HUMAN MOBILITY MODELS FOR INDOORS

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Daniel Moser
2013

Leiter der Arbeit:
Professor Dr. Torsten Braun
Dr. Desislava Dimitrova
Institut für Informatik und angewandte Mathematik

Contents

List of Figures

List of Tables

Listings

Summary

The ability to determine the position of persons of interest and to follow them as they move is in the core of personalising services such as targeted advertisement, real-time discount offering and navigation in indoor premises. To achieve such tracking through wireless devices a localisation system should be thoroughly tested prior to deployment. Simulations or emulations are not appropriate to achieve that since they make abstractions of the real world and test beds are a much better option. A test bed should be able to represent the system behaviour as well as the human mobility behaviour.

With this thesis, the author addresses the problem of representing human mobility accurately by robots in order to enable repeatability and increase the reliability of testing with wireless networks. The thesis provides the means to deploy mobile robotic-based platforms for testing of localisation systems. The platforms are relevant for other research areas dealing with mobility as well.

To achieve the goal, the author implements two mobility models on a low end robotic system and evaluates multiple aspects of the robot's desired behaviour (e.g. driving accuracy, following implemented model). The findings of the thesis show that using basic robotic hardware poses some challenges related to movement accuracy and sensoric feedback. In the context of mobility accuracy some linear and angular distortions were observed in the followed path. Also the ability to cope with even small obstacles and the issue of battery lifetime pose challenges. Hence, depending on the choice of robotic platform one should expect certain inaccuracy in location. The wireless connectivity, however, remains unaffected, as we will also show.

Chapter 1

Overview

1.1 Introduction

The process of manual testing can be tedious, time consuming and prone to human errors. This applies to any field of operations: be it software, hardware, complex systems, such as wireless test beds, or many other fields in computer science. Moreover, testing and evaluation scenarios should be reproducible for statistical reliability and result replication. Achieving this by manual testing is challenging. Automated testing can significantly improve evaluation repeatability. Simulations, for example, offer such means but often cannot reflect real-life aspects. Test beds are a better approach concerning the latter but repeatability is more difficult to achieve. In the context of wireless network experimentation, varying propagation conditions and node mobility are among the most important aspects to reproduce, when feasible.

Within this thesis the author investigates approaches to provide replication of node mobility in wireless sensor test beds. The scope of the thesis involves the preparation of a robotics platform acting as a mobility platform and the implementation of appropriate mobility models.

The author first takes a look at different state-of-the-art indoor human mobility models. The constrained mobility model was selected for implementation since it has the potential to realistically reflect a person's movement thanks to the fact that it only accounts for possible paths in a real environment. The more commonly referred in literature random walk model is used as benchmark. The mobility models are further described in Chapter 2.

Due to the evolution in electronics during the last years, robotics hardware in the low cost range has emerged and gives many choices for small scale robotics experiments. In addition, to support the unfamiliar developer, the hardware should best provide a plug and play hardware interface. In Chapter 3 the author evaluates different available robots below $1000 according to the available specifications and takes a look at some possible extensions for the chosen platform.

In terms of software, discussed in Chapter 4, the goal is to provide a standard system easy to operate and modify requiring only basic Linux knowledge. Also, in case of new hardware being bought, the implemented mobility model should work with as little adjustments as possible. The Robot Operating System provides these possibilities, owing to the fact that the implementations are abstracted from hardware and can be easily adapted to new platforms.

Chapter 5 takes a look at the practical part of the thesis and provides detailed information on how to install Linux and the Robot Operating System on the Gumstix platform. Additionally, the

Chapter provides the reader with the basic knowledge on how to run the implemented mobility models.

After the complete implementation and installation of the system, the author evaluates the accuracy of the robot (see Chapter 6). Special focus will be put onto the accuracy of driving distance and angular deviation due to their importance in accurately reproducing any mobility model, especially concerning indoor spaces. The findings reveal that the system's low price comes at the cost of low accuracy.

The implemented mobility models can be modified to introduce variation in the waiting time distribution and, in the case of the constrained reality model, to adapt the probabilistic model according to which path is selected next, e.g., allowing some path to have higher chances to be selected.

1.2 Contributions

The author first evaluated several papers dealing with human mobility models to find out which models are currently used in different scientific areas of computer science and which would be feasible to be implemented. The contributions in the hardware section (see Chapter 3) included searching for documentation and information on different low cost robotics platforms and additional hardware extensions.

In terms of software platforms (see Chapter 4) an overview of different Linux distributions is provided. The purpose was to find the most compatible system to work with the Gumstix Overo Fire. The author investigated the installation possibilities of the Robot Operating System for both the **Fuerte** and **Groovy** variants and decided to compile the whole framework by hand.

The thesis also offers a possibility to use an additional package (`turtlebot_apps`) in **Fuerte** to enable remote steering by an Android phone. Due to the broken backwards compatibility and a new setup routine in **Groovy** the installation of this package could not be reconstructed and will not be described in detail. However, the author will provide an image of his experimentation system with a working version of the Fuerte version of the Robot Operating System including remote steering.

A central contribution of the thesis is the implementation and testing of two mobility models, namely, random walk and constrained mobility. The mobility models were implemented on top of the Robot Operating System and written in Python. Moreover, the author conducted detailed tests on the robot's ability to reproduce the intended movements. That allowed him to validate the controlling and movements of the robot.

Working on a new project including new hardware poses many challenges and brings up a vast list of lessions learned. This list includes, but is not limited to:

- Rapid innovation in the consumer grade, low cost electronics and robotics sector might spoil one's set-up routine;
- Inexpensive robotics platforms tend to lack of accuracy in displacement;
- Difficult to find balance between cost and practicability of computation devices;
- Radio connectivity between the mobile platform and a static testbed is not an issue.

Chapter 2

Mobility Models

There is a wide variety of papers available, discussing human mobility models for wireless networks. The scope of this thesis requires a model that can present human mobility in distance, speed and direction. The majority of papers on the topic of node mobility focuses on contact and inter-contact times between different nodes [1, 2, 3], which is of high relevance for connectivity mechanism and protocols [4, 5]. Contact time defines the duration two nodes stay in sight without losing contact, while inter-contact time describes the time between two sightings [1].

The authors of [1] use real life data traces in a shopping mall environment to model human mobility. They oppose to using a simulated environment, as most human mobility models do not reflect real human behaviour. Especially the random walk model, used by many researchers as a reference, was rejected for being ”unnatural”. The technology used to measure device contacts is Bluetooth and based on the measurements the authors recommend possible ways to introduce different forwarding algorithms.

In [2] a pocket switched network is proposed, where different mobile network nodes can communicate with each other without the need of internet access. To collect data, the authors gave a Bluetooth communication device to conference attendees at IEEE INFOCOM 2005. Contact and inter-contact times can be approximated with the power law with an exponent smaller than one. They found that often proposed forwarding algorithms are not optimal. This is especially the case if a member of another group should be contacted which reduces the exponent of the power function significally.

The contribution [3] describes a program to simulate human mobility in different fashions to test various routing algorithms. The algorithms were evaluated by the normalized packet delivery ratio. It was shown that it can be dangerous to propose a routing algorithm based on too simple human mobility models. Hence, also our goal in this work is to strive for realism.

However, such models do not take into account the actual movements of the individual nodes. Although there are other models that simulate human movements based on different activities during the day and across wide areas (eg. cities), these are less appropriate for the current study since the thesis interest is in mobility within buildings. The author chooses the random walk model as a rather basic model, but nevertheless used by many researches, and the constrained mobility model as a more realistic model.

2.1 Random Waypoint Mobility Model

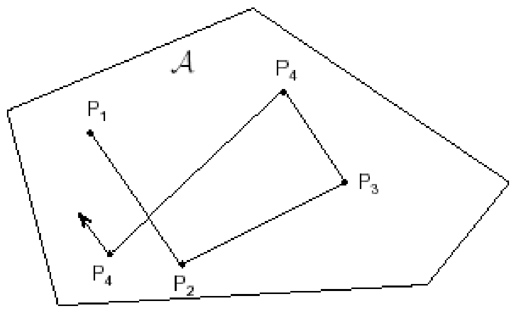

Figure 2.1: Example of the random waypoint model[7]

The first and most straightforward model considered for this thesis is the random waypoint model. It is extensively used by many different research groups for simulations in virtual environments [5, 6].

The authors of [5] used the random waypoint model in simulations for the evaluation of different routing alogrithms. However, the paper offers no critical argument against the use of the random waypoint model.

Just like [5], [6] used the random waypoint model inside a simulated environment. After multiple unrealistic results, they resorted to using a model they called random direction model, which is fairly similar to the model described in Section 2.2. Their reason was that the nodes started to converge to similar areas.

Using this model in simulations rather than real life experiments makes sense. There are no hardware caused restrictions and anomalies of the virtual nodes, which would prevent them from traveling to the defined coordinates.

The basic model consists of a mobility domain as a convex set $A \subset \mathbb{R}$, a node that moves from waypoint P_i to P_{i+1} with speed $v = 1$ and uniformly distributed waypoints, $P_i \sim U(A)$. A scenario is depicted in Figure 2.1. The standard extensions include the possibility of a random pause time at each waypoint before starting to move further on and the possibility of having random velocities for each section of the path [8].

2.2 Random Walk Mobility Model

The random walk model contains a mobile node that behaves in an random manner regarding to movement. From the randomness point of view it resembles the random waypoint model but it does not feature any pause time. This means that the node is constantly driving and only changing direction and speed after a certain time interval. In each interval the current direction and speed are independent of the preceding and following directions and speed, with the direction being uniformly distributed between 0 and 2π and the speed being distributed either uniformly or Gaussian between 0 and v_{max} [10]. An example of a track generated by the random

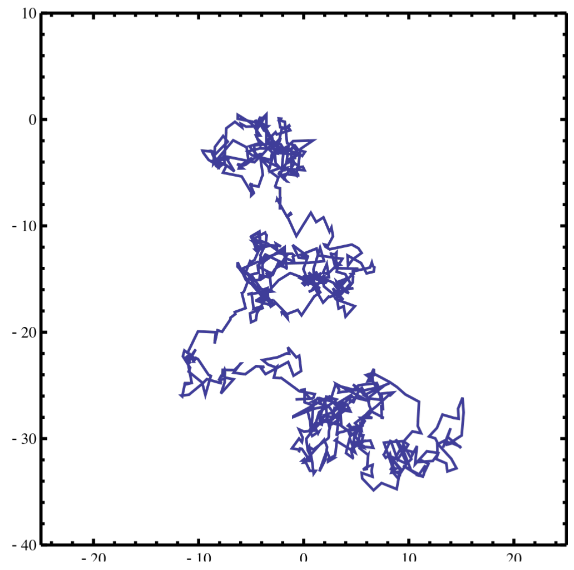

Figure 2.2: Example of the random walk model[9]

path model is shown in Figure 2.2.

The author considers this model for a standard comparison to be able to evaluate if a more complex model like the constrained mobility model provides more useful data. Many other studies use either the random waypoint or the random walk model as a baseline for first experiments and feasibility testing.

2.3 Constrained Mobility Model

The constrained mobility model offers a very appropriate way to cope with obstacle-rich environments. In its core, it uses a mobility graph consisting of vertices, standing for valid locations that can be visited, and edges as the valid paths between vertices [11]. It falls into the group of mobility models that consider geographic restrictions when computing the mobility path [12]. From a practical perspective this knowledge also minimizes the possibilities of the robot getting stuck.

There are two main groups of studies when working on human mobility models in obstacle rich environments. The first group uses maps to validate the correctness of localisation results [13, 14].

H. Liu et al. [13] provide a general overview over different localisation technologies for indoors and analize different location estimation techniques. They describe, e.g., a grid-based Bayesian robot localisation algorithm using WiFi to compute the probability of a calculated location.

In [14], D. Fox et al. recognize that all kinds of sensors (e.g. GPS, Infrared) are subordinated to bias. They propose Bayes filters to estimate the probability of a location based on previous informations and measurements.

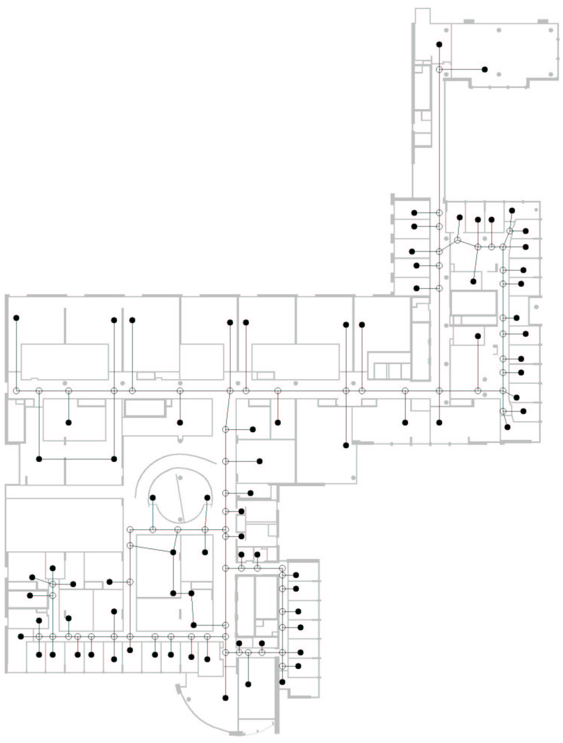
Figure 2.3: Example of the constrained mobility model[11]

The second group actively uses maps to predetermine movements within the spacial constraints [11]. The constrained mobility model falls into the second group. Due to its design the constrained mobility model can take into account obstacles such as walls, tables and others in the robot movement.

Constrained mobility uses concepts from graph theory. A graph G consists of vertices V and edges E, where $E \subseteq V \times V$ applies. Self-loops, meaning edges that start and end at the same vertex, are normally not part of a graph. Graphs can either be directed or undirected. A directed graph has an edge set consisting of ordered vertex pairs. A graph is weighted if it consists of either vertices, edges or both, tagged with a weight value [15].

The implemented constrained mobility model (see Section 4.2) consists of a non-weighted graph, which can be directed or undirected by choice, without self-loops. It offers many different possibilities for extensions. Introducing a weighted graph could be mapped to probabilities of movement and pausing patterns. A weighted graph can represent a node's (and the related user's) individual preferences, diversity in targeted end locations and variety of motivations in mobility. For instance, a female customer in a shopping mall can exhibit a different mobility pattern compared to a male customer. This difference can be easily reflected by the constrained mobility model by adjusting weights on the edges.

For a visual example, see Figure 2.3, which depicts a mobility graph overlaid over a floor plan.

2.4 Related research

Extensive research on the topic of human mobility related with network applications and wireless sensor networks has been conducted in the last several years. Some papers have already been discussed in the previous sections.

