Inhaltsverzeichnis

Kapitel 1

Einleitung

Seit einiger Zeit wird an der Hochschule Karlsruhe die Projektidee eines künstlichen Chors verfolgt, dessen Ziel darin besteht, eine Audiospur simultan in mehreren Tonhöhen wiederzugeben, um einen mehrstimmigen Chor zu erzeugen. Das zugrundeliegende technische Prinzip lässt sich recht leicht erläutern: die originale Tonspur wird mittels FFT in den Frequenzbereich transformiert, wo das Signal in die gewünschte Tonhöhe bzw. Frequenzbereich verschoben wird. Anschließend werden die Signale zurück in den Zeitbereich transformiert und abgespielt. Allerdings gestaltet sich die Umsetzung weitaus anspruchsvoller als das theoretische Konzept. Beispielsweise führen Aussetzer im WLA-Netzwerk, Puffergrößen der Audiotreiber und diskrete Stützstellen des Frequenzspektrums zu unerwarteten Problemen, die sich in Form von knackenden Lautsprechern Gehör verschaffen. Nichtsdestotrotz hat der Entwicklungsprozess den Punkt erreicht, an dem der Chor noch kaum unerwünschte Töne von sich gibt, weshalb nun die erste große Erweiterung angestrebt wird. Aktuell setzt sich das System aus einem Master-PC und vier BeagleBones zusammen, die jeweils mit einem Lautsprecher ausgestattet in einem Raum verteilt werden. Auf dem Master wird eine Tonspur eingespielt, die er in die gewünschten Tonhöhen pitcht und per WLAN an die BeagleBones sendet, welche jeweils eine der Stimmen über den Lautsprecher wiedergeben.

Im nächsten Schritt soll der Chor mobilisiert werden. Das heißt jedes BeagleBone plus Lautsprecher wird auf einem Roboter montiert, die bei einer Vorführung einen beliebigen Raum betreten können und diesen auf der Suche nach einem passenden Podium erkunden. Wurde Letzteres gefunden, so sollen sich die Roboter dort im Halbkreis formieren und anschließend vorsingen.

Die neue Aufgabe, die sich mit der autonomen Navigation der Roboter beschäftigt, kann vollkommen entkoppelt von dem Chor betrachtet und bearbeitet werden. Dieser zweite Entwicklungsstrang befindet sich allerdings noch im Anfangsstadium, da sich bisher nur ein einzelnes Entwicklungsprojekt dem Thema gewidmet hat. In dieser Arbeit wurde ein erster Grundstein gelegt, der sich einerseits aus der Auswahl und Inbetriebnahme der Roboter, andererseits aus einem ersten Proof-of-Concept in Sachen Navigation zusammensetzt.

Als mobile Plattform für die Lautsprecher fiel die Wahl auf den Turtlebot 2, der von Willow Garage entwickelt wurde. Der Roboter setzt sich aus der Kobuki-Basis, einem Rechner und einer ASUS-Xtion-Pro-Live zusammen, wobei Letztere sowohl Kamera als auch Tiefensensor zur Verfügung stellt. Der Turtlebot 2 bringt als Vorteil mit sich, dass der Roboter weite Anwendung im Bereich von Hobby- und Forschungsanwendungen findet, weshalb eine breite

Community-Untersützung zur Verfügung steht. Ein in dieser Arbeit wichtiges Beispiel stellt die Simulation des Roboters mithilfe von gazebo dar: Hier besteht bereits eine vollständige Integration des Turtlebot 2, die open-source zugänglich ist. Im Rahmen der Vorgängerarbeit wurde der roboter zusätzlich mit dem Laserscanner Tim551 der Marke SICK ausgestattet, der als primärer Sensor für die Navigation verwendet wird.

