und Reparatur

Wie in anderen Dokumenten diskutiert, wird die Implementierung der Erzeugung und Ausführung von Inspektions- und Reparaturpfaden durch eine Zustandsmaschine ausgeführt. Dieser Abschnitt wird verwendet, um den Prozess der Implementierung des Verhaltens und der interessierenden Parameter zu erläutern.

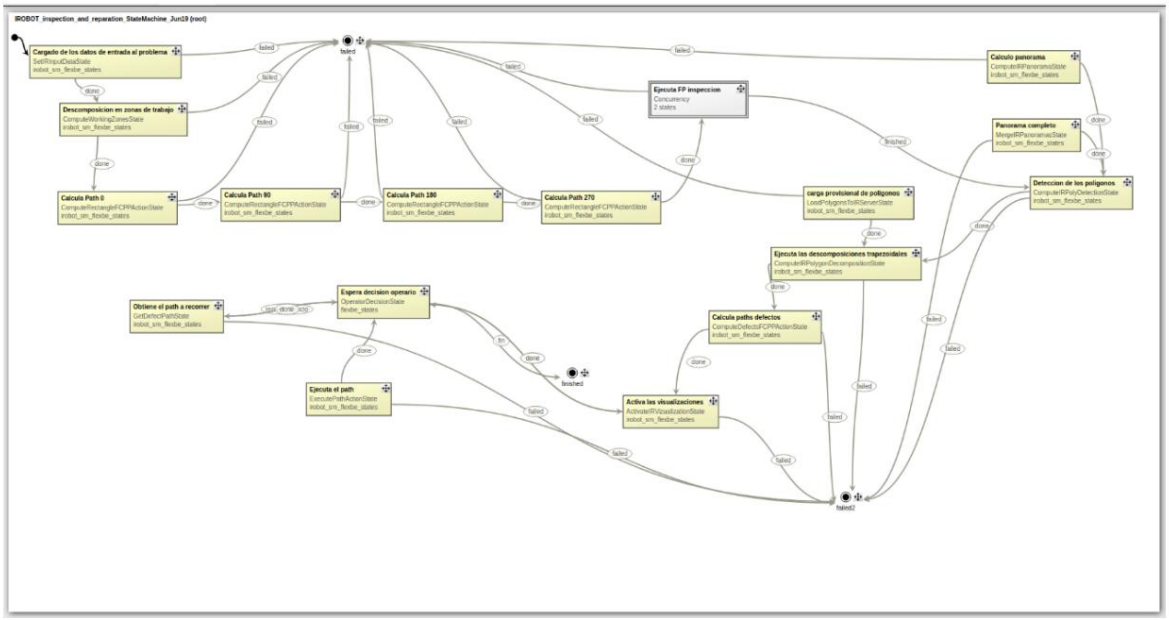

Abbildung 12: Zustandsmaschine im Zusammenhang mit der Generierung von Inspektions- und Reparaturpfaden.

3.1 Ausführung Zur

Ausführung dieser Umgebung ist eine .sh-Datei enthalten, die vom Terminal aus mit bash launchenvironmentsummit_ir_flexbe.sh ausgeführt werden kann.

Die Dummy-Position des Dummy-Tools kann in aux_controllers/main_coupledtool geändert werden:

```
19
20 Abst.x=0,70;
21 Dist.y=0,60;
22 Abstand z=0,25;
23
24 CoupledRobot gekoppelt (SUMMITXL,dist, 0, -pi/2, 0,
        "monkey_wrench_tool","coupled_tool", true);
25
```

In den Zeilen 20, 21 und 22 können die Meterwerte der Übersetzung geändert werden. Wenn es erwünscht ist, die Ausrichtung des Werkzeugs zu modifizieren, können Roll, Nick und Gieren jeweils im dritten, vierten und fünften Parameter des Konstruktors (Zeile 24) modifiziert werden.

Zustandsmaschinenparameter können auf der Registerkarte Verhaltens-Dashboard geändert werden und sollten immer mit den simulierten Werten übereinstimmen.