The Scuola universitaria professionale delle Svizzera italiana does extensive research on sensor networks and distributed systems. In [16], Förster et al. propose a wireless sensor network called MOTEL, where small robots equipped with sensors can be deployed and are used to do experiments related to wireless sensor networks. They use cameras for the robot localisation tracking, because autonomous robots with complex localisation and path finding algorithms often prove to be unreliable and slow.

As described in Section 2.1, many researchers use simulation software to evaluate different network systems. At UPC, E. Zola et al. [17] use OMNeT++ to simulate random waypoint and gauss-markov mobility models and evaluate cell residence time and changes between access-points (handoffs). They build a virtual 802.11 WLAN infrastructure environment where all access point have overlapping reception areas. The access points are all interconnected with one switch and a host that is constantly pinged by the simulated mobile nodes. The findings show an impact of the mobility models on handovers and network performance.

C. Fok [18] from the University of Texas introduced a test bed named Pharos, which consists of different mobile sensor nodes. Like this thesis, they use inexpensive, consumer grade robots. Through a special programming interface abstracted from hardware they implement network applications in simulation environments and can port them to physical hardware with only little work. The main focus of the Pharos test bed is repeatability and movement accuracy of the robots. More information on the Pharos test bed can be found at the wiki of the project's website [19].

Researchers at the University of Uppsala [20] developed a nomadic sensor network test bed called Sensei. The main characteristic of Sensei that distinguishes it from many other works on WSNs is the possibility to easily relocate the test bed from lab environments up to prototype deployment. Rensfelt et al. use wireless access points running a customized distribution of Linux as sensor hosts. The moving nodes can either be humans equipped with laptops or small robots with laser range finding for localisation and implemented mobility models. The focus of the publication is on repeatability of localisation testing of the WSN.

Chapter 3

Robot Hardware

3.1 Selection of Robot Platform

Making a choice of robotic hardware platform faces many challenges. The available technical specifications of the assessed platforms, especially concerning the robots, tend to be limited and incomplete. Incomplete information about a product makes selecting a specific system difficult.

A further challenge is the general availability and delivery of the products. There is a fast development in the low-cost electronics and robotics sector, which inevitably leads to many new models continuously appearing on the market. In addition, some hardware platforms might not be shipped world wide, which needs to be taken into account.

With small dimensions and a light weight platform in mind, the author faced many challenges. Mounting a notebook on top of a robotic platform would offer more computational power at the cost of more weight (around 1 kg for a standard notebook) and results in bigger dimensions and requires a cargo bay. This is, e.g., not supported by the Kobuki platform. The iRobot Create platform on the other hand offers a cargo bay and a connector for an embedded system, at no dimensional cost and nearly no additional weight.

Finally, after several consultations with fellow researchers, the author has chosen for the iRobot Create (see Section 3.3.2) platform available at the start of the research. It comes at low cost and acceptable physical dimensions and capabilities. One issue we expected is the manuverability-accuracy and moving over different terrain, which the author evaluates in Section 6. The iRobot Create was equipped with a TurtleCore expansion board (see Section 3.4.2) including an Overo Fire embedded computer (see Section 3.4.1), to allow high customisability in both hardware and software.

3.2 Basic Robots

There are many advanced robots out on the market. For example the PR2 (**P**ersonal **R**obot **2**) built and sold by Willow Garage. It is one of the more advanced robotics systems. Featuring two arms including gripers, a vast variety of sensors (eg. IMU, 5MP camera...), WiFi and even two on-board Xeon servers with 8 cores i7 and 25 GB RAM each, makes the PR2 attractive for research. However, such extended and complex systems come with a major downside: a price of $400,000.00 [21]. For the purpose of research in wireless networks other, more financially feasible solutions are a better alternative. Affordable solutions in the low cost sector (around $500) are more appropriate.

3.3 iRobot

The American based company iRobot was founded in 1990 by robotics researchers of the Massachusetts Institute of Technology. iRobot offers robots in mainly two categories: home robots available to the general public for cleaning work; and defence and security robots offered to military and civil defence forces [22].

3.3.1 Roomba Vacuum Cleaning Robots

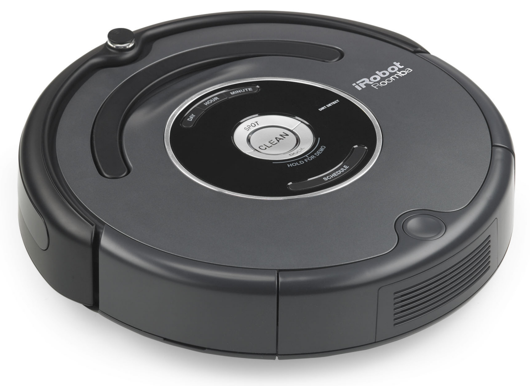

Figure 3.1: iRobot Roomba 560 Model[23]

The iRobot Roomba (shown in Figure 3.1) vacuum cleaning robots are a consumer product sold by the company iRobot. They are intended for autonomous vacuuming, featuring path finding and obstacle detection [24]. Models manufactured after October 2005 offer a serial command interface so developers can easily extend the robots functionality and behaviour [25].

The University of Bonn created a museum guide robot called **Robotinho** for the Deutsches Museum in Bonn. Its base consists of four Roomba robots placed side by side with a servo actuator, which gives it the possibility of moving omni-directionally [26].

3.3.2 iRobot Create

The iRobot Create is a trimmed down, developer only version of the Roomba 400 series product line. All the cleaning related parts have been removed and instead replaced by a cargo bay and a DB-25 connector to extend development possibilities [28].

The Create features the same serial command interface, called Open Interface, the consumer versions of Roomba offers. Although the wheels' actuators might offer higher speeds, the Open

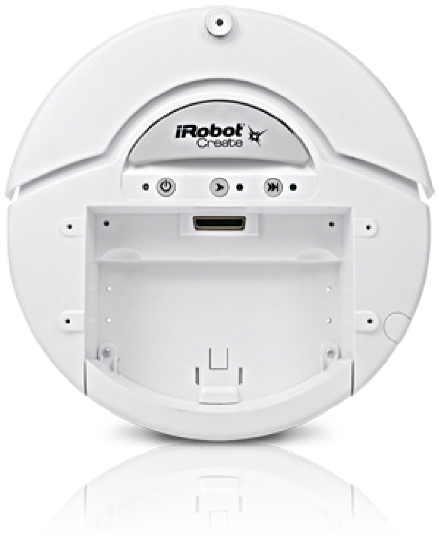

Figure 3.2: iRobot Create[27]

Interface sets the boundaries at 500mm/sec for forward and backward movements. Over the Open Interface the wheels can also be controlled individually [29].

A major advantage to the Roomba is the cargo bay for storing sensors and embedded computers. Another advantage is the existence of a dedicated embedded solution for steering and controlling the Create (see Section 3.4.2).

A somewhat big disadvantage of the Create was discovered after the robot was purchased and first driven. When driving the robot, it is very often a little bit off track. This occurs for example when starting to drive forward from standing still, in which case the two wheels do not start turning at exactly the same time. This results in a slight turn in the moment it starts driving. This would not be a big problem, if there was a gyroscope reporting back the real turning and driving values.

Unfortunately, the Create only features a low precision odometry controller. It uses the motor power information of the wheels to calculate the distance and directions, which the Create has moved. Also the motors, or just one of them, might report inaccurate values at some speeds while at others it would be very accurate [30].

3.4 Hardware Extensions

As the robots themselves only offer limited computational power, a developer requires either an embedded computing platform or a notebook allowing access to larger amounts of memory and faster processing units. Computational platforms also offer interfaces for further extensions, i.e., vision sensors or input devices.

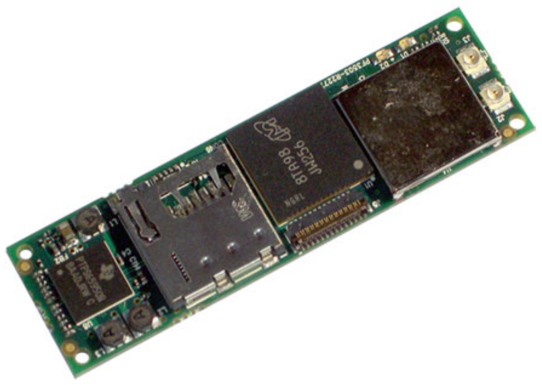
Figure 3.3: Overo Fire COM[31]

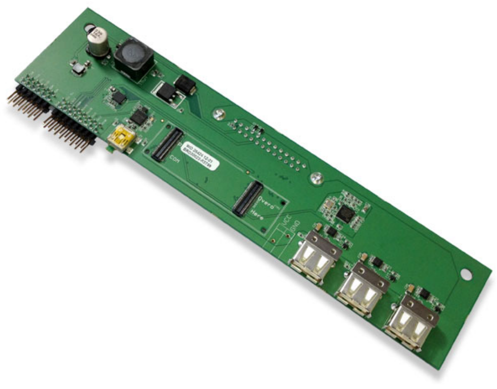
Figure 3.4: TurtleCore expansion board[32]

3.4.1 Gumstix Overo Fire Computer-on-Module

The Gumstix Computer-on-Module (COM) platform offers a small and embedded computing platform, powered through so called expansion boards. The particular model used for this thesis is the Overo Fire. It is equipped with a Texas Instruments ARM processor running on 720MHz, 512 MB of RAM and connectivity through Bluetooth and 802.11 b/g WiFi (see Figure 3.3). It can be run out of the box, when connected to an expansion board, with a pre-installed copy of Angstrom Linux on its ROM. If a microSD-card is inserted and contains all necessary files, the Overo Fire COM resorts to booting from the microSD-card, hence allowing custom images to be loaded [33].

The Overo uses a multi-stage booting architecture. The first stage bootloader, called x-Loader, is in charge of setting up the pin multiplexing, clock and memory initialization and the loading of the second stage bootloader. It is a minimized version of the second stage, built to run in the processors on-chip SRAM.

The U-Boot (the second stage bootloader) is responsible for setting the boot arguments and loading the kernel image. All these three stages (x-Loader, U-Boot and Kernel) are located in a separate file. So, for example trying out different Linux kernels is just a matter of downloading and copying a different kernel file to the microSD-card [34].

Advantages of Gumstix like embedded computers are their small size and lower costs. As a downside, they might need customized Linux images for lightweight operations, which might not be directly obtainable.

3.4.2 Gumstix TurtleCore Expansion Board

The TurtleCore expansion board offers the developer multiple USB ports, current and general purpose data pins on the board for adding sensors and other electronic elements. In combination with an Overo Fire COM, for WiFi connectivity, it offers the developers great freedom in developing remote steering and real-time controlling solutions. Another convenience is the direct power supply from the robot's battery via the DB-25 connector [35]. This way a developer only has one battery that has to be charged, instead of having an additional notebook battery.

3.5 Alternative Solutions

3.5.1 iClebo Kobuki

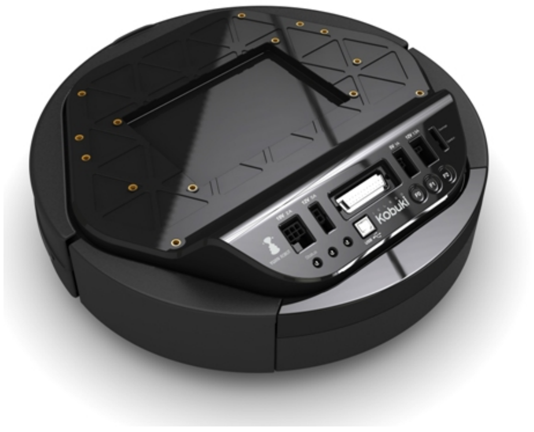

Figure 3.5: iClebo Kobuki[36]

The Korean company **Yujin Robot**, following the success of the iRobot Create, in 2013 started delivery of their new, relatively low-cost, robotics platform. It resembles the Create but offers several more accurate sensors (like the gyroscope) and easy to use power plugs for additional hardware (notebook, cameras, etc). All the additions and improvements come at the cost that the Kobuki does not feature a cargo bay.

With a maximum translational speed of 70 cm/s the Kobuki is 40% faster than the Create. It can turn with 180 deg/s with the limitation that the gyroscopes accuracy drops with speeds greater than 110 deg/s. On hard floors the Kobuki can carry up to 5 kg of payload, placed or mounted on top of its even top surface. Equipped with the small battery its expected time of operation tops at 3 hours, a large battery extends the operating time to around 7 hours.

From the basic safety sensors point of view the Create and Kobuki are similarly equipped:

- left, center and right bumper sensors
- left, center and right cliff sensors
- left and right wheel drop sensors

All the information stated inside this section can be found on the specifications web page of the Kobuki Yujin Robot [37].

3.5.2 Prebuilt Robot Systems

A pre-built, complete robot platform offers several advantages to self-crafted basic systems. First of all, these robots come pre-installed with all software to use it out-of-the-box and if not, there is a way to get pre-installed images of the system. The software packages for the turtlebot are available as open source via github [38] and thus can be easily acquired and modified to one's own needs.

Also, all the hardware combined is meant to work as-is. There is no need for tinkering with sensors, cameras and robots, until an acceptable interplay is found. Equipped with carrying platforms, these systems also offer a great way of extending them with additional sensors, input devices etc.

Of course, this all comes at a cost of larger dimensions (mostly with more height). Another downside is the price, when compared to a basic setup of a self-crafted solution. A TurtleBot 2 equipped with a Laptop, an ASUS Xtion 3D sensor, a docking station and additional battery comes in at around $1800 [39]. An iRobot Create [40] with a Gumstix Overo Fire COM [33] and a TurtleCore Expansion Board[35] costs around $530.

