

Bachelor's Thesis

Controlling a TurtleBot 2 through a web interface

Jorge Alas
Anders Holm



Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.



Controlling a TurtleBot 2 through a web interface

By

Jorge Alas and Anders Holm

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Sammanfattning

Omegapoint AB förvärvade en TurtleBot 2 för att utforska en av de mer populära open source robotarna på marknaden. Omegapoint ville ha en webbapplikation, skriven speciellt för dem, för att styra TurtleBot 2 för användning i Omegapoint kontorsmiljö. Ett av de viktigaste hänsynstagandet vid utformningen av programmet var att det skulle vara körbart och användbart inte bara på stationära datorer och bärbara datorer utan även på smartphones och surfplattor. Ett mycket liknande projekt är coffee bot från <http://learn.turtlebot.com/>, vilket bygger på samma princip, och styr TurtleBot 2 från en webbapplikation. TurtleBot är en open source robot byggd i samarbete med de ursprungliga skaparna av Robot Operating System (ROS). Ett pythonskript, `coffee_bot.py`, används för att hantera kommunikationen mellan databasen och roboten. Webbplatsen har utvecklats med HTML, JavaScript och CSS. En databas har skapats där koordinaterna från webbsidan förvaras. Pythonskriptet `coffee_bot.py`, loopar och kontrollerar om databasen har nya poster. De flesta av de planerade funktionerna är implementerade och har klarat tester på flera enheter och anses därför färdiga och fullt fungerande.

Nyckelord: TurtleBot, Robot Operating System, HTML, JavaScript, CSS, Python

Abstract

Omegapoint AB acquired a TurtleBot 2 to explore one of the more popular open source robots on the market. Omegapoint wanted a web application, written specifically for them, to control the TurtleBot 2 for use in the Omegapoint office environment. One of the main things taken into consideration when designing the application was that it had to be runnable and usable not only on stationary computers and laptops but also on smartphones and tablets. One very similar project is the coffee bot from <http://learn.turtlebot.com/> building on the same principle, controlling the TurtleBot 2 from a web application. The TurtleBot is an open source robot built in collaboration with the original makers of the Robot Operating System (ROS). A python script, `coffee_bot.py`, is used to handle the communication between the database and the robot. The web site was developed with HTML, JavaScript and CSS. A database was created where the coordinates from the web page were stored. The python script, `coffee_bot.py`, would loop and check that database for new entries. Most of the planned features are implemented and have passed testing on several devices and are thus considered done and fully working.

Keywords: TurtleBot, Robot Operating System, HTML, JavaScript, CSS, Python

Acknowledgments

We would like to thank Omegapoint for providing the TurtleBot 2 and coming up with such interesting project idea. Big thanks to Mats Lilja för all the useful help and engaging discussions and thanks to Christian Nyberg for being our examiner.

Jorge Alas and Anders Holm

Contents

Sammanfattning.....	2
Abstract	3
Acknowledgments.....	4
1. Introduction	9
1.1. Background	9
1.2. Purpose	9
1.3. Goal.....	9
1.4. Problem.....	10
1.4.1. What already available functionality can be used in this project?10	
1.4.2. How can location information be presented through the web application in a way that makes a user able to control the TurtleBot? 10	
1.4.3. How can coordinates be passed from a web application in a way that the TurtleBot can map them to its internal navigation system?10	
1.5. Design scope	10
2. Method	11
2.1. Planning and designing interaction with the TurtleBot	12
2.1.1. Form.....	12
2.1.2. Manner of behavior	12
2.1.3. Function	12
2.2. Designing the application	13
2.2.1. Approach.....	13
2.2.2. Scope.....	13
2.2.3. Feedback	13

2.2.4.	Feedforward.....	14
2.2.5.	Fitts's Law	14
2.2.6.	Standards	14
2.3.	Sending coordinates to the TurtleBot.....	15
2.4.	Work flow and dependencies	15
2.5.	Source criticism.....	18
2.5.1.	Online examples	18
2.5.2.	Lynda.....	18
2.5.3.	Online documentation	19
2.5.4.	WC3School.....	19
2.5.5.	Designing for interaction 2nd ed. - Creating Innovative Applications and Devices.....	19
2.5.6.	TurtleBot tutorial sites and ros.org.....	19
2.6.	Related Work	19
2.6.1.	Coffee bot	19
3.	Technical background	21
3.1.1.	TurtleBot	21
3.1.2.	ROS.....	22
3.1.3.	Programming languages	22
3.1.4.	Web server.....	29
4.	Result	30
4.1.	TurtleBot software.....	30
4.1.1.	Starting the TurtleBot	30
4.1.2.	Navigation	30
4.1.3.	Creating a map.....	31
4.1.4.	Coffee_bot.py	33