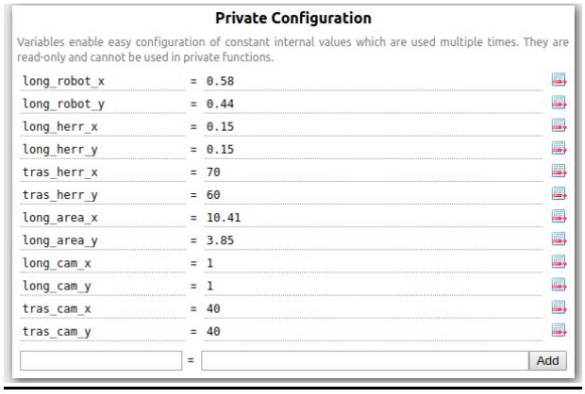

Abbildung 13: Konfigurierbare Parameter der Zustandsmaschine in der FlexBE-Schnittstelle.

Die *langen* Parameter beziehen sich auf die Größe in x und y des betreffenden Elements und *nach* der Translation, ebenfalls in beiden Koordinaten, desselben in Bezug auf das Zentrum des Roboters. Die Abkürzung *Werkzeug* bezieht sich auf das Reparaturwerkzeug, *Nocken* auf die Abdeckung mit dem Vision-Sensor, *Bereich* auf den Bereich der Platte und *Roboter* auf die Basis des mobilen Roboters. Widersprüchliche Werte für die Parameter können zu fehlerhaftem Verhalten der Simulation führen.

Die Ausführung der Simulation erfolgt automatisch bis zur Auswahl des zu reparierenden Defekts. Um den Standard auszuwählen, müssen Sie den Hauptdienst dieser Umgebung anrufen.

```
1  ~$ rosservice call /irobot_ir_data "load_inputdata: false load_polygon: false get_input_data:
2                                                                                    false
3
4                                                                        get_working_zonesh: falsch
5                                                                        get_working_zonesc: falsch
6                                                                              get_rot_zone: false
7                                                                     compute_adquireimgs: false
8                                                                         compute_panorama: false
9                                                                          merge_panoramas: falsch
10                                                                    compute_detectpolys: falsch
11                                                                 compute_fcpp_defect_i: falsch
12                                                                    compute_fcpp_zone_i: falsch
13                                                                 execute_fcpp_defect_i: falsch
14                                                                    execute_fcpp_zone_i: falsch
15                                                  change_defect_reparation_goal: falsch get_poly_i: falsch
16                                                                           get_defect_i: falsch
17                                                                         get_defect_path: falsch
18                                                                  activate_vizs: falsch long_robotx:
19                                                                                  0.0
20
21                                                                              long_roboty: 0.0
22                                                                               long_herrx: 0.0
23                                                                               long_herry: 0.0
24                                                                 tras_herrx: 0.0 0.0, z: 0.0} tras_herry: 0.0
25
26                                                                              long_camx: 0,0
```

```
27                                                  long_camy: 0,0
28                                                  tras_camx: 0,0
29                                                  tras_camy: 0,0
30                                                long_areax: 0,0
31                                                long_areay: 0,0
32                                                      input_poly:
33                                                          Punkte:
34                                  - {x: 0,0, y: 0,0, z: 0,0} poly_id: 0
35                                                      default_id: 0
36                                                  zone_id: 0"
37
```

Wie man sieht, enthält der Aufruf viele Parameter, die intern verwendet werden. Um es aufzurufen, schreiben Sie einfach „rosservice call irobot_ir_data“ in ein Terminal und drücken Sie zweimal die Tabulatortaste, um den Befehl automatisch mit den Standardwerten zu vervollständigen. Um den zu reparierenden Defekt auszuwählen, geben Sie einfach seine Nummer in der *id_defect an* und aktivieren Sie die Änderung des *Zielpolygons im Parameter change_defect_reparation_goal.*

4 Zusätzliche Pakete

Zusätzlich zu den genannten erfordert der Code die Installation mehrerer anderer Pakete im System. Diese können direkt mit dem Linux-Befehl installiert werden:

Sudo apt-get install ros-kinetics-*

Wobei * der Name des Pakets ist. Manchmal erfolgt die Installation mehrerer Pakete mit demselben Befehl, daher wird empfohlen, die Ros-Wiki-Seite für jedes der Pakete zu konsultieren. Es wird auch empfohlen, die Registerkarte im Terminal in Ubuntu zu verwenden, um die möglichen Autovervollständigungsoptionen anzuzeigen.