3.5.3 TurtleBot First and Second Generation

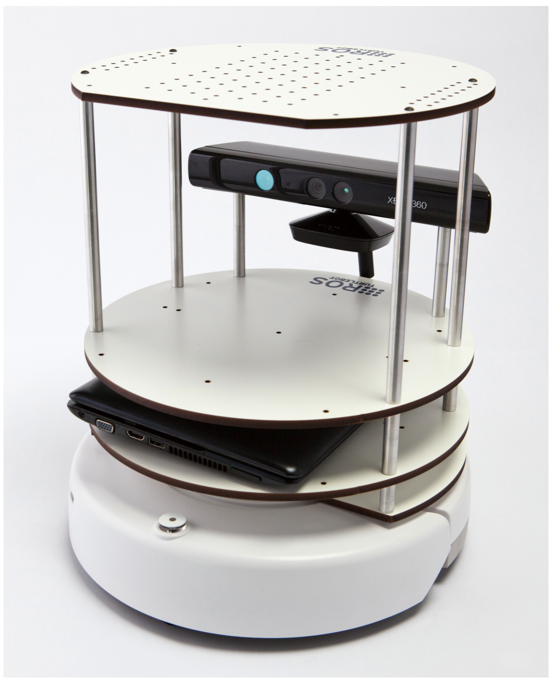
Figure 3.6: First generation TurtleBot[41]

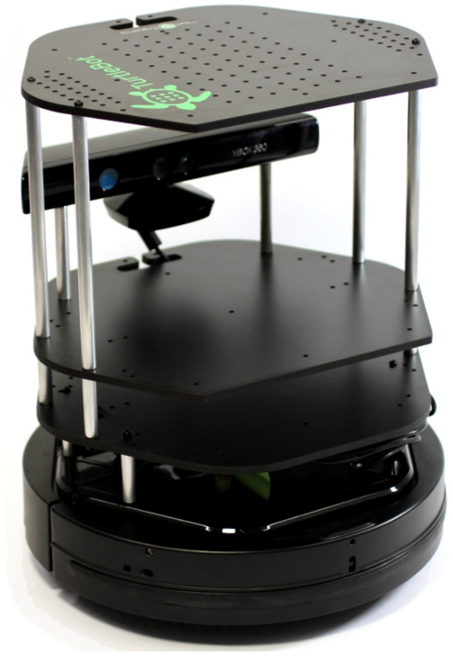
Figure 3.7: Second generation TurtleBot[42]

The TurtleBot is an extension to the basic iRobot Create. It is equipped with a netbook (Asus EeePc 1215n), a Microsoft Kinect camera (see Section 3.5.5), the TurtleBot power board (see Section 3.5.4) and trays built from laser cut plates [43].

The first generation TurtleBot is built up from an iRobot Create, the second generation uses the iClebo Kobuki as a robot base. Furthermore, there is the option to upgrade from the Kinect camera to the Asus Xtion [44]. A comparison of the Microsoft Kinect and Asus Xtion can be found in Section 3.5.5.

3.5.4 TurtleBot Power Board

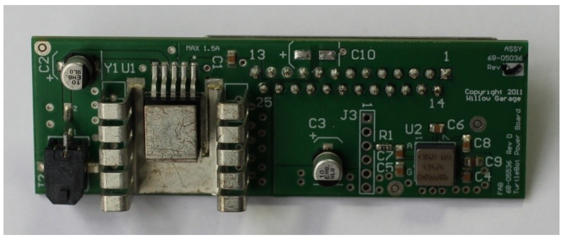

Figure 3.8: TurtleBot Power Board[45]

To compensate the inaccuracy of the odometry device built into the iRobot Create the TurtleBot Power Board can be connected to the DB-25 port inside the cargo bay. It offers a 12 volts power output for the Kinect system (see Section 3.5.5) as well as a gyroscope with 3 degrees of freedom and a resolution of 250 deg/s [46].

As this device occupies the DB-25 port inside the cargo bay of the Create, using it would restrain us to use a Gumstix COM with the TurtleCore expansion board.

3.5.5 Vision Sensors

A vision sensor enables the robot to scan its surroundings to estimate the current position and heading, in addition to its own sensors. Furthermore, the robot can map the surrounding area in case it is operated in an unknown environment.

Reading image and depth information from the sensor and processing them require a strong computational platform. As the Gumstix and other embedded devices lack of computational power, the developer would need to resort to a notebook.

Kinect

In June 2009 Microsoft announced and demonstrated its newest and state-of-the-art game controller. First known under the codename **Natal**, the **Kinect** controller was released in November 2010 [47]. The Kinect controller is fully packed with many sensors and other electronic devices [48] (shown in Figure 3.9), including:

- an RGB camera with a resolution of 1280x960 pixels and a frame rate of 30 fps;
- an infrared emitter and sensor for capturing a depth image of the surroundings;
- a microphone array consisting of four microphones to record audio and locating the source of it;

- a 3 axis accelerometer to determine the orientation of the Kinect in space.

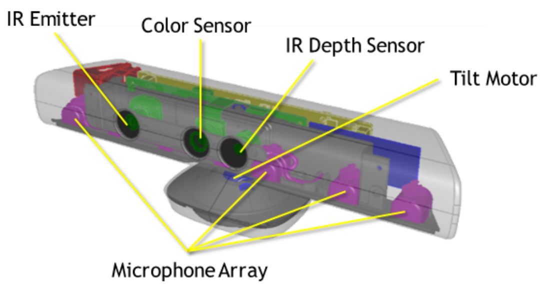

Figure 3.9: Schematic representation of the Kinect[49]

Asus Xtion

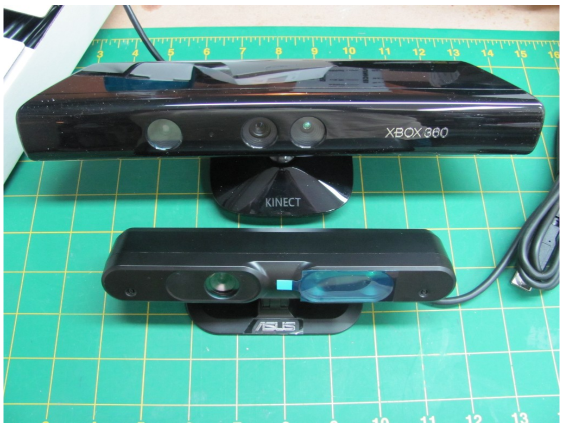

Figure 3.10: Size comparison between Kinect and Xtion[50]

Another system for recording RGB and depth imaging is produced by Asus and is shown in Figure 3.10. An obvious advantage of the Xtion to the Kinect is its size as seen in the figure. In terms of specifications, the Asus Xtion is somewhat similar to the Microsoft Kinect:

- an RGB camera with a resolution of 1280x1024 pixels;
- an infrared emitter and sensor with resolutions of 640x480 and 320x240 pixels and a framerate of 30 and 60 fps, respectively;
- two microphones for sound recording.

The Xtion platform comes bundled with its own SDK and supports writing programs in C++, C# and Java for Windows, Linux and Android [51]. A clear advantage of the Asus Xtion over the Microsoft Kinect is the standard USB plug. Whereas the Microsoft Kinect only offers a proprietary, customized USB plug, for which an adapter is required, the Asus Xtion can be plugged into any system that offers an USB port.

Chapter 4

Robot Software

4.1 General Overview

4.1.1 Operating System

From the operating system point of view, there is a wide variety of Linux distributions available to choose from. Without modification, the Gumstix Overo COM runs an embedded Linux called Angstrom, which is installed on the computers ROM chip [52]. Different operating systems can be booted on the Overo using the on-board MicroSD slot [53].

An organisation called Linaro develops open source software to bring better support for ARM-based computers to Linux. Linaro was founded by several big players in the electronics industry, including ARM, IBM, Samsung and TI. The companies within Linaro try to bring broader compatibility for open-source software running on ARM-based CPUs. The operating system images provided by this company are not directly an independent system but rather a renamed Ubuntu with ARM-specific patches and specifically built with the drivers for several embedded devices (PandaBoard, OrigenBoard) [54].

After some tinkering with the Linaro images for the Overo Fire COMs, the author decided not to use this Linux distribution for the following reasons:

- Most of the images were built with a GUI (even the pre-installed Angstrom image). Since our hardware runs on small capacities and has no video port this was not a good choice.
- The last available image for the Overo Fire was version 12.03 (built upon Ubuntu 11.10). So it lacked actuality.
- Many errors arose during the installation of the Robot Operating System (see Section 4.1.2) due to campatibility issues with Linaro.

After a detailed search for a feasible solution, the author chose for a self-crafted minimal Ubuntu with ARM optimizations and the minimum required software to be productive.

This is where a program called `debootstrap` comes into place. It offers the possibility to create a minimal root file system for many different CPU-architectures. More on `debootstrap` and the procedure of creating a root file system will follow in Section 5.1.1 [55].

Contribution

- Evaluating different Linux distributions and binary packages for use with the Gumstix platform;
- Building a basic and small Ubuntu image.

4.1.2 Robot Operating System

While the name can be a little misleading, the Robot Operating System (short: ROS) is not an operating system per se, but rather a complex open-source framework for robotics. It is being developed by Willow Garage, a company that also develops robotics hardware like the Personal Robot 2. Due to the open-source manner of ROS there are extensive additional packages for robotics, vision and many other applications available [56].

The following description about the inner workings of the Robot Operating System is based on the information from the technical overview [57].

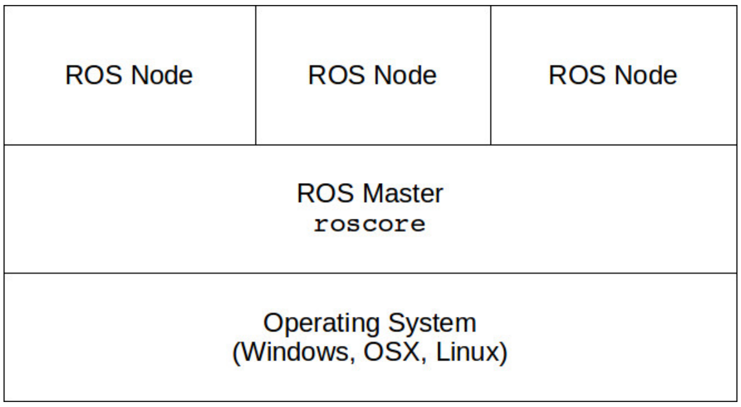

Figure 4.1: Architecture of a ROS system

This framework uses a modular architecture of many different programs. These programs are called **nodes**. A special case of node is the ROS **master**, which runs on top of the actual operating system (Windows, OSX or Linux). It operates as a central service to provide an environment for other nodes to comunicate with eachother (see Figure 4.1). Nodes can be distributed onto different machines, so calculation intensive tasks can run on high-performance computers while device drivers run on the robot itself. They are distributed alone or multiple nodes together in packages.

Nodes can publish and subscribe to **topics**. A topic describes a type of information a node shares in a standard form (e.g. steering commands or sensor data). The master acts as a broker between the different publishing and subscribing nodes. Formal communication between nodes happens through an HTML-based protocol called XMLRPC.

When a node sends a subscription package for a desired topic to the master node, the master returns the URI of the publisher and all communication after this point happens directly between the publisher and the subscriber. After an initial connect the XMLRPC protocol is dropped and the desired data is streamed over TCP to the subscriber. For a visual example of this process, have a look at Figure 4.2.

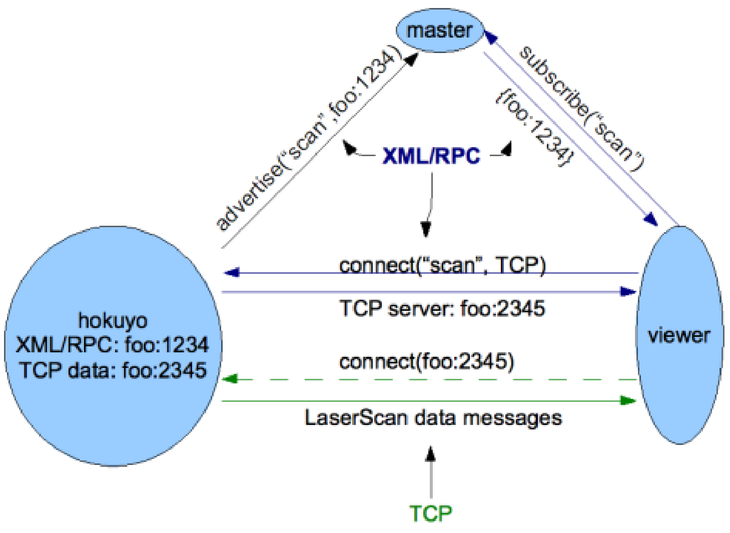

Figure 4.2: Example process of node communication[58]

The upside in using the iRobot Create coupled with ROS is that the packages written for the TurtleBot (see Section 3.5.3) can also be used for the steering and sensoric feedback.

In the course of writing this thesis, Willow Garage released a new version of ROS named **Groovy** in December of 2012 [59]. This new version introduced changes leading to backwards incompatibility. Especially the installation of ROS itself and also its packages have now a completely different procedure. The `turtlebot` package has undergone major refactoring towards a more modular approach. The driver for the iRobot Create was moved to a seperate package (`create_node`) and an additional package was constructed featuring the driver for the iClebo Kobuki.

Due to these changes, only the setup to use the implemented mobility models on the Create will be described in more detail.

Because of the rather unique hardware setup and only few basic tutorials on the installation of ROS, the author had to combine many online sources and his own knowledge of Linux to come up with a feasible setup routine on the Gumstix platform. The installation process in Chapter 5 describes the installation of version **Groovy**. A binary image of the previous **Fuerte** version is available on the enclosed DVD.

Contribution

- Research installation process of two different versions of ROS (including relevant packages);
- Compiling ROS on the Gumstix platform.

4.2 Mobility Model Implementations

During the work on this thesis, the author programmed several basic nodes for testing purposes (see Chapter 6) and two, more complex nodes with the implementation of the two mobility models described in Section 2.2 and 2.3.

The driver for the iRobot Create named `create_node` (`turtlebot_node` in ROS versions prior to groovy) subscribes to a topic called **Twist**. This topic holds two vectors; one for linear speed in meters per second and one for angular speeds in radian units. The implemented mobility models publish to the Twist topic and send the appropriate vectors for turning and driving according to the selected mobility model's properties.

All newly implemented nodes are distributed through the package `turtle_mover` on the enclosed DVD.

Contribution

- Programming test nodes for evaluation purposes;
- Implementing two nodes with the random waypoint and the constrained mobility model.

4.2.1 Random Walk

The implemented random walk mobility model (see Section 2.2) also includes some traits of the random waypoint model (see Section 2.1).

An important requirement towards the implemented model was a random distributed waiting period after each driving interval. Also, neither driving time nor driving distance should be constant. Listing 4.1 shows the methods that calculate the random distributed variables. As seen, the speed is normally distributed around half the maximum speed of the iRobot Create ($0.25m/s$) with a variance of $0.1m/s$. For the driving time a mean of $5s$ with a variance of $2s$ has been chosen. When the robot stops, it will wait for a normally distributed pause time around a mean of $10s$ with a variance of $2s$. The turning direction and the turning time are both uniformly distributed so that any driving angle between 0°and 360°are similarly possible.

Listing 4.1: Random distributions inside random_walk.py

```
def calc_waiting_time():
    return random.normalvariate(10,2)

def calc_turning_direction():
    return random.choice([1,-1])

def calc_turning_time():
    return random.uniform(0,4)

def calc_driving_speed():
    return random.normalvariate(0.5,0.2)

```

```
def calc_driving_time():
    return random.normalvariate(5,2)
```

4.2.2 Constrained Mobility

The constrained mobility model was implemented for two different use-cases. In the first case, the robot visits the vertices using the uniformly distributed `choice` algorithm from the Python `random` module. This way, it mimics the movements of a human seeking for a room or location in unknown surroundings. The second possibility lets the user give the program a specified path in a vertex-by-vertex manner. So it is possible to have a re-run of a previous experiment where the vertices were visited randomly and compare the results.