4.1.5.	TurtleBot server	35
4.1.6.	Modifications of coffee_bot.py and coffee_queue.php	36
4.1.7.	Turtlepower	37
4.2.	Application	38
4.2.1.	Home.....	38
4.2.2.	Location List	39
4.2.3.	Route List	39
4.2.4.	Order.....	40
4.2.5.	Power.....	41
4.2.6.	About	41
4.2.7.	Continue.....	42
4.2.8.	Header	42
5.	Conclusion.....	44
5.1.	Application Functionality	44
5.2.	Application Design	44
5.3.	Robot Functionality.....	45
5.4.	Problem Statements	45
5.4.1.	What already available functionality can be used in this project?45	
5.4.2.	How can location information be presented through the web application in a way that makes a user able to control the TurtleBot? 46	
5.4.3.	How can coordinates be passed from a web application in a way that the TurtleBot can map them to its internal navigation system?46	
6.	Terminology.....	47
	References.....	48

Appendix A: Web site screen shots	52
Home View	52
Laptop	52
Smartphone.....	52
Exapanded routes list	53
Remove confirmation popup	53
Continue error popup	54
Power	54
Appendix B: TurtleBot specification sheet	55

1. Introduction

1.1. Background

Omegapoint AB was founded in 2002 with a vision of creating the leading IT consulting firm in Sweden. As part of Omegapoint's focus on competence and expertise combined with the desire to keep up with an ever-changing market, the company on occasion acquire products that might be of interest either to their current field of operation or to possible future fields of operation.

Robotics is a rapidly growing field with practical purposes ranging from domestic and commercial to military use. Omegapoint acquired a TurtleBot 2 (Ros.org, 2016) to explore one of the more popular open source robots on the market. TurtleBot 2 is an open robotics platform designed for education and research on state of the art robotics (Ros.org, 2016).

Omegapoint wants a web application, written specifically for them, to control the TurtleBot 2 for use in the Omegapoint office environment. Although there already are many applications and programs written for the TurtleBot 2, this thesis will be concerned with tailoring an application specifically for Omegapoint.

1.2. Purpose

This bachelor's thesis aims to develop a web application for controlling a TurtleBot 2 in Omegapoint's office environment.

1.3. Goal

Set up a framework for a web application. This includes a database, a web server and the basic classes needed to communicate with the robot.

Develop a web application with functionality for controlling a TurtleBot 2 in the Omegapoint office in Malmö.

1.4. Problem

The basic functionality of the web application that would control the TurtleBot was the basis for the project. The following three questions (1.4.1, 1.4.2, 1.4.3) were phrased to address problems regarding that basic functionality.

1.4.1. What already available functionality can be used in this project?

Some functionality for the TurtleBot is available online (Ros.org, 2016). Part of this project will be to find that functionality and include it in the application.

1.4.2. How can location information be presented through the web application in a way that makes a user able to control the TurtleBot?

The TurtleBot will be controlled through the sending of location specific coordinates. The initial purpose of the web application will be to handle the coordinates and pass them on to the TurtleBot.

At a later stage the location information should be presented in a way that is comprehensible to the employees of Omegapoint.

1.4.3. How can coordinates be passed from a web application in a way that the TurtleBot can map them to its internal navigation system?

The interpretation and translation to a TurtleBot 2 readable format of the coordinates is also included in the project.

1.5. Design scope

The web application will be runnable in Chrome version 50.0.2661.102 m and Firefox version 35.0.1. Only critical functionality not available online will be developed. Additional functionality to the TurtleBot 2 will not be implemented.

2. Method

The workflow of the project was divided into iterations, with one iteration spanning over approximately one week. The three main parts of one iteration were developing, testing and report writing. Each iteration ended with an oral evaluation and decisions regarding the content of the next iteration.

As new objectives arose they were added to a project specific trello board (Trello.com, 2016). There were no expressed limitations on how much work that could be in progress at any given time except for the limitations of the iterations. As the content of each iteration was decided, the workload of the iteration was implicitly decided. There were no defined roles within the project.

The work method was loosely based on the agile method Kanban. According to Henrik Kniberg and Mattias Skarin in Kanban and Scrum - making the most of both (Kniberg and Skarin, 2010), Kanban can be summarized in the following way:

- Visualize the workflow o Split the work into pieces, write each item on a card and put on the wall. o Use named columns to illustrate where each item is in the workflow.
- Limit Work In Progress (WIP) – assign explicit limits to how many items may be in progress at each workflow state.
- Measure the lead time (average time to complete one item, sometimes called “cycle time”), optimize the process to make lead time as small and predictable as possible.

The workflow was visualized through trello and the work in progress was limited through the limiting of content per iteration. The lead time was not measured in the project but as the workload of each iteration was decided, the content of the previous iteration was taken into account, so that each iteration did not contain more work than the team could handle.

2.1. Planning and designing interaction with the TurtleBot

When planning interaction with a robot there are three main things that are needed to take into consideration.