Die Programmierung erfolgt mithilfe des sogenannten Robot-Operating-System (ROS), wobei es sich ironischerweise um kein Betriebssystem sondern eine Middleware handelt, die eine Vielzahl von Tools und Paketen für die autonome Navigation bereitstellt. Dadurch müssen die Algorithmen für die Navigation und Kartographierung nicht eigenständig implementiert werden, sondern können in Form von ROS-Paketen eingesetzt werden, sodass sich die Inbetriebnahme auf die Parametrisierung der ROS-Funktionen beschränkt. Außerdem beinhaltet ROS ein Netzwerk, über das die Roboter kommunizieren können. Mithilfe der Kommunikationskanäle können auch externe Werkzeuge auf relevanten Daten zugreifen, womit komfortable und effiziente Debugging-Wege geschaffen werden. Umgekehrt können zu Beginn die Roboter auch aus dem Netz entfernt und durch ein Simulationstool wie gazebo ersetzt werden. So können die Algorithmen zunächst anhand einer Simulation erprobt und konfiguriert werden und im Anschluss unverändert auf die Roboter übertragen werden.

Die im Vorgängerprojekt erarbeitete Navigationslösung basiert auf dem ROS-Navigation-Stack und setzt sich aus zwei Teilen zusammen. Im ersten Schritt wird mit dem ROS-Paket **`hector_slam`** eine Karte der Umgebung aufgezeichnet. Anschließend navigieren die Roboter anhand der Karte zu dem vorgegebenen Ziel, wobei das ROS-Paket **`move_base`** zum Einsatz kommt. Bei dessen Konfiguration wurden zwei verschiedene Ansätze verfolgt: Bei dem Ersten erfolgt die Lokalisierung des Roboters ausschließlich anhand der Odometriedaten. Der zweite Ansatz greift auf die AMC-Lokalisierung zurück. Der Vorteil der zweiten Variante liegt darin, dass die Ausgangsposition des Roboters nicht bekannt sein muss. Bei der Navigation mittels Odometrie übertragen sich sämtliche Fehle bei der Angabe der Anfangsposition unmittelbar auf die Navigation. Im Gegensatz dazu verspricht die AMCL-Variante - zumindest theoretisch - ein höheres Maß an Robustheit. Allerdings konnte diese Hoffnugn in den Experimenten nicht bestätigt werden, weshalb letzten Endes die Odometrie-Navigation verwendet wurde. Zusätzlich sei angemerkt, dass diese Lösung nicht im Stande ist, Hindernisse, die nicht auf der Karte verzeichnet sind, während der Navigation zu erkennen, geschweige denn, denen auszuweichen.

An dieser Stelle knüpft die vorliegende Arbeit an: Die Navigation soll an den Punkt gebracht werden, wo die Roboter im Stande sind, sich selbst auf der Karte zu lokalisieren, unerwartete Hindernisse zu detektieren und diesen entsprechend auszuweichen. Hierfür soll auf bestehende Möglichkeiten des ROS-Navigation-Stack zurückgegriffen werden, um die Menge von Lösungsmöglichkeiten einzuschränken. Auch die Anforderung, dass die Navigation auf eine mit **`hector_slam`** aufgezeichnete Karte zurückgreifen kann, wird übernommen. Im ersten Schritt wird lediglich ein einzelner Roboter betrachtet. Sobald dieser die Anforderungen im Bereich der autonomen Navigation erfüllt, werden die Ergebnisse auf eine Gruppe von vier Robotern übertragen.

Neben den rein funktionalen Zielen legt diese Arbeit einen besonderen Wert auf die dabei verwendete Vorgehensweise. Die Arbeit beginnt bei den Grundlagen. Der Navigation-Stack

soll in seine funktionalen Einheiten zerlegt werden, deren Funktionsprinzipien im Detail erläutert werden sollen. Mithilfe der so gewonnen Erkenntnisse sollen die Komponenten konfiguriert und wieder zusammengesetzt werden. Die Arbeit erhebt den Anspruch nicht nur eine Lösung zu erzielen, die die Anforderungen erfüllt, sondern auch zu kläre, weshalb die Anforderung erfüllt werden konnten. Mit der Simulation können die Teilprobleme nicht nur entkoppelt betrachtet werden, sondern auch gezielt untersucht und illustriert werden. Es sollen aussagekräftige Anwendungssituationen konstruiert werden, die die Funktionsprinzipien der Algorithmen entweder in der Simulation oder Realität aufzeigen.