The graph specification for the constrained mobility model happens with two different files named `graph.txt` and `coordinates.txt`. The `graph.txt` file defines the edges of the graph in the format `<current vertex>:<neighbour vertex 1>,<neighbour vertex 2>`. An example of this can be seen in listing 4.2. The file `coordinates.txt` contains the informations of the coordinates of each vertex, specified as `<current vertex>:<x-coordinate in meters>,<y-coordinate in meters>`. For a practical example, see listing 4.3.

Listing 4.2: Example content of graph.txt

```
A:B
B:A,C,D
C:B,E
D:B,E
E:D,F
F:E
```

Listing 4.3: Example content of coordinates.txt

```
A:0,0
B:1,0
C:1,1
D:2,0
E:2,1
F:2,2
```

The specification of the driving speed during the execution is given with the file `speed.txt`. The speed is entered in units of m/s and the maximum possible value is $0.5m/s$, limited by the maximum speed of the iRobot Create.

As stated before, the program can be given an optional file containing a predefined path. The file, named `path.txt`, contains a vertex per line and the path is defined in a vertex-by-vertex manner. Path finding to a remote vertex has not been implemented. Keep in mind that the robot will always start at vertex `A`, looking towards the positive x-axis.

To simplify computation and due to accuracy limitations, only right angled edges are allowed in the graph. The author's implementation uses two variables to keep track of the facing direction

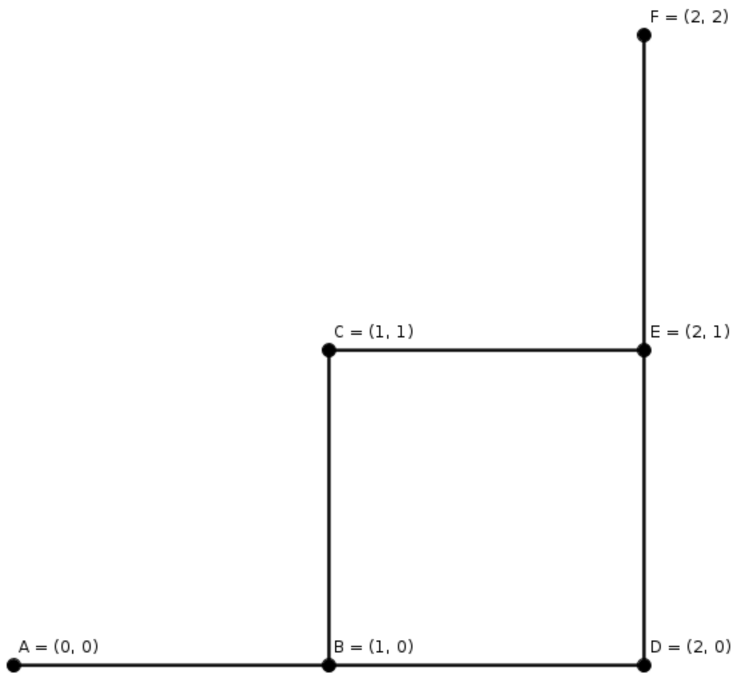

Figure 4.3: Visualisation of the example graph

of the robot and to calculate the required turning angles. The variable `facing_coordinates` can either contain a value of `x` or `y`. It tells whether the robot is facing towards the x- or y-axis in the coordinate system. The second variable, `facing_direction`, holds a value of either `-1` or `1`, describing if the robot is looking towards the negative or the positive area of the facing axis.

4.3 Extensibility

With a limited time budget, only basic functionality for the mobility models was implemented. Extensibility is ensured with the usage of the Robot Operating System and with an easy to learn language such as Python as the programming language. With basic programming skills in Python and knowledge of the Robot Operating System, one could extend the implemented mobility models in many different ways.

A possibility is to introduce a different pause time scheme to the constrained mobility model. The pause time could be randomly distributed at each stop, which includes the need of logging them and implementing a way to reintroduce them in a predefined path run. Also each vertex could have its own individual pause time, which would be easier to implement and would not pose an issue with predefined re-runs.

Another way to extend the constrained mobility model is to add weight values, representing traversing probabilities, to the edges of the graph. Resulting in a non uniformly distributed chance that the robot will visit a specific neighbouring vertex. This scenario corresponds to, e.g., a shopping mall where male and female shoppers would not frequent the same stores with the same probability.

Chapter 5

Installation Process

5.1 Setup

All instructions in the following subsections are expected to work on Ubuntu (10.04 Lucid Lynx or newer) and similar flavours of Linux (e.g. Mint). The author expects the reader to have an understanding of these platforms and also the reader must bring the ability to install software by him-/herself (in case a basic program out of the scope of the following guide is not present on his/her system). Furthermore, a `$` or > sign at the beginning of a listings-line of the Bash-shell signals a user issued command and lines without a preceding sign stand for output.

The basic setup instructions described on the following pages are the results of extensive online research. The creation of a root file system[55] and the serial console[60] is taken from the Ubuntu wiki, the creation of a bootable SD-card[53] has its origin on the Gumstix developer pages. The rest of knowledge brought to paper originates from the research blog of a computer science student[61] and the author's personal knowledge of Linux systems.

5.1.1 Creating a Root File System

The console program `debootstrap` can be used to build custom root file systems for a variety of different CPU architectures. The installation is initiated with issuing the following command.

```
$ sudo apt-get install debootstrap
```

Listing 5.1: Installing debootstrap

The bootstrapping process for the new file system consists of two stages. For the first stage the `debootstrap` command with some additional arguments is entered into the shell. The `--arch armel` argument tells the program to use binaries built for ARM processors, `--include=[...]` adds specific additional software and with `--keyring=[...]` the Ubuntu keyring is used. To see the progress, `--verbose` prints out additional informations. `--foreign` tells `debootstrap` only to do the first stage of the bootstrapping and `--variant=buildd` adds software-building specific packages, as this system will be used to compile programs. Finally, `precise` is the project name of the Ubuntu version to be used (here Ubuntu 12.04 LTS) and `rootfs` specifies a folder where the file system will be written to.

```
$ sudo debootstrap \
>     --arch armel \
>     --include=sudo,wget,net-tools,wireless-tools,vim,openssh-server
   ,lsb-release \
>     --keyring=/usr/share/keyrings/ubuntu-archive-keyring.gpg \
>     --verbose \
>     --foreign \
>     --variant=buildd \
>     precise \
>     rootfs
```

Listing 5.2: First stage execution of the bootstrapping

This already concludes the first stage. For the second stage, the QEMU virtualisation software will be used. The easiest way in this particular situation is to use syscall emulation, meaning machine-codes will be translated and system calls remapped from the virtual machine to the kernel of the actual machine.

Install the syscall emulation functionality of qemu.

```
$ sudo apt-get install qemu-kvm-extras-static
```

Listing 5.3: Installing the syscall emulation

Check that QEMU registered the ARM format properly inside the binfmt-support package with:

```
$ cat /proc/sys/fs/binfmt_msisc/qemu-arm
enabled
interpreter /usr/bin/qemu-arm-static
flags: OC
offset 0
magic 7f454c4601010100000000000000000002002800
mask ffffffffffffff00fffffffffffffffffeffffff
```

Listing 5.4: Check the status of qemu-arm

Since the `chroot` program (**ch**ange **root**) will be looking for the ARM-interpreter inside the `rootfs` folder, the interpreter needs to be copied over.

```
$ sudo cp /usr/bin/qemu-arm-static rootfs/usr/bin/
```

Listing 5.5: Copy the ARM interpreter to the root file system

Next, use `chroot` to start the (not yet fully finished) root file system inside a virtual environment and trigger the second stage.

```
$ sudo chroot rootfs /bin/bash
$ /debootstrap/debootstrap --second-stage
```

Listing 5.6: Trigger the second stage

After the successful completion of the second stage, close the virtual machine with the `exit` command.

5.1.2 Creating a bootable microSD-card

Preparation

Insert the microSD, using an SD-card or USB adapter, into a computer running your choice of Linux. The microSD will most likely show up as /dev/sdX (where X can be any letter from b to z). On `bash` use `df` or `mount` to show the mounted partitions and verify the microSDs device name. If your system automounts plugged in storage devices unmount it first, with ”n” being the partition number.

```
$ sudo umount /dev/sdXn
```

Listing 5.7: Unmount eventually mounted partitions of the microSD

Calculating Card Size

To ensure accessibility of the data on the microSD, the required number of cylinders for your particular card has to be calculated using the `fdisk` command.

```
$ sudo fdisk -l /dev/sdX

Disk /dev/sdX: 3974 MB, 3974103040 bytes
70 heads, 5 sectors/track, 22176 cylinders, total 7761920 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdX1            8192     7761919     3876864    b  W95 FAT32
```

Listing 5.8: Get the exact bytes count of the microSD

Calculate the count of cylinders by dividing the card size in bytes by 255 heads, 63 sectors and 512 bytes per sector and rounding down to the next integer.

$$\left\lfloor \frac{card\ size}{255\ heads \times 63\ sectors \times 512\ bytes} \right\rfloor = cylinders$$

Partitioning

There are a minimum of two partitions required to be able to boot the Overo from the microSD: a FAT partition for the bootloader and an ext3 Linux partition where the root file system created in Section 5.1.1 will be stored.

First of all, any existing partition data on the microSD needs to be deleted.

```
$ sudo dd if =/dev/zero of=/dev/sdX bs=1024 count=1024
1024+0 records in
1024+0 records out
1048576 bytes (1.0 MB) copied, 0.28741 s, 3.6 MB/s
```

Listing 5.9: Zero out the first MB

The required partitioning can be done using `sfdisk`:

```
$ sudo sfdisk --force -D -uS -H 255 -S 63 -C <cylinders> /dev/sdX
```

Listing 5.10: Creating the partition table

For <cylinders>insert the calculated cylinder count from above without the brackets.

At the first prompt from the program, type **128,130944,0x0C,*** and press the enter key. This will create the first, 64MB bootable FAT partition. At the second prompt insert **131072,,,-** and press enter another three times to create the second (Linux) partition and complete the partitioning process. When asked to write the new partition table, type **y** to confirm.

Example output (with user-input marked bold):

```
Checking that no-one is using this disk right now ...
OK

Disk /dev/sdX: 483 cylinders, 255 heads, 63 sectors/track

sfdisk: ERROR: sector 0 does not have an msdos signature
 /dev/sdX: unrecognized partition table type
Old situation:
No partitions found
Input in the following format; absent fields get a default value.
<start> <size> <type [E,S,L,X,hex]> <bootable [-,*]> <c,h,s> <c,h,s>
Usually you only need to specify <start> and <size> (and perhaps <
   type>).

/dev/sdX1 :128,1309344,0x0C,*
/dev/sdX1   *       128    131071     130944   c  W95 FAT32 (LBA)
/dev/sdX2 :131072,,,-
/dev/sdX2        131072   7761919    7630848  83  Linux
/dev/sdX3 :
/dev/sdX3             1       127        127  83  Linux
/dev/sdX4 :
/dev/sdX4             0         -          0   0  Empty
New situation:
Units = sectors of 512 bytes, counting from 0

   Device Boot    Start       End   #sectors  Id  System
/dev/sdX1   *       128    131071     130944   c  W95 FAT32 (LBA)
/dev/sdX2        131072   7761919    7630848  83  Linux
/dev/sdX3             1       127        127  83  Linux
/dev/sdX4             0         -          0   0  Empty
```

```
Warning: partition 1 does not end at a cylinder boundary
Warning: partition 2 does not start at a cylinder boundary
Warning: partition 2 does not end at a cylinder boundary
Warning: partition 3 does not end at a cylinder boundary
end of partition 2 has impossible value for cylinders: 483 (should be
     in 0-482)
Do you want to write this to disk? [ynq] y
Successfully wrote the new partition table

Re-reading the partition table ...

If you created or changed a DOS partition, /dev/foo7, say, then use
   dd(1)
to zero the first 512 bytes:  dd if=/dev/zero of=/dev/foo7 bs=512
   count=1
(See fdisk(8).)
```

Listing 5.11: Example of the partitioning dialog

Formatting

After the creation of the partition table, the partitions need to be formatted according to their specific file system.

Set up the first partition to be formatted as a FAT file system with the label ”boot”.

```
$ sudo mkfs.vfat -F 32 /dev/sdX1 -n boot
mkfs.vfat 3.0.13 (30 Jun 2012)
```

Listing 5.12: Format first partition as FAT

To format the second partition for Linux with the label ”rootfs”, issue the following command:

```
$ sudo mke2fs -j -L rootfs /dev/sdX2
mke2fs 1.42.5 (29-Jul-2012)
[Output cut]
```

Listing 5.13: Format second partition als ext3

5.1.3 Writing all required files to microSD

Create two mountpoints and mount the two partitions into your system.

```
$ sudo mkdir -/media/{boot,rootfs}
$ sudo mount -t vfat /dev/sdX1 /media/robot/
$ sudo mount -t ext3 /dev/sdX2 /media/rootfs/
```

Listing 5.14: Mount the microSD

Before proceeding, make sure you have all of the following files:

- a root file system (in our case self crafted, see Section 5.1.1)
- a kernel binary image named 'uImage'[1]
- a u-boot bootloader binary image 'u-boot.img'[2]
- a x-loader binary image 'MLO'[3]

If all of the required files are present they can be copied over to the microSD. Navigate the shell to the location where these files are located, copy them all over and issue the `sync` command so everything will be written to the disk.

```
$ sudo cp MLO /media/boot/MLO
$ sudo cp u-boot.img /media/u-boot/u-boot.img
$ sudo cp uImage /media/boot/uImage
$ sudo cp -rf rootfs/* /media/rootfs/
```

Listing 5.15: Copying everything over

5.1.4 Additional Configurations

Copy Hardware-specific Files

Download the file `gumstix-console-image-overo-20120930223122.rootfs.tar.bz2` from `http://cumulus.gumstix.org/images/angstrom/developer/yocto/` and extract it to a temporary folder.

```
$ sudo tar xjf gumstix-console-image-overo-20120930223122.rootfs.tar.
   bz2
```

Listing 5.16: Untar the Gumstix image

Change into the extracted folder and copy the kernel modules and drivers firmware to the microSD.

```
$ sudo cp -rf lib/firmware/ /media/rootfs/lib/
$ sudo cp -rf lib/modules/ /media/rootfs/lib/
```

Listing 5.17: Copying kernel modules and firmware

[1]downloaded from `http://cumulus.gumstix.org/images/angstrom/developer/yocto/` on 02.02.2013

[2]downloaded from `http://cumulus.gumstix.org/images/angstrom/developer/yocto/` on 02.02.2013

[3]downloaded from `http://cumulus.gumstix.org/images/angstrom/developer/yocto/` on 02.02.2013

Hostname

Create a file inside `/media/rootfs/etc` called `hostname` and save the desired name of the machine into it.

```
robot
```

Listing 5.18: Contents of /etc/hostname

Also add the name to the `/media/rootfs/etc/hosts` file as an alias for localhost.

```
127.0.0.1   localhost   robot
```

Listing 5.19: Changed line inside /etc/hosts

Serial Console

To be able to log in over the serial port of a typical Gumstix expansion board, the `getty` process needs to be configured accordingly.