Aktionslib

actionlib_enhanced
actionlib_lisp actionlib_msgs
actionlib_tutorials

Amkl

Winkel
aruco

aruco_detect
aruco_msgs
aruco_ros
audio_common_msgs
base_local_planner bond

bondcpp
bondpy
kalibrierung_msgs
kamera_kalibrierung
kamera_kalibrierung_parser
kamera_info_manager catkin

cl_utils
class_loader
clear_costmap_recovery
cmake_modules collada_parser
collada_urdf
komprimierte_tiefe_image_transport
komprimiertes_image_transport control_msgs
control_toolbox controller_interface
controller_manager controller_manager_msgs
costmap_2d costmap_converter costmap_cspace
costmap_cspace_msgs costmap_prohibition_layer
costmap_queue

cpp_common
cv_bridge
Depth_Image_Proc
Diagnostic_Aggregator
Diagnostic_Analysis
Diagnostic_Common_Diagnostics
Diagnostic_msgs Diagnostic_Updater

dynamic_reconfigure
dynamic_tf_publisher
eigen_conversions
eigen_stl_containers fiducial_msgs
fiducial_slam

Filter

forward_command_controller gazebo_dev
gazebo_msgs gazebo_plugins gazebo_ros
gazebo_ros_control gencpp

Gattung
genlisp
genmsg
gennodejs
genpy
geographic_msgs
geometric_shapes
geometry_msgs
gl_dependency gmapping
graph_msgs
hardware_interface
hector_gazebo_plugins
image_geometry image_proc
image_publisher image_rotate
image_transport image_view
interactive_marker_tutorials
interactive_markers
joint_limits_interface
joint_state_controller
joint_state_publisher joint_states_settler
jsk_footstep_msgs jsk_gui_msgs jsk_hark_msgs
jsk_recognition_msgs jsk_recognition_utils
jsk_rviz_plugins jsk_topic_tools kdl_conversions
kdl_parser laser_assembler laser_filters
laser_geometry libg2o libmavconn

Freiheitssinn

librviz_tutorial map_aruco

map_msgs
map_server
mavlink

Mavros

mavros_extras
mavros_msgs
media_export

5 Bibliographie

/ [1] «DataspeedInc Verfügbar https velodyne_simulator/DataspeedInc/velodyne_simulator. [Zugriff: In: [Online]. Linie]. 20. Januar 2019]. [2] "teb_local_planner - ROS-Wiki". [Online]. Verfügbar unter: http://wiki.ros.org/teb_local_planner. [Zugriff: 21. November 2018]. [3] «neobotix/neo_mpo_700», *GitHub.* Linie]. Verfügbar In: [Unter https://github.com/neobotix/neo_mpo_700. [Zugriff: 05.10.2019]. [4] "Navigation - ROS-Wiki". [Online]. Verfügbar unter: http://wiki.ros.org/navigation. [Zugriff: 21-November 2018]. [5] "costmap_2d - ROS-Wiki". [Online]. Verfügbar unter: http://wiki.ros.org/costmap_2d. [Zugriff: 25. Januar 2019]. [6] «costmap_2d/hydro/staticmap – ROS-Wiki». http:// [In Linie]. Verfügbar In: wiki.ros.org/costmap_2d/hydro/staticmap. [Zugriff: 25. Januar 2019]. [7] «costmap_2d/hydro/inflation - ROS-Wiki». [In Linie]. Verfügbar In: http://wiki.ros.org/costmap_2d/hydro/inflation. [Zugriff: 25. Januar 2019]. [8] "geometry_msgs/PolygonStamped-Dokumentation". [Online]. http://docs.ros.org/ Verfügbar In: api/geometry_msgs/html/msg/PolygonStamped.html. [Zugriff: 26. Januar 2019].