Kapitel 2

Aufbau des Gesamtsystem und Infrastruktur

In der Endvorstellung des Projektes navigieren vier Roboter simultan durch denselben Raum, woran recht leicht ersichtlich wird, dass ein Weg zum Datenaustausch zwischen den Robotern geschaffen werden muss. Auch eine zentrale Steuereinheit soll in der Lage sein, alle Roboter anzusprechen und zu dirigieren. Selbst ein einzelner Roboter stellt bereits einige Anforderungen an die Kommunikationsstruktur: Da eine erfolgreiche Navigation auf dem Zusammenspiel mehrerer komplexer Algorithmen basiert, müssen externe Analysetools relevante Daten abgreifen, die zwischen den Komponenten ausgetauscht werden. Diese Form des Debuggings ist unerlässlich, um das Verhalten des Roboters nachvollziehen zu können. In diesem Projekt erfüllt ROS diese Anforderungen, weshalb in den folgenden Abschnitten das Kommunikationskonzept unter ROS erläutert wird. Auch die Simulations- und Analysewerkzeuge, die in der Arbeit zum Einsatz kommen, werden vorgestellt und deren ROS-Schnittstellen erläutert.

Jeder TurtleBot ist mit einem Mini-PC ausgestattet, auf denen Ubuntu 14.04 und ROS-Indigo installiert sind. Als zentrale Steuereinheit des Roboterverbundes fungiert ein weiterer Mini-PC, der über WLA-Netzwerk mit den Roboter-PCs verbunden ist. Das private Netzwerk wird mittels eines TP-Link Routers verwaltet, in dem auch weitere Entwicklungsrechner beitreten können. Auf dem Master-PC wird **`ros_core`** ausgeführt, über den die Kommunikation abläuft. Als Kommunikationsmittel werden in ROS Nachrichten verwendet, die jeweils unter einer so genannten **`topic`** veröffentlicht werden. Beispielsweise versendet der Laserscanner zyklisch eine Nachricht des Typs **`sensor_msg/laser_scan`** unter der topic **`/scan`**. Alle Nodes, die an den Sensordaten interessiert sind, abonnieren die Topic. Indem Analysewerkzeuge wie MATLAB relevante Topics mithören, können die zugehörigen Daten aufgezeichnet und visualisiert werden. Umgekehrt ist es auch möglich den TurtleBot vollständig durch eine Simulation zu ersetzen. Gazebo veröffentlicht alle Topics, die für gewöhnlich von dem TurtleBot versendet werden, wodurch dieser ersetzt wird. Die restlichen Bestandteile des Netzwerkes bleiben erhalten, weshalb der Wechsel zwischen Realität und Simulation mühelos abläuft.

Die Idee Gazebo in Verbindung mit ROS für die Simulation zu verwenden entstammt der Arbiet [3]. Dort wurde unter einer ähnlichen Konfiguration ein P3-DX Roboter verwendet. Der primäre Vorteil dieser Simulationsstruktur besteht darin, dass die Algorithmen und Konfigurationen unverändert von der Simulation auf den Roboter übertragen werden können. Gazebo