Create a file called `/etc/init/ttyO2.conf` with the following content:

```
# ttyO2 - getty
#
# This service maintains a getty on ttyO2 from the point the system
# is started until it is shut down again.

start on stopped rc or RUNLEVEL=[2345]
stop on runlevel [!2345]

respawn
exec /sbin/getty -L 115200 ttyO2 vt102
```

Listing 5.20: Contents of /etc/init/ttyO2.conf

WiFi Connectivity

For ease of operation and for an easy setup, this project uses an unprotected WiFi network. Turn to `www.google.com` to find out how to set up a connection to a WPA/WPA2 or WEP secured network. Append the following lines to `/etc/network/interfaces` with the according values of <static-ip>, <gateway-ip>and <network-name>for your network:

```
auto wlan0
iface wlan0 inet static
        address <static-ip>
        netmask 255.255.255.0
        gateway <gateway-ip>
        wireless-essid <network-name>
```

Listing 5.21: Added contents of /etc/network/interfaces

Enable Root Account for Login

Since there is no ordinary user set up until now, the root account can be enabled to have a valid login on the first boot-up. Open up the `shadow` file on the microSD (found at `/mount/rootfs/etc/shadow`) and delete the content between the first and second colon on the line of the root user. The result should look similar to the next listing.

```
root::15713:0:99999:7:::
```

Listing 5.22: Adjusted line inside /etc/shadow

First Bootup

For the final configurations of the system, the easiest way to access the Gumstix is through the serial console (configured in Section 5.1.4). Connect a mini-B to standard-A USB-cable to the console port of the expansion board and point your preferred terminal program (eg. `kermit`, `minicom`, `screen`) to the serial device on the computer (`/dev/ttyO0` or `/dev/ttyUSB0`). The connection parameters include: no flow control, 115200 bps and 8N1 parity [62].

Now connect the TurtleCore expansion board to the iRobot Create and wait for the system to finish its boot process. At the login prompt enter `root` and login without a password.

User Management

The Linux-savvy reader may already be alarmed that we are working on the root account and have not yet setup a standard user. We are going to fix that right now.

```
$ adduser <username>
Adding user '<username>'...
Adding new group '<username>' (1001).
Adding new user '<username>' (1001) with group '<username>'.
Creating home directory '/home/<username>'.
Copying files from '/etc/skel'
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for <username>
Enter the new value, or press ENTER for the default
Full Name []:
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [y/N] y
```

Listing 5.23: User creation dialog

The new user also needs to be added to the sudoers group, so we do not need the real root account for administrative stuff like installing software and so on.

```
$ adduser <username> sudo
```

Listing 5.24: Adding a user to the sudoers group

After adding the new user and adding it to the sudoers group, create a password to the root account and logout to use your newly created user.

```
$ passwd
```

Listing 5.25: Creating a password

DNS

On some networks the Gumstix with the current configuration does not get any nameservers, so we are going to add the free and public nameservers from `www.google.com` to the `/etc/resolv.conf` file.

```
nameserver      8.8.8.8
nameserver      8.8.4.4
```

Listing 5.26: Contents of /etc/resolv.conf

Change rights for /tmp

During the bootstrapping process of the root file system, the rights for the temporary folder `/tmp` were not properly set and only the root user can write to it. Since the installation scripts of ROS(see Section 5.1.5) require write permissions to `/tmp`, the permissions need to be adjusted. Issue the following command to add write access:

```
$ sudo chmod a+w /tmp
```

Listing 5.27: Give write access for /tmp to all users

Updating

Next, Ubuntu needs to be told from where to fetch new programs or system updates. This is specified inside the `/etc/apt/source.list` file.

```
deb http://ports.ubuntu.com/ubuntu-ports precise main
deb http://ports.ubuntu.com/ubuntu-ports precise-updates main
deb http://ports.ubuntu.com/ubuntu-ports precise-security main
deb http://ports.ubuntu.com/ubuntu-ports precise universe multiverse
deb http://ports.ubuntu.com/ubuntu-ports precise-updates universe
    multiverse
deb http://ports.ubuntu.com/ubuntu-ports precise-security universe
    multiverse
```

Listing 5.28: Contents of /etc/apt/sources.list

To finish the basic system setup, issue the `update` and `upgrade` commands of `apt-get`.

```
$ sudo apt-get update && sudo apt-get upgrade
```

Listing 5.29: Updating the system

You have now a fully configured and working operating system on your Gumstix COM.

5.1.5 Installing ROS

In order to sucessfully install ROS on the Gumstix Overo Fire combining several sources was necessary. First, describing the installation from source[63] adding the ROS repository to `apt`[64] installing Collada and ASSIMP for the Raspberry Pi (and other ARM computers)[65] creating an overlay workspace[66] and finding the `create_node` package repository [67].

The reader may also visit the general installation page of ROS [68] and check if there are new tutorials on installing the Robot Operating System on ARM systems.

Prerequisites

First add the software sources from ROS to the apt sources list.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise
   main" > /etc/apt/sources.list.d/ros-latest.list'
```

Listing 5.30: Adding ROS software sources

After that these software sources need to be verified with the key from the ROS repositories.

```
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
OK
```

Listing 5.31: Adding the ROS repository key

With the key added, update the sources so that ROS specific packages can be installed.

```
$ sudo apt-get update
```

Listing 5.32: Update the software sources

Install the dependency and workspace administration tools to your system.

```
$ sudo apt-get install python-rosdep python-wstool
```

Listing 5.33: Install ROS administration tools

Initialize the ROS dependency system which checks dependencies between ROS packages.

```
$ sudo rosdep init
Wrote /etc/ros/rosdep/sources.list.d/20-default.list
Recommended: please run

    rosdep update

```

```
$ rosdep update
reading in sources list data from /etc/ros/rosdep/sources.list.d
Hit https://github.com/ros/rosdistro/raw/master/rosdep/osx-homebrew.
   yaml
Hit https://github.com/ros/rosdistro/raw/master/rosdep/gentoo.yaml
Hit https://github.com/ros/rosdistro/raw/master/rosdep/base.yaml
Hit https://github.com/ros/rosdistro/raw/master/rosdep/python.yaml
Hit https://github.com/ros/rosdistro/raw/master/rosdep/ruby.yaml
Hit https://github.com/ros/rosdistro/raw/master/releases/fuerte.yaml
Hit https://github.com/ros/rosdistro/raw/master/releases/groovy.yaml
updated cache in /home/robot/.ros/rosdep/sources.cache
```

Listing 5.34: Initialize dependency system

Download the core ROS packages

Create a catkin workspace folder, catkin is the build system for the core packages.

```
$ mkdir ~/ros_catkin_ws
$ cd ros_catkin_ws/
```

Listing 5.35: Create catkin workspace folder

Fetch the core packages for a robot system using `wstool`, the argument `-j1` tells the program only to download one package at a time. Higher values can cause errors with the Gumstix.

```
$ wstool init src -j1 http://packages.ros.org/web/rosinstall/generate
   /raw/groovy/robot
Using initial elements from: http://packages.ros.org/web/rosinstall/
   generate/raw/groovy/robot
Writing /home/robot/ros_catkin_ws/src/.rosinstall
[Output cut]

update complete.
```

Listing 5.36: Fetch core packages

Installing Collada and ASSIMP

Because `collada-dom-dev` is not available for the armel architecture over `apt` it needs to be built by hand. First create the folder `/opt/collada`, then download the sources from `sourceforge.net`, compile and install them.

```
$ sudo apt-get install libxml2-dev
$ sudo mkdir /opt/collada
$ sudo chown <username> /opt/collada
$ cd /opt/collada
$ wget http://sourceforge.net/projects/collada-dom/files/latest/
   download -O collada.tgz
```

```
$ tar -xf collada*; rm collada*tgz; cd collada*; mkdir build; cd
   build
$ cmake .. ; make -j1; sudo make install
```

Listing 5.37: Installing Collada

The compilation and installation takes up some time because of the limited hardware capabilities of the Gumstix.

Collada relays on ASSIMP and the standard binary causes the build process inside ROS to fail, so we are going to install this package manually as well.

```
$ mkdir /opt/ros/assimp
$ cd /opt/ros/assimp
$ wget http://sourceforge.net/projects/assimp/files/assimp-3.0/assimp
   --3.0.1270-source-only.zip/download -O assimp.zip
$ unzip assimp*; rm assimp*zip*; cd assimp*; mkdir build; cd build
$ cmake ..; make -j1; sudo make install
```

Listing 5.38: Installing ASSIMP

Afterwards remove the dependency declaration of `collada-dom` inside `collada_urdf/package.xml`, `collada_parser/package.xml` and `robot_model/package.xml` in /home/robot/ros_catkin_ws/src/. This is due to the fact that `rosdep` can not notice dependencies installed outside of `apt`.

Build the catkin Workspace

Next resolve all still unmet dependencies with `rosdep`. The argument `--from-paths src` specifies to install all system dependencies of the packages in the `src` folder and `--ignore-src` specifies that we are building the packages inside `src` by ourselves.

```
$ rosdep install --from-paths src --ignore-src --rosdistro groovy -y
[Output cut]
#All required rosdeps installed successfully
```

Listing 5.39: Installing system dependencies for ROS

After `rosdep` has finished installing the dependencies, the catkin workspace needs to be installed. This might take some time (up to several hours on the Gumstix): go and grab some coffee.

```
$ ./src/catkin/bin/catkin_make_isolated --install
Base path: /home/robot/ros_catkin_ws
Source space: /home/robot/ros_catkin_ws/src
Build space: /home/robot/ros_catkin_ws/build_isolated
Devel space: /home/robot/ros_catkin_ws/devel_isolated
Install space: /home/robot/ros_catkin_ws/install_isolated
[Output cut]
```

Listing 5.40: Compile and install all the robot core packages

To enable the bash to know the ROS commands, `source` the `setup.bash` file generated by the catkin workspace installation.

```
$ source ~/ros_catkin_ws/install_isolated/setup.bash
```

Listing 5.41: Source the setup.bash file

Create an Overlay Workspace

Make a separate folder to have a workspace to build existing third-party and your own packages and initialize it with `rosws`.

```
$ mkdir ~/ros_ws
$ cd ~/ros_ws/
$ rosws init . ~/ros_catkin_ws/install_isolated/
Using ROS_ROOT: /home/robot/ros_catkin_ws/install_isolated
Writing /home/robot/ros_ws/.rosinstall
(Over-)Writing setup.sh, setup.bash, and setup.zsh in /home/robot/
   ros_ws

rosinstall update complete.

Type 'source /home/robot/ros_ws/setup.bash' to change into this
   environment. Add that source command to the bottom of your ~/.
   bashrc to set it up every time you log in.

If you are not using bash please see http://www.ros.org/wiki/
   rosinstall/NonBashShells
$ source ~/ros_ws/setup.bash
```

Listing 5.42: Create separate workspace

Afterwards, merge the components of the ROS variant chosen before and download them to the overlay workspace.

```
$ rosws merge http://packages.ros.org/web/rosinstall/generate/raw/
   groovy/robot
[Output cut]
update complete.
$ rosws update -j1
```

Listing 5.43: Merge the robot variant to your workspace

Build the ROS Stack

Finally, `source` the newly created `setup.bash` in the overlay workspace and build all packages using `rosmake`.

```
$ source ~/ros_ws/setup.bash
$ rosmake -a
[ rosmake ] rosmake starting...
[ rosmake ] Building all packages
[Output cut]
```

Listing 5.44: Building all packages

5.1.6 Adding TurtleBot Packages to ROS

If ROS was installed as the `robot` variant, all dependencies for the `turtlebot_create` package are already met. It is sufficient to just directly get the package from `github.com`.

```
$ wstool set turtelbot_create --git https://github.com/turtlebot/
    turtlebot_create.git

     Add new elements:
  turtlebot_create      git  https://github.com/turtlebot/
     turtlebot_create.git

Continue: (y)es, (n)o: @\textbf{y}@
Overwriting /home/robot/ros_ws/.rosinstall

Do not forget to do ...
$ source /home/robot/ros_ws/setup.sh
... in every open terminal.
Config changed, remember to run 'rosws update turtlebot_create' to
    update the folder from git
$ wstool update turtlebot_create
[turtlebot_create] Updating /home/robot/ros_ws/turtlebot_create
[turtlebot_create] Done.
```

Listing 5.45: Adding turtlebot_create package

Next, let `rosdep` check for any system dependencies on `create_node`, as this subpackage is responsible for communicating with the Create, and install them, if any.

```
$ rosdep install create_node
```

Listing 5.46: Check for system dependencies on create_node

After all system dependencies were resolved, build the package.

```
$ rosmake create_node
[ rosmake ] rosmake starting...
[ rosmake ] Building ['create_node']
[ rosmake ] Packages requested are: ['create_node']

[Output cut]
```

```
[ rosmake ] Results:
[ rosmake ] Built 43 packages with 0 failures.
[ rosmake ] Summary output to directory
```

Listing 5.47: Build create_node

5.1.7 turtle_mover

To install the `turtle_mover` package developed in the course of this thesis, simply clone the package from a versioning server (via git, mercurial or svn), if provided. Otherwise, just copy the folder `turtle_mover` over to your ROS workspace.

In case the reader wants to use a ROS version before Groovy Galapagos, adjust the dependency in the `manifest.xml` file from `create_node` to `turtlebot_node`.

5.1.8 Android Phone

With the development of version **groovy**, Willow Garage is also reorganizing its Android applications for steering and controlling ROS enabled robots [69].

The following statements are valid under the pretence that the overall workflow with the ROS Android apps will not change too severely and the apps will only be redesigned but not renamed.

Provided the reader managed to install the full `turtlebot` package, there are two apps from the Google Play Store needed to drive the Create with the smartphone:

- ROS Application Chooser[70]
- Android Teleop[71]

First of all, the phone needs to be connected to the same wireless network the robot uses. When the robot is running and enlisted into the app with its IP, the **ROS Application Chooser** should list it as the TurtleBot. On selecting the robot, the app shows different applications the TurtleBot can run. Select **Teleop**, this will open the app **Android Teleop** and the robot will signalize he is ready with turning off all three LEDs.

5.2 Using an Implemented Mobility Model

The easiest way to run the implemented mobility models is to use the program `screen` which emulates several terminal sessions in one. For the basic controls, `Ctrl-A` followed by `c` creates a new terminal session, change the session with `Ctrl-A` followed by the number of the session and for exiting a session just issue the `exit` command.

Because of the nature of the Robot Operating System, there are multiple programs that need to be started. First of all, the core of the ROS framework needs to be run with the `roscore` command.

