Der erste Schritt besteht in der Implementierung einer fähigen Infrastruktur, um sowohl die effiziente Implementierung als auch Erprobung von Lösungsansätzen zu ermöglichen. Nach dem Vorbild von [1] wird eine Kombination aus ROS und Gazebo verwendet, wodurch ein Algorithmus zunächst in der Simulationsumgebung Gazebo getestet und im unmittelbaren Anschluss auf den realen Roboter übertragen werden kann. Bei Gazebo handelt es sich um eine Simulationsumgebung, die einerseits komplexe Dynamik in einer dreidimensionalen Welt zu simulieren vermag, andererseits über ein Plugin mit ROS gekoppelt werden kann, wodurch eine einzelne Implementierung sowohl für die Simulation als auch den realen Roboter verwendet werden kann.

Um den Turtlebot in ROS zu modellieren, wird das sogenannte Universal Robotic Descripction Format (URDF) verwendet [1], welches einer XML-ähnlichen Struktur folgt. In [1]wird die URDF-Datei um zusätzliche Tags erweitert, welche die Kompatibilität zu dem in Gazebo verwendeten Simulation Description Format (SDF) schaffen. Um die Simulationsergebnisse zu visualsieren verwenden Takaya et al. Das RViz Packet.

Die ROS Architektur wird in drei Ebenen: Filesystem, Computation Graph und Community, unterteilt.

Die untereste Ebene, das Dateisystem, setzt sich aus den elementaren Bausteinen von ROS zusammen, wobei als elementarer Baustein so gennante Packages verwendet werden. Dabei handelt es sich Ordner, welche die minimalen Anforderungen an ein ausführbares Programm unter ROS enthalten. Die Dokumentation eines Package erfolgt in Form eines Manifest, welches relevante Information wie Abhängikeiten und Compiler-Einstllungen enthält. Um umfangreiche Funktionalitäten zu erreichen, werden mehrere Packages zu seinem so genannten Stack zusammengefasst. Als Beispiel dient ein Navigationsstack, der sich aus Funktionen für die Auswertung von Sensorik, SLAM-Verfahren, Pfadplannung und Ansteuerung der Aktorik zusammensetzt, welche jeweils als eigenes Package implementiert werden. Die Dokumentation des Stacks erfolgt ebenfalls in Form eines Manifest, welches in diesem Fall als Stack-Manifest bezeichnet wird.

Am Beispiel von Stacks wird bereits deutlich, dass ein Grundprinzip von ROS darin besteht, dass mehrere Prozesse interagieren. Um Daten zwischen Prozessen auszutauschen werdne Nachrichten verwendet, welche im Rahmen von ROS als Messages bezeichnet werden.

Zusätzlich gibt es noch Services, die ähnlich wie Nachrichten allerdings nur kurz beschrieben werden und deshalb auch an dieser Stelle nicht weiter ausgearbeitet werden.

In der zweiten Ebene wird ROS als Netzwerk beschrieben, in welchem die verschiedenen Prozesse verknüpft sind. Die einzelnen Elemente des Netzwerks werden als Nodes bezeichnet, wobei es sich um Prozesse handelt, die über das ROS Netzwerk miteinander verbunden sind. Ein besonderer Bestandteil des Netzwerks stellt der so genannte Master dar, der eine Namen Registrierung und Lookup für die restlichen Nodes bereitstellt, wodurch dieser notwendig für die Kommunikation zwischen den Nodes wird. Eine weitere zentrale Einheit bildet der Paramter Server, welcher die Möglichkeit bietet, Daten in einer zentralen Position mithilfe von Keys zu speichern. Die Kommunikation zwischen den Nodes erfolgt über Nachrichten, welche in Form von Message Types auf der Ebene des Dateisystems festgelegt wurden. Bei der Nachrichtenkommunikation wird ein Publish/Subscribe Schema eingehalten, welches darauf basiert, dass alle Nachrichten mit einem Namen, der als Topic bezeichnet wird, versendet werden. Nodes können Topics subscriben, woraufhin alle Nachrichten dieser Topic an sie weitergeleitet werden. Da es sich bei der Nachrichtenkommunikatoin mithilfe von Topics um eine many-to-many Verbindung handelt, können Topics nicht genutzt werden, um eine direkte Kommunikation zwischen zwei Nodes umzusetzen, wofür die so genannten Services verwendet werden. Wenn eine Node einen Service anbietet, muss dieser einen einzigartigen Namen besitzen, woraufhin andere Nodes den Service über das ROS-Netzwerk ansprechen und nutzen können. Als letzte Einheit der Computation-Graph-Ebene sind die so genannten Bags zu nennen, welche als Datenspeicher fungieren. Beispielsweise können Nachrichten in Bags gespeichert werden, um den gesamten Werteverlauf eines Sensors im Anschluss an ein Experiment zu rekonstruieren.

Erstellen von eigenen Packages erfolgt mithilfe des Befehls

Roscrate-pkg [package\_name] [depend1] [depend2] [depende3]

Beispiel: roscreate-pkg chapter2\_tutorials std\_msgs rospy roscpp

Erstellt das Package chapter2\_tutorials, das von den Packeten std\_msgs, rospy und roscpp abhängt. Wichtig dabei ist, dass das neue Packet bzw. der Ordner, der das Packet darstellt, auf dem ROS\_PACKAGE\_PATH liegt.

Anleitung nach ROS-Wiki für Turtlebot Simulation in Gazebo/Visualisierung mit Rviz [3]:

Intsallation:

sudo apt-get install ros-kinetic-turtlebot ros-kinetic-turtlebot-apps ros-kinetic-turtlebot-interactions ros-kinetic-turtlebot-simulator ros-kinetic-kobuki-ftdi ros-kinetic-rocon-remocon ros-kinetic-rocon-qt-library ros-kinetic-ar-track-alvar-msgs

Fertiges Beispiel, um den Turtlebotsimulation in Gazebo zu starten:

roslaunch turtlebot\_gazebo turtlebot\_world.launch

Durch das festlegen der Umgebungsvariable TURTLEBOT\_GAZEBO\_WORLD\_FILE kann eine .world-Datei bestimmt warden, die als Umgebung für den Trutlebot in Gazebo genutzt wird.

Um die Geschwindigkeit de sRoboters zu setzen, muss die Topic /mobile\_base/commands/velocity gepublished werden, wofür die Nachricht geometry\_msgs/Twist mit dem Format:

rostopic pub /mobile\_base/commands/velocity geometry\_msgs/Twist "linear:

x: 0.1

y: 0.0

z: 0.0

angular:

x: 0.0

y: 0.0

z: 0.0"

ausgegeben werden.

[1] Kenta Takaya, Toshinori Asaiy, Valeri Kroumov, Florentin Smarandach: „Simulation Environment for Mobile Robots Testing Using ROS and Gazebo“

[2] Aaron Martinez, Enrique Fernandez: “Learning ROS for Robotics Programming”

[3] TurtleBot – ROS-Wiki: http://wiki.ros.org/Robots/TurtleBot?distro=indigo#turtlebot.2BAC8-Tutorials.2BAC8-indigo.Simulation