bietet auch für die TurtelBots eine vollständige Unterstützung, das heißt es besteht eine frei zugängliche Integration des TurtleBot in Gazebo. Allerdings existiert keine Implementation des hier verwendeten Laserscanner, Tim551, weshalb die Sensorik nicht exakt abgebildet werden kann. Prinzipiell können die Sensoren eingepflegt werden, was jedoch im Rahmen dieser Arbeit aus zeitlichen Gründen nicht erfolgt ist. Daraus resultiert der Nachteil, dass die Simulation nur beschränkt Schlüsse auf die Realität zulässt. Nichtsdestotrotz ergeben sich zwei wichtige Vorteile: Einerseits können die verschiedenen Konfigurationen des ROS-Navigation-Stack anhand der Simulation erprobt und auf ihre Richtigkeit überprüft werden, bevor sie auf reale Anwendungsfälle übertragen werden. Andererseits können im Rahmen der Simulation die Komponenten des Navigation-Stack vollkommen entkoppelt werden. Beispielsweise hängen die Ergebnisse der Navigations-Algorithmen stark von der Qualität der Lokalisierung ab, weshalb die Planungs- und Lokalisierungsproblematik in einem realen Umfeld nicht getrennt untersucht werden können. In der Simulation können Positionsdaten jedoch unmittelbar abgegriffen und der Navigation zur Verfügung gestellt werden. Insofern stellt Gazebo ein mächtiges Werkzeug für die schrittweise Inbetriebnahme des Navigation-Stacks dar.

MATLAB kann im Rahmen der Robotics-Toolbox ebenfalls als ROS-Node betrieben werden, wodurch eine Schnittstelle zwischen MATLAB und ROS geschaffen wird. Hier bringt MATLAB den Vorteil mit sich, dass es für die Implementierung von Algorithmen oftmals besser geeignet ist als C++ oder auch Python. Insbesondere die Möglichkeiten im Bereich der Visualisierung von Karten und Plots erweisen sich als mächtige Werkzeuge bei der Implementierung und Erprobung von Algorithmen. Außerdem können simple Anwendungsszenarien, wie sie zum Beispiel im Rahmen der diskreten Planung auftreten, vollständig mit MATLAB simuliert werden.

Als letztes Werkzeug sei an dieser Stelle RViz genannt, das als ROS-Paket vorliegt und für die Visualisierung von Robotikanwendungen konzipiert wurde. Mithilfe von RViz können sowohl Karten als auch Roboter und Pfade graphisch dargestellt werden. Da RViz für die Anwendung mit ROS konzipiert wurde, stehen vollständige Konfiguration für den Einsatz mit der autonomen Navigation zur Verfügung, die Out-of-the-Box genutzt werden können.

Literaturverzeichnis

[1] Sebastian, Thrun; Wolfram, Burgard; Dieter Fox: „Probabilistic Robotics“, 1. Auflage, Massachusetts Institute of Technology 2006, MIT Press

[2] LaValle, Steven M. „Planning Algorithms“, 1. Auflage, Cambridge 2006, Cambridge University Press

[3] Takaya, Kenta; Asai, Toshinori; Kroumov, Valeri; Smarandache, Florentin „Simulation Environment for Mobile Robots Testing Using ROS and Gazebo“, 2016, 20th ICSTCC

[4] Pearl, J. „Heuristics“, Addison-Wesley, 1984

[5] Fikes, R. E.; Nilsson, N. J. „STRIPS: A new approach to the application of theorem proving.“Artificial Intelligence Journal, 1971

[6] Clearpath Robotics „TurtleBot Data Sheet “2015

[7] global_planner: `http://wiki.ros.org/global_planner` , aufgerufen am 30.11.2017

[8] navfn algorism[sic]: `https://answers.ros.org/question/11388/navfn-algorism/?answer=16891#answer-container-16891`, aufgerufen am 30.11.2017

[9] costmap_2d: `http://wiki.ros.org/costmap_2d`, aufgerufen am 30.11.2017

[10] staticmap: `http://wiki.ros.org/costmap_2d/hydro/staticmap`, aufgerufen am 30.11.2017

[11] obstalces: `http://wiki.ros.org/costmap_2d/hydro/obstacles`, aufgerufen am 30.11.2017

[12] inflation: `http://wiki.ros.org/costmap_2d/hydro/inflation`, aufgerufen am 30.11.2017

[13] dwa_local_planner: `http://wiki.ros.org/dwa_local_planner`, aufgerufen am 30.11.2017