```
$ roscore
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
[...]
```

Listing 5.48: Starting of roscore

In a new session, the connection to the robot needs to be specified, `rosparam set create_node/port /dev/ttyO0` and the node for communicating with the robot needs to be started with `rosrun turtlebot_create/create_node create_node.py`.

```
$ rosparam set create_node/port /dev/ttyO0
$ rosrun turtlebot_create/create_node create_node.py
```

Listing 5.49: Changing port and starting the robot's driver

The next step, in a new `screen` terminal, is starting the desired mobility model. First navigate to the folder the parameter files lay (in case of the constrained mobility model) or where the log of random mobility should go. Afterwards issue `rosrun turtle_mover <model>.py` and have fun.

```
$ rosrun turtle_mover random_walk.py
```

Listing 5.50: Starting the random walk model

To stop the execution of a model, press `Ctrl-C` and the program will halt.

5.3 Creating Network Traffic

While steering the robot with a mobile phone, wifi traffic is created through the communication between the robot and the mobile phone. This enables a wireless test bed to pick up the robots signals. If the robot is run using an implemented moblity model there is little to no traffic. In order to create network traffic, several possibilities to create network traffic from the bash shell are shown in the following listing.

```
$ while true; do echo "something"; sleep 0.1; done
$ watch -n 0.1 echo "something"
$ ping x.x.x.x
```

Listing 5.51: Creating network traffic with bash one-liners

Chapter 6

Evaluation

The purpose of the conducted work was to offer a mobility platform to a WSN test bed. The requirements towards a mobile sensor node are mainly two: the ability to follow a predefined path and to connect to the test bed via the chosen radio technology.

After the successful installation of the robotic system and the finished implementation of the mobility models, the evaluation of the robot's accuracy came into focus. The software can only ensure the proper processing of the given mobility model but it cannot perform the tasks of execution and provide reliable signal strength measurement for localisation without accurate hardware.

The experiments were arranged in two sets. The first set, containing Scenarios I to III, for the purpose of hardware accuracy testing and the second, with Scenario V, for connectivity. Scenario IV falls into both sets since driving accuracy as well as connectivity data was recorded.

The first set of experiments began with Scenario I, which examines the ability of the robot to accurately move forward by a certain number meters, i.e., distance accuracy. In a later experiment (Scenario II), the ability of the robot to drive a straight line was examined, i.e., angular accuracy. Scenario II was conducted because of the observations of the author during initial driving experiments. The observation was that the two wheels, which are powered by two individual actuators, did not respond simultaneously to a driving command. This behaviour resulted the robot to be off by some degrees from the intended straight line especially when driving at higher speeds. A later test (Scenario III) was done to see the effect of the accumulation of the previous encountered deflexions.

The second set of experiments included the validation that the robot can be picked up by the wireless test bed. One experiment included steering the robot via commands sent over wifi from a cellular phone (Scenario V). In another experiment, the robot was run independently with an implemented mobility model (Scenario IV).

6.1 Scenario Descriptions

There are many different error sources with a physical robot running on physical hardware compared to a simulated model. The robot needs to travel pre-defined distances to some degree of accuracy; also it needs to be able to follow a straight line. Additionally, inaccuracy in turning angles might introduce accumulated errors over multiple turns, which could result in unacceptable path deviations.

In order to evaluate the impact of external, hardware related factors on the performance of the mobility models testing, we conducted a set of experiments, outlined in Table 6.1.

Scenario	Evaluated parameter	Description	Findings
I	distance [m]	6.1.1	6.2.1
II	deviation from straight [m]	6.1.2	6.2.2
III	deviation from origin [m]	6.1.3	6.2.3
IV	constrained mobility model test	6.1.4	6.2.4
V	connectivity	6.1.5	6.2.5

Table 6.1: Conducted experiments

All tests were conducted indoors at the Institute for Applied Mathematics and Computer Science at Neubrückstrasse 10 in Berne on floors two and three.

6.1.1 Scenario I: Distance Accuracy

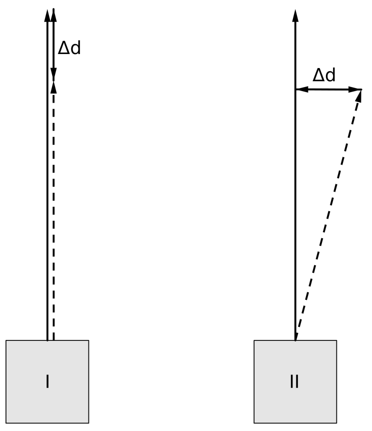

Figure 6.1: Concept of Scenarios I (left) and II (right)

The purpose of the test is to validate whether the distance defined in software is followed by the hardware. The test for distance accuracy was run with a rather low $(0.1m/s)$ and the highest speed supported by the robot $(0.5m/s)$. The author used a set of distances from 1 to 5 meters with 10 data points collected for each tuple of speed-distance to check if the robot can accurately cover the chosen distance. Figure 6.1 shows the experimental set-up. The detailed measurements are listed in the Appendix in Table 8.1. The evaluated data can be found in Table 6.7 for the speed of $0.1m/s$ and Table 6.8 for $0.5m/s$.

Script		linear.py
par.	metric	values
v	ms^{-1}	$0.1, 0.5$
d	m	$1-5$
n	$-$	10

Table 6.2: Parameters for Scenario I

6.1.2 Scenario II: Angular Accuracy

The purpose of the test is to show if the robot will follow a straight line. The speeds used in this experiment were again $0.1m/s$ and $0.5m/s$. For the experimental set-up, the robot was placed inside a square marked on the floor with the robot's dimensions ($35cm$ diameter). A straight line in front of the robot labelled the desired path the robot should take to its distance of $5m$ (see Figure 6.1). For both speeds the author made 10 experiments which can be found in Table 8.2. The evaluation results are shown in Table 6.9. The measured deviation of the robot equals the distance from the straight line to the robot's center point in a right angle from the line.

Script		linear.py
par.	metric	values
v	ms^{-1}	$0.1, 0.5$
d	m	5
n	$-$	10

Table 6.3: Parameters for Scenario II

6.1.3 Scenario III: Returning to Starting Position

To see the effect of the accumulation of the angular deviations in a more complex situation, the robot was run in a square with lateral length of $2m$(a), turning clockwise. The deviation from the starting and the end position was measured from the center of the robots back. The program was written so that in a perfect run the robot would stand exactly in the same location as it started, facing the same direction. Concerning speed worst-case scenario was chosen, meaning full speed at $5m/s$.

Script		square.py
par.	metric	values
v	ms^{-1}	0.5
a	m	2

Table 6.4: Parameters for Scenario III

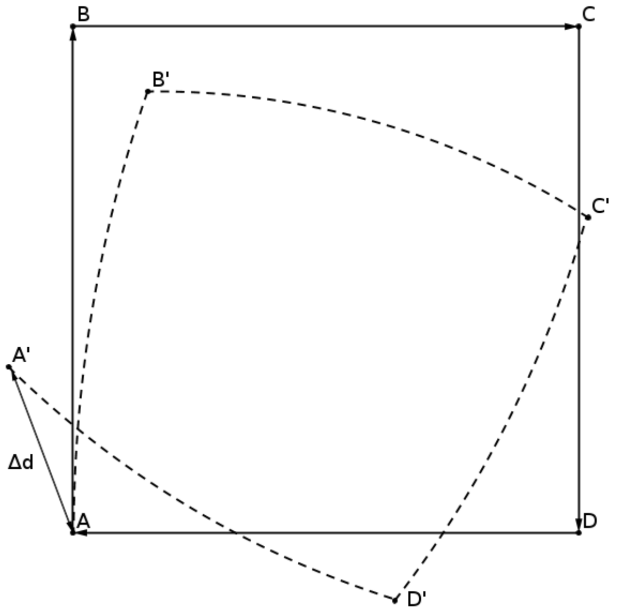

Figure 6.2: Sketch of the square experiment, Scenario III

6.1.4 Scenario IV: Small Scale Test

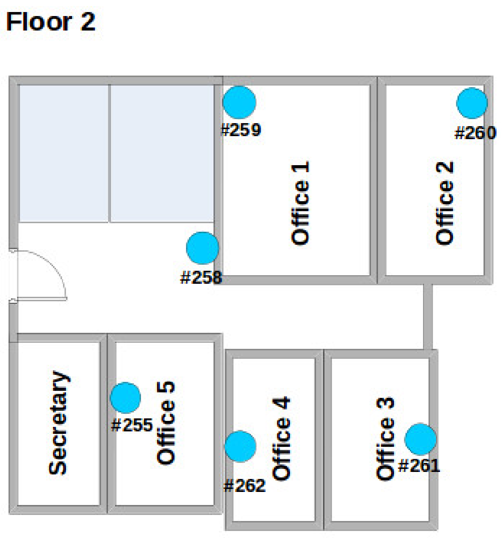

Figure 6.3: Sensor positions on the second floor, Scenario IV

This experiment was conducted in the hallway of the second floor at the Institute for Applied Mathematics and Computer Science at Neubrückstrasse 10 in Berne. The graph for this experiment is shown in Figure 6.4. The robot runs two different paths with the same total distance but with different numbers of turning points from vertex A to vertex F, namely: A - B - D - E - F and A - B - C - E - F.

Vertices indicate different rooms or locations in the hallway. Vertex A is close to the door of office N201, vertices B, C and E are in the middle of the hallway and offer an alternative path around vertex D besides the door of meeting room N206. Finally, vertex F is positioned in front of the door to office N205 (see Figure 6.4).

During the test runs, we also attempted to pick up the robot's signals with a wireless test bed to verify the robot's connectivity. Wireless network traffic was created with the `ping` command as described in Section 5.3. The positions of sensors are shown in Figure 6.3. Sensors 256, 257 and 263 are positioned outside the group's hallway and thereafter not shown in the figure. Moreover, the robot's position is measured after each experiment in order to test if more turns equal less accuracy.

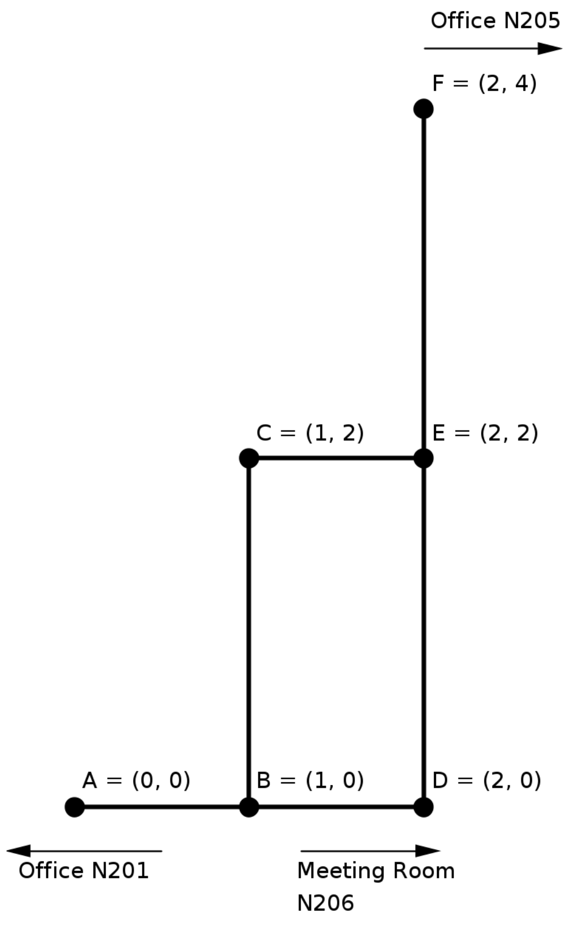

Figure 6.4: Graph layout for the combined experiment, Scenario IV

Script		constrained_mobility.py
par.	metric	values
v	ms^{-1}	0.5
$path_1$	–	A - B - D - E - F
$path_2$	–	A - B - C - E - F

Table 6.5: Parameters for Scenario IV

6.1.5 Scenario V: Connectivity

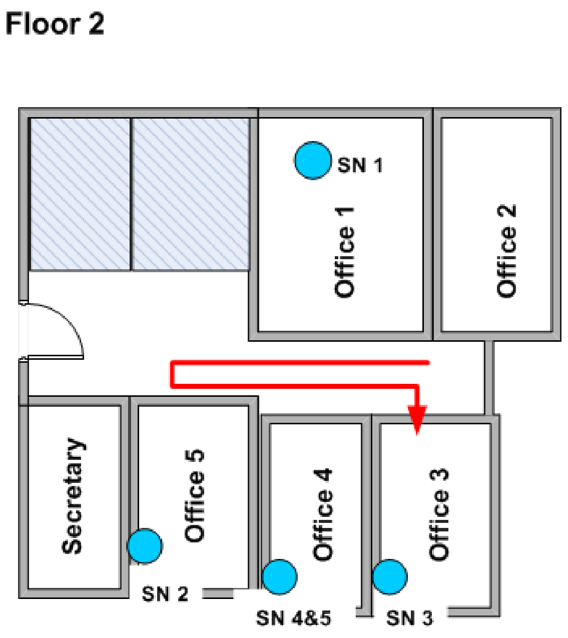

Figure 6.5: Travelled path in connectivity test, Scenario V

The evaluation was concluded by a pure connectivity test to show if the robot can be picked up by the wireless test bed. Again taking place on the second floor of the institute's building. The robot was driven by phone from the front of office 3 nearly to the end of the hallway, back and entering office 3 (see Figure 6.5). Wireless sensor nodes were placed in offices 1 and 3-5 to see if the robot can be picked up while receiving driving commands over wifi. For comparison of signal strength and collected packets by the test bed, a mobile phone with enabled wifi was placed in the robot's cargo bay.

Script		custom.launch
par.	metric	values
v	ms^{-1}	var.

Table 6.6: Parameters for Scenario V

6.2 Evaluation Results

6.2.1 Scenario I: Distance Accuracy

Distance [m]	1	2	3	4	5
Avg [m]	0.908	1.843	2.778	3.714	4.650
%	9.24	7.86	7.39	7.15	7.01
Stddev [m]	0.005	0.004	0.003	0.003	0.004

Table 6.7: Distance accuracy at a speed of 0.1 m/s

Distance [m]	1	2	3	4	5
Avg [m]	0.864	1.873	2.723	3.631	4.473
%	13.60	6.34	9.24	9.23	10.54
Stddev [m]	0.021	0.124	0.029	0.127	0.020

Table 6.8: Distance accuracy at a speed of 0.5 m/s

When looking at the evaluated data, an obvious result is that the robot constantly drives too short distances. This would not be a problem when the robot is run at slow speeds around $0.1m/s$. The difference in distance is merely $7 - 10\%$ with a standard deviation of $3 - 5mm$.

With higher speeds, the robot becomes less reliable (see Table 6.8). Not only does it drive even shorter distances as with $0.1m/s$ but the fluctuations in the covered distance do not allow for corrections in software. Even if the average of the driven distances could be calibrated to match the expected distance, the standard deviations still range from $2cm$ to $12.7cm$. In a single straight line run these deviations do not pose a real problem. The problems arise with the accumulated errors in the course of running the implemented models for even a short period of time.

It can be concluded to avoid the use of the maximum supported speed.

6.2.2 Scenario II: Angular Accuracy

Speed [m/s]	0.1	0.5
Avg [m]	0.100	1.094
Side	left	right
Stddev [m]	0.080	0.181

Table 6.9: Angular deviation from the ideal straight line

This experiment confirmed the author's earlier expectation and is concurrent with the data from the first experiment. While a deviation of $0.1m$ at $0.1m/s$ can be acceptable, at higher speeds the robot drifts to over 1 meter away from the ideal line. These deviations have two main causes, which come from one error source: the two wheels of the robot are controlled by two separate actuators. The author observed that the wheels do not start driving simultaneously, which resulted the robot to be slightly off already from the start. The second problem was that the robot drove in a curve and not as intended in a straight line. This problem is due to the fact that the robot's wheels are driven by two individual actuators. Cheap actuators exhibit varying accuracy when identical current and voltage are applied, resulting in deviating wheel speeds.

6.2.3 Scenario III: Returning to Starting Position

Run #	1	2	3	4	5	6	7	8	9	10
Δd [m]	1.017	0.834	0.792	0.897	0.911	1.105	1.036	0.737	0.769	0.476

Table 6.10: Deviation between starting and end position after driving in a square

The data in Table 6.10 shows that a maximum speed, where the robot tends to drift to the right, paired with multiple right turns affects seriously the robot's accuracy. The situation is depicted in Figure 6.2. The full arrows show the desired path whereas the dashed path depicts the way the robot actually drove.

It can be stated that with this experiment the usual shortcomings in distance accuracy of the robot do not affect the accuracy in this case as the final position was measured and not the individual corners of the square. So the robot drove a slightly smaller square but the deviation from the final position came mostly from the fact that it can not keep a straight line (see Section 6.1.2).

6.2.4 Scenario IV: Small Scale Test

Experiment #	1	2	3	4	5	Avg
Displacement w/ one turn [m]	0.621	1.108	1.016	0.717	0.770	0.846
Displacement w/ three turns [m]	1.305	1.236	1.204	1.048	0.729	1.104

Table 6.11: Deviation between intended and actual end position with the constrained mobility model

Sensor #	Count	Avg. Signal [dB]
255	0	-
256	139	-63.7
257	0	-
258	140	-31.5
259	176	-48.9
260	99	-56.3
261	157	-57.9
262	186	-53.3
263	0	-

Table 6.12: Picked up signals and average signal strength per sensor

The data in Table 6.11 confirms the initial assumption that with the same distance travelled and the same number of stops, more turns mean less accuracy. During a run with three turns, the robot was off from the desired position around an average of $1.104m$. This is $0.258m$ more than with only one turn at vertex `D`. The individual data points also confirm this observation. While with one turn only two deviations were above 1 meter, with three turns only a single run achieve a deviation below 1 meter.

When running the robot in confined spaces (e.g. hallways, through doors, etc) one has to keep in mind to put the vertices and edges as far enough from the walls as possible. While conducting this experiment, the robot hit the wall once and scratched it twice in eleven runs.

The result of the communications part of this experiment is shown in Table 6.12. During the experiment, which lasted for 22 minutes, the robot's signal was picked up around 900 times. The data reveals that the robot can be picked up the test bed and thus is suitable for testing it.

6.2.5 Scenario V: Connectivity

Node #	1	2	3	4	5
Phone [dB]	-77.9	-63.2	-75.1	-76.0	-75.8
Robot [dB]	-	-69.4	-72.9	-68.3	-68.2

Table 6.13: Average signal strength of the robot's and phone's wifi signal

Node #	1	2	3	4	5
Phone	28	144	169	190	148
Robot	-	58	132	111	112

Table 6.14: Number of picked up network packets of robot and phone

As shown in Table 6.13, the robot was picked up by the test bed but its signal strength was mostly significantly lower than the one of the phone. A possible explanation would be that the Gumstix antennas were not mounted on top of the robot but were inside the cargo bay, thus not having the best possible signal coverage. Another factor could be that the phone has a more powerful wifi chip and antenna, hence the better reception. Table 6.14 backs up the theory of worse coverage by the robot. Although there was constant communication back and forth between the robot and the steering phone, the test bed shows a much lower count of sightings of the robot than of the phone in the cargo bay.

Chapter 7

Conclusions and Outlook

7.1 Conclusions

The addressed topic of this thesis was providing a mobile platform for mobility tests in wireless sensor networks.

The author evaluated the applicability of the iRobot Create platform in terms of mobility and wireless connectivity. To test its use for experimentation with realistic human mobility, the author implemented two mobility models. The implementations include the random walk mobility model, which is often used in academic research, and the constrained mobility model as a more realistic approach in modeling human mobility.

When evaluating mobility models for implementation, one has to keep in mind the hardware's capabilities to follow said mobility model's properties. Furthermore, the testing environment has to be taken into account in measures of obstacle richness, dimensions and floor conditions. Obstacle rich environments require mobility models with predefined maps or possible locations and paths. Otherwise, the mobility platform might only visit small portions of the whole testing environment.

This thesis underlines the fact that working with low-end and small robots yields both distortions in accuracy as well as difficulties in mobility due to small obstacles. During the evaluation phase of this thesis (see Chapter 6) the author found that the robot tends to drift from the straight line. Furthermore, accuracy of the driven distance declines with higher speeds. However, connectivity poses no challenge as the robot can be picked up by the wireless test bed and thus is suitable for testing purposes.

It can be concluded that the hardware set-up for this thesis is best suited for obstacle free environments with medium to large dimensions (e.g. the size of a meeting room). Equipping the robot with additional sensors could also allow more confined spaces to be traversed and might introduce autonomous path finding.

7.2 Outlook

There are many possible ways to extend the robot's abilities in both hardware and software.

With the successful implementation of the mobility models (see Section 4.2), specifically the constrained mobility model, these models could be enriched with additional features and extensions of said models.

With intermediate programming knowledge in Python, the constrained mobility model could be extended to include weights to the graph's edges. These weights could correspond to the probabilities that a certain path is chosen for traversal. In turn this knowledge can be based on real world observations, e.g., location of type A (bakery) get visited more often than location of type B (kid's store). Also, other mobility models could be implemented for comparison reasons.

As noted in Chapter 3 the low cost sector in robotics and electronics has been emerging in the last several years and will presumably keep emerging in the future. With new hardware coming to the market constantly, the human mobility models could be transferred to another hardware platform. Furthermore, the option of making the implemented models independent of the Robot Operating System could be considered, for even more lightweight operation of the robot.

The inaccuracy experienced in following straight lines may be compensated by adding additional sensors such as depth cameras. The outcome of this upgrade would be improved checking of how and where the robot really is headed and on-the-fly correction if it deviates by a certain amount. Another advantage could be that the robot would be more autonomous and could explore areas without the need of a human constantly watching over it.

A different approach could be introduced by transforming the robot from a testing device for the test bed into a mobile node, integrated into the test bed. As the robot uses similar hardware with the Gumstix Overo Fire, it is nearly self evident to come to this conclusion. An advantage of this approach is that localising individual devices inside a crowded room could be done more dynamically and with better resolution.

Chapter 8

Appendix

Listing 8.1: Implementation of the Random Path Model

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('turtle_mover')
import rospy
import math
import random
import datetime

from geometry_msgs.msg import Twist
from turtlebot_node.msg import *
from turtlebot_node.srv import *

def sensor_callback(msg):
    global stuck
    if (twist.linear.x > 0) and ((msg.bumps_wheeldrops > 0) or
        msg.cliff_left or msg.cliff_front_left or msg.
        cliff_front_right or msg.cliff_right):
        stuck = True

def move_callback(event):
    global pub
    global twist
    global start_time
    global driving_time
    global timer
    global stuck
    if stuck:
        if twist.linear.x > 0:
            driving_time = rospy.get_time() - start_time
            twist.linear.x = 0
        timer.shutdown()
    pub.publish(twist)

def calc_waiting_time():
    return random.normalvariate(10,2)

def calc_turning_direction():
    return random.choice([1, -1])

def calc_turning_time():
    return random.uniform(0,4)

def calc_driving_speed():
    return random.normalvariate(0.5,0.2)

def calc_driving_time():
    return random.normalvariate(5,2)

def execute():
    global pub
    global twist
    global timer
    global driving_time
    global start_time
    global stuck

    rospy.init_node('random_walk')

    log_file = open('path{}.txt'.format(datetime.datetime.now()), 'w')

    print('turn_direction;turn_time;waiting_time;driving_time;driving_speed', file=log_file)
    log_file.flush()

    rospy.wait_for_service('turtlebot/set_operation_mode')
    mode = rospy.ServiceProxy('turtlebot/set_operation_mode', SetTurtlebotMode)
    mode(3)

    rospy.Subscriber('turtlebot/sensor_state', TurtlebotSensorState, sensor_callback)

    pub = rospy.Publisher('cmd_vel', Twist)
    twist = Twist()
    pub.publish(Twist())

    waiting_time = 0
    turning_time = 0
    turning_direction = 0
    driving_speed = 0
    driving_time = 0
    stuck = False

    direction_name = {-1: 'right',1: 'left'}

    random.seed()

    while not rospy.is_shutdown():
        waiting_time = calc_waiting_time()
        turning_time = calc_turning_time()
        turning_direction = calc_turning_direction()
        driving_speed = calc_driving_speed()
        driving_time = calc_driving_time()

        twist.angular.z = turning_direction * math.radians(45)
        twist.linear.x = 0
        stuck = False
        pub.publish(twist)
        timer = rospy.Timer(rospy.Duration(0.1),move_callback)
        rospy.sleep(turning_time)
        timer.shutdown()
        pub.publish(Twist())

        waiting_time = waiting_time - turning_time
        if waiting_time < 0:
            waiting_time = 0
        rospy.sleep(waiting_time)

        twist.linear.x = driving_speed
        twist.angular.z = 0
        pub.publish(twist)
        start_time = rospy.get_time()
        timer = rospy.Timer(rospy.Duration(0.1),move_callback)
        rospy.sleep(driving_time)
        timer.shutdown()
```

```
            pub.publish(Twist())
            print('{};{};{};{};{}'.format(turning_direction,
                turning_time, waiting_time, driving_time, driving_speed
                ), file=log_file)
            log_file.flush()


if __name__ == '__main__':
    try:
        execute()
    except rospy.ROSInterruptException:
        rospy.loginfo('Stopped!')
        rospy.loginfo('Node exiting.')
```

Listing 8.2: Implementation of the Constrained Mobility Model

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('turtle_mover')
import rospy
import math
import random
import os

from geometry_msgs.msg import Twist
from turtlebot_node.msg import *
from turtlebot_node.srv import *



def move_callback(event):
    global pub
    global twist
    pub.publish(twist)

def choose_next_node(node, graph):
    return random.choice(graph[node])

def needs_turn(facing_coordinates, facing_direction, current_node
        , next_node, graph, coordinates):
    dx = coordinates[next_node][0] - coordinates[current_node
        ][0]
    dy = coordinates[next_node][1] - coordinates[current_node
        ][1]
    if facing_coordinates == 'x':
        if ((facing_direction > 0) and (dy > 0)) or ((facing_direction < 0) and (dy < 0)):
            return 1
        elif ((facing_direction < 0) and (dy > 0)) or ((facing_direction > 0) and (dy < 0)):
            return -1
        elif ((facing_direction > 0) and (dx < 0)) or ((facing_direction < 0) and (dx > 0)):
            return 2
        elif ((facing_direction > 0) and (dx > 0)) or ((facing_direction < 0) and (dx < 0)):
            return 0
    elif facing_coordinates == 'y':
        if ((facing_direction > 0) and (dx > 0)) or ((facing_direction < 0) and (dx < 0)):
            return -1
        elif ((facing_direction < 0) and (dx > 0)) or ((facing_direction > 0) and (dx < 0)):
            return 1
        elif ((facing_direction > 0) and (dy < 0)) or ((facing_direction < 0) and (dy > 0)):
            return 2
        elif ((facing_direction > 0) and (dy > 0)) or ((facing_direction < 0) and (dy < 0)):
            return 0


def execute():
    global pub
    global twist

    rospy.init_node('turtle_mover')

    rospy.wait_for_service('turtlebot/set_operation_mode')
    mode = rospy.ServiceProxy('turtlebot/set_operation_mode', SetTurtlebotMode)
    mode(3)

    pub = rospy.Publisher('cmd_vel', Twist)
    twist = Twist()

    speed_file = open("speed.txt","r")
    linear_speed = float(speed_file.readline())
    speed_file.close()
    max_speed = 0.5
    angular_speed = math.radians(45)
    # either x or y
    facing_coordinates = 'x'
    # either +1 or -1
    facing_direction = 1
    # a node in the graph
    current_node = 'A'
    next_node = ''

    graph = {}
    # read in the graphs edges
    with open("graph.txt") as graph_file:
        for line in graph_file:
            (key, val) = line.split(":")
            val = val.strip()
            graph[key] = val.split(",")

    graph_file.close()

    # read in the graphs coordinates
    coordinates = {}
    with open("coordinates.txt") as coordinates_file:
        for line in coordinates_file:
            (key, val) = line.split(":")
            coordinates[key] = map(int, val.split(","))

    coordinates_file.close()

    predefined = os.path.exists("path.txt")
    if predefined:
        predefined_node_list = []
        path_file = open("path.txt","r")
        for line in path_file:
            predefined_node_list.append(line.strip())
        predefined_node_list.append("exit")
        nodes = iter(predefined_node_list)

    while (not rospy.is_shutdown()):
        if predefined:
            next_node = nodes.next()
        else:
                next_node = choose_next_node(current_node, graph)
            if next_node == "exit":
                rospy.signal_shutdown("Reached end of path")
            rospy.loginfo('Current Node: {} \n Next Node: {}'.
                format(current_node, next_node))
            turn_direction = needs_turn(facing_coordinates,
                facing_direction, current_node, next_node, graph,
                coordinates)
            if turn_direction != 0:
                turn_time = 0
                twist.linear.x = 0
                if turn_direction == 1:
                    twist.angular.z = angular_speed
                    turn_time = 2
                    if facing_coordinates == 'x':
                        facing_coordinates = 'y'
                    elif facing_coordinates == 'y':
                        facing_coordinates = 'x'
                        facing_direction = -1 * facing_direction
                elif turn_direction == -1:
                    twist.angular.z = -1 * angular_speed
                    turn_time = 2
                    if facing_coordinates == 'y':
                        facing_coordinates = 'x'
                    elif facing_coordinates == 'x':
                        facing_coordinates = 'y'
                        facing_direction = -1 * facing_direction
                elif turn_direction == 2:
                    twist.angular.z = angular_speed
                    turn_time = 4
                    facing_direction = -1 * facing_direction

                pub.publish(twist)
                timer = rospy.Timer(rospy.Duration(0.1),
                    move_callback)
                rospy.sleep(turn_time)
                timer.shutdown()
                twist.angular.z = 0
                pub.publish(Twist())
                rospy.sleep(2)

            distance = coordinates[next_node][0] - coordinates[
                current_node][0] + coordinates[next_node][1] -
            coordinates[current_node][1]
        driving_time = distance/linear_speed
        if driving_time < 0:
            driving_time = -1 * driving_time
        twist.angular.z = 0
        twist.linear.x = linear_speed
        timer = rospy.Timer(rospy.Duration(0.1),move_callback)
        rospy.sleep(driving_time)
        timer.shutdown()
        pub.publish(Twist())
        current_node = next_node
        next_node = ''
        rospy.sleep(2)

if __name__ == '__main__':
    try:
        execute()
    except rospy.ROSInterruptException: pass
```

Expected [m]	0.1 m/s [m]	0.5 m/s [m]
1	0.895	0.871
1	0.910	0.869
1	0.911	0.853
1	0.909	0.868
1	0.906	0.822
1	0.907	0.864
1	0.911	0.865
1	0.911	0.860
1	0.907	0.861
1	0.909	0.907
2	1.847	1.788
2	1.844	1.823
2	1.848	1.861
2	1.845	1.818
2	1.838	1.795
2	1.842	2.114
2	1.833	1.813
2	1.846	1.807
2	1.842	1.816
2	1.843	2.097
3	2.782	2.754
3	2.778	2.734
3	2.775	2.727
3	2.779	2.785
3	2.774	2.721
3	2.780	2.713
3	2.776	2.704
3	2.781	2.707
3	2.780	2.713
3	2.778	2.690
4	3.712	3.606
4	3.715	3.599
4	3.721	3.524
4	3.717	3.600
4	3.713	3.616
4	3.714	3.596
4	3.712	3.570
4	3.710	3.622
4	3.715	3.592
4	3.713	3.982
5	4.650	4.486
5	4.654	4.455
5	4.652	4.494
5	4.650	4.487
5	4.654	4.477
5	4.639	4.489
5	4.650	4.491
5	4.648	4.448
5	4.649	4.463
5	4.651	4.440

Table 8.1: Datapoints from the linear accuracy test

Speed [m/s]	linear deviation [m]	deviation side
0.1	0.211	left
0.1	0.251	left
0.1	0.055	left
0.1	0.01	left
0.1	0.02	left
0.1	0.054	left
0.1	0.050	left
0.1	0.122	left
0.1	0.093	left
0.1	0.134	left
0.5	1.101	right
0.5	1.077	right
0.5	1.366	right
0.5	1.361	right
0.5	0.980	right
0.5	0.892	right
0.5	0.899	right
0.5	1.250	right
0.5	1.100	right
0.5	0.910	right

Table 8.2: Datapoints of the angular accuracy test

Bibliography

[1] A. Galati and C. Greenhalgh, ”Human mobility in shopping mall environments”, in *Proceedings of the Second International Workshop on Mobile Opportunistic Networking*, ser. MobiOpp ’10, New York, NY, USA: ACM, 2010, pp. 1-7. [Online]. Available: `http://doi.acm.org/10.1145/1755743.1755745`

[2] P. Hui, A, Chaintreau, J. Scott, R. Gass, J. Crowcroft and C. Diot, ”Pocket switched networks and human mobility in conference environments”, in *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, ser. WDTN ’05, New York, NY, USA: ACM, 2005, pp. 244-251. [Online]. Available: `http://doi.acm.org/10.1145/1080139.1080142`

[3] M. J. Feeley, N. Hutchinson, and S. Ray, ”Realistic mobility for mobile ad hoc network simulation”, in *Ad-Hoc, Mobile, and Wireless Networks*, ser. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 324-329. [Online]. Available: `http://dx.doi.org/10.1007/978-3-540-28634-9_28`

[4] A. Jardosh et al., ”Towards realistic mobility models for mobile ad hoc networks”, in *Proceedings of the 9th annual international conference on Mobile computing and networking*, ser. MobiCom ’03, New York, NY, USA: ACM, 2003, pp. 217-229. [Online]. Available: `http://doi.acm.org/10.1145/938985.939008`

[5] J. Broch et al., ”A performance comparison of multi-hop wireless ad hoc network routing protocols”, in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, ser. MobiCom ’98, New York, NY, USA: ACM, 1998, pp. 85-97. [Online]. Available: `http://doi.acm.org/10.1145/288235.288256`

[6] E.M. Royer et al., ”An analysis of the optimum node density for ad hoc mobile networks”, in *Communications, 2001. ICC 2001. IEEE International Conference on*, vol.3, 2001, pp. 857-861 vol.3. [Online]. Available: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=937360&isnumber=20263`

[7] `http://www.netlab.tkk.fi/~esa/java/rwp/rwp-sample.gif` on 27.02.2013

[8] E. Hyytiä et al., ”Random Waypoint Model in Wireless Networks”, Slide 6, Helsinki University of Technology, Helsinki, June 2005, `http://www.math.helsinki.fi/mathphys/EVERGROW/virtamo.pdf` on 27.02.2013

[9] http://upload.wikimedia.org/wikipedia/commons/thumb/6/66/BrownianMotion.svg/1000px-BrownianMotion.svg.png on 27.03.2013

[10] F. Bai and A. Helmy, ”A Survey of Mobility Models in Wireless Adhoc Networks”, Page 8, University of Southern California, U.S.A, http://www.cise.ufl.edu/~helmy/papers/Survey-Mobility-Chapter-1.pdf on 27.03.2013

[11] A.L. Cavilla et al., ”Simplified simulation models for indoor MANET evaluation are not robust”, in *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, 2004, pp. 610-620. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1381964&isnumber=30129

[12] M. de Berg et al., ”Computational geometry: algorithms and applications”, Springer Verlag, 2000

[13] H. Liu et al., ”Survey of Wireless Indoor Positioning Techniques and Systems”, in *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol.37, no.6, 2007, pp.1067-1080. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4343996&isnumber=4343957

[14] D. Fox et al., ”Bayesian Filtering for Location Estimation”, in *Pervasive Computing, IEEE*, vol.2, no.3, 2003, pp. 24-33. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1228524&isnumber=27556

[15] P. Santi, ”Appendix B: Elements of Graph Theory, Asymptotic Notation, and Miscellaneous Notions”, in *Mobility Models for Next Generation Wireless Networks*, John Wiley & Sons Ltd, 2012, pp. 323-333. [Online]. Available: http://dx.doi.org/10.1002/9781118344774.app2

[16] A. Förster et al., ”MOTEL — A Mobile Robotic-Assisted Wireless Sensor Networks Testbed”, in *Proceedings of the WISH seminar on Wireless Integration of Sensor networks in Hybrid architectures*, University of Berne, 2012, pp. 13-15. [Online]. Available: http://cds.unibe.ch/research/pub_files/DWB12.pdf

[17] E. Zola et al., ”Impact of Mobility Models on the Cell Residence Time in WLAN Networks”, in *Sarnoff Symposium, 2009. SARNOFF ’09. IEEE*, 2009, pp. 1-5. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4850370&isnumber=4850271

[18] C. Fok et al., ”Pharos: A Testbed for Mobile Cyber-Physical Systems”, in *Tech. Rep. TR-ARiSE-2011-001*, University of Texas at Austin, 2011. [Online]. Available: http://pharos.ece.utexas.edu/pubs/TR-ARiSE-2011-001.pdf

[19] http://pharos.ece.utexas.edu/wiki/index.php/Main_Page on 17.08.2013

[20] O. Rensfelt et al., ”Sensei-UU: A Nomadic Sensor Network Testbed Supporting Mobile Nodes”, Division of Computer Systems, Department of Information Technology, Uppsala University 2009. [Online]. Available: `http://user.it.uu.se/~frehe489/docs/09_tech_report_sensei_uu.pdf`

[21] `http://www.willowgarage.com/pages/pr2/order` on 24.02.2013

[22] `http://www.irobot.com/en/us/Company/About.aspx` on 16.04.2013

[23] `http://john.whelans.net/wp-content/uploads/2010/04/roomba560_sideview.jpg` on 02.02.2013

[24] `http://www.ros.org/wiki/Robots/Roomba` on 24.02.2013

[25] iRobot Roomba Serial Command Interface (SCI) Specification, Page 2, `www.irobot.com/images/consumer/hacker/Roomba_SCI_Spec_Manual.pdf` on 05.03.2013

[26] `http://www.robotshop.com/blog/robotinho-on-roomba-quaddrive-a-robotic-guide-` on 24.02.2013

[27] `http://irbt.imageg.net/graphics/product_images/pIRBT-3426567v380.png` on 02.02.2013

[28] `http://store.irobot.com/product/index.jsp?productId=2586252` on 02.02.2013

[29] iRobot Create Open Interface (OI) Specification, Page 9, `http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf` on 30.1.2013

[30] `http://answers.ros.org/question/12385/irobot-create-odometry-package/` on 27.02.2013

[31] `https://d3iwea566ns1n1.cloudfront.net/images/product/GUM3503F.top.jpg` on 02.02.2013

[32] `https://d3iwea566ns1n1.cloudfront.net/images/product/PKG30023.top.jpg` on 02.02.2013

[33] `https://www.gumstix.com/store/product_info.php?productsid=227` on 30.12.2012

[34] `http://omappedia.org/wiki/BootloaderProject` on 02.02.2013

[35] `https://www.gumstix.com/store/product_info.php?products_id=280` on 30.12.2012

[36] `http://www.generationrobots.com/site/medias/Mobile-Robot-Kobuki.jpg` on 02.02.2013

[37] http://kobuki.yujinrobot.com/about/specifications/ on 30.01.2013

[38] https://github.com/turtlebot on 16.04.2013

[39] http://store.iheartengineering.com/robots/turtlebot/ihe-2700-000c-0000.html on 05.03.2013

[40] http://store.iheartengineering.com/robots/irobot/ihe-4710-0000-0000.html on 05.03.2013

[41] http://www.flickr.com/photos/willowgarage/5589373976/sizes/o/in/photostream/ on 02.02.2013

[42] http://1.bp.blogspot.com/-3VYHwkvGa8U/UIVK7DSMhVI/AAAAAAAAC8o/r3oHFO4Mxhw/s1600/IHE-2700-000C-0000-00.jpg on 02.02.2013

[43] http://turtlebot.com/ on 30.12.2012

[44] http://store.iheartengineering.com/robots/turtlebot/ihe-2700-000c-0000.html on 30.12.2012

[45] http://store.iheartengineering.com/catalog/product/gallery/id/334/image/614/ on 02.02.2013

[46] http://store.iheartengineering.com/robots/turtlebot/accessories/ihe-0200-0000-fa00.html on 30.12.2012

[47] http://electronics.howstuffworks.com/microsoft-kinect.htm on 25.02.2013

[48] http://msdn.microsoft.com/en-us/library/jj131033.aspx on 25.02.2013

[49] http://i.msdn.microsoft.com/dynimg/IC584396.png on 25.02.2013

[50] http://3.bp.blogspot.com/-Q-pB7CzbBgQ/Tm23QaqSryI/AAAAAAAACfA/_UwxBXGkhiw/s1600/xtion-open-03.jpg on 25.02.2013

[51] http://www.asus.com/Multimedia/Xtion_PRO_LIVE/#specifications on 25.02.2013

[52] http://gumstix.org/add-software-packages.html on 02.02.2013

[53] http://gumstix.org/create-a-bootable-microsd-card.html on 02.02.2013

[54] http://www.dailytech.com/IBM+Freescale+Samsung+Form+Linaro+to+Aid+in+Developing+ARMcompatible+Software/article18611.htm and http://www.linaro.org/downloads/1203 on 02.02.2013

[55] https://wiki.ubuntu.com/ARM/RootfsFromScratch/QemuDebootstrap on 30.12.2012

[56] http://www.willowgarage.com/pages/software/ros-platform on 25.02.2013

[57] http://www.ros.org/wiki/ROS/Technical%20Overview on 17.08.2013

[58] http://www.ros.org/wiki/ROS/Technical%20Overview?action=AttachFile&do=get&target=master-node-example.png on 17.08.2013

[59] http://www.willowgarage.com/pages/software/ros-platform on 12.03.2012

[60] https://help.ubuntu.com/community/SerialConsoleHowto on 02.02.2013

[61] http://kfuresearch.wordpress.com/2011/08/05/installing-ubuntu-and-ros-on-the-gumstix-overo-fire/ on 02.02.2013

[62] http://gumstix.org/connect-to-my-gumstix-system.html on 02.02.2013

[63] http://www.ros.org/wiki/groovy/Installation/Source on 24.02.2013

[64] http://www.ros.org/wiki/groovy/Installation/Ubuntu#groovy.2BAC8-Installation.2BAC8-Sources.Setup_your_sources.list on 26.02.2013

[65] http://ros.org/wiki/groovy/Installation/Raspbian/Source#Collada on 26.02.2013

[66] http://www.ros.org/wiki/catkin/Tutorials/workspace_overlaying#Adding_Packages_to_Your_catkin_Workspace on 24.02.2013

[67] http://ros.org/wiki/turtlebot_create?distro=groovy# on 27.02.2013

[68] http://www.ros.org/wiki/ROS/Installation on 27.02.2013

[69] http://ros.org/wiki/turtlebot/Tutorials/AndroidControl on 05.03.2013

[70] https://play.google.com/store/apps/details?id=org.ros.android.app_chooser on 30.12.2012

[71] https://play.google.com/store/apps/details?id=ros.android.teleop on 30.12.2012