Form: Appearance and physical characteristics.

Manner of behavior: How it goes about its activities, handles interaction with its environment and how social it is.

Function: How it communicates and what “senses” does it use to communicate. (Saffer, 2010, p.200)

Two of these points were left pretty much untouched since it felt like the current solution was good enough.

2.1.1. Form

The TurtleBot was already built and there was no real need to change its physical appearance to improve its ability to execute its chores.

2.1.2. Manner of behavior

A little work had to be done to make sure the TurtleBot was docking to its charging station correctly when needed and with the monitoring of its battery status.

But the main things like collisions and other working environment problems are already handled at a “good enough” level. Improving this was estimated to be too time consuming in relation with the expected performance gain.

2.1.3. Function

This is where the main part of the work was put into. The robot has some sensors and the ability to emit sound. But considering the user wouldn't always be close to the TurtleBot when communicating, developing the ability to communicate through gestures and sound was never really an option. Instead an application was built to handle both the input to and the output from the TurtleBot.

Even though some interaction is possible using the buttons installed in the TurtleBots base the same functionality is accessible through the application since using the physical may be inconvenient.

2.2. Designing the application

2.2.1. Approach

When designing the application there was a lot to take into consideration, like the availability of the customer and the limited resources (time). For this reason the approach used for the interaction design was a “genius design” or “rapid expert design” where the designers use previous experience and intuition to create a design they think is best suited for the user and the product. The user is usually not involved during the development process but is sometimes incorporated at later stages to confirm the design works as the designer intended. One of the benefits of this approach is that it can be less time consuming than the other approaches and in this case gives a good enough design in a product where the interaction design is not the most critical component. (Saffer, 2010, p.43)

Some elements of the “activity centered design” approach were also present in the design on the application. Where tasks and activities were selected as critical for the project. (Saffer, 2010, p.35)

2.2.2. Scope

One of the main things taken into consideration when designing the application was that it had to be runnable and usable not only on stationary computers and laptops but also on smartphones and tablets. None or very little effort was put into satisfying the needs of users with color blindness, users with other disabilities, robots and screen readers.

2.2.3. Feedback

During test runs it was discovered that there was a need to improve the feedback when interacting with the application. Ideally every keypress should give some kind of feedback and anything loading more than 1 second should be noticed with a loading bar or

some kind of indicator that the application is working (Saffer, 2010, p.131). In this case it was decided that most of the feedback could be done with popups for information and confirmation.

With popups the application is able to communicate a good amount of information to assure the user that the correct action is being processed. The popups were designed to visually melt in with the rest of the application so they don't feel like a disturbing object.

2.2.4. Feedforward

Telling the user what will happen before performing action with plain text or other media (Saffer, 2010, p.133) was planned to be implemented throughout the application but got discarded because of time limitations.

2.2.5. Fitts's Law

When placing and sizing buttons in the different views some consideration has been taken to Fitts's law to ease the navigation, especially when navigating on touch screens. Fitts's law claims that the time it takes to rapidly move to a target area is a logarithmic function of the ratio between the distance to the target and the width of the target. The law is applied to web design to make the user's cursor movements more efficient. (Saffer, 2010, p.134)

2.2.6. Standards

There are good reason for following interface standard. Using the same layout as other common applications lets the user use previous experience when interacting with the application. Having to commit time to learn or relearn how to navigate the application may give the user a negative user experience. Of course violating standards is acceptable if there is a superior alternative. (Saffer, 2010, p.134)

The application tried to stick to standards as much as possible as there was no gain in breaking them. Having the header bar on top with a clickable icon on the top left corner and the placement of text and buttons on the popups are examples of pretty standardized layouts.

2.3. Sending coordinates to the TurtleBot

In the beginning of the project, before it was decided that the `coffee_bot.py` script should be used, the project team wrote another python script that would take coordinates as parameters and send the robot to the corresponding location. Another python script from Mark Silliman's public GitHub repository was used as a base when writing the new script (`go_to_specific_point_on_map.py`). (GitHub, 2016)

The resulting script would take the two coordinates as command line arguments and it worked well when running it from the Ubuntu terminal. A rudimentary homepage was also developed, which also would take two coordinates as user input and pass them along to the command line script. The script would not send the robot to the corresponding location however. The web server did not have the permissions needed to run the python file with the necessary library imports. To get around that problem a database was created where the coordinates from the web page were stored. The python script would then loop and check that same database for new entries.

The `coffee_bot.py` script already had this functionality along with other functionality that would fit well into the project. That other functionality included keeping track of the battery levels of the Kobuki base and the onboard laptop, automatically going to the docking station to charge, and waiting for user input before driving from one location to the next.

2.4. Work flow and dependencies

To get an overview of the critical phases, a work flow and dependencies diagram was made. The diagram can be seen in Figure 1 as well as a short description of each phase on page 16 and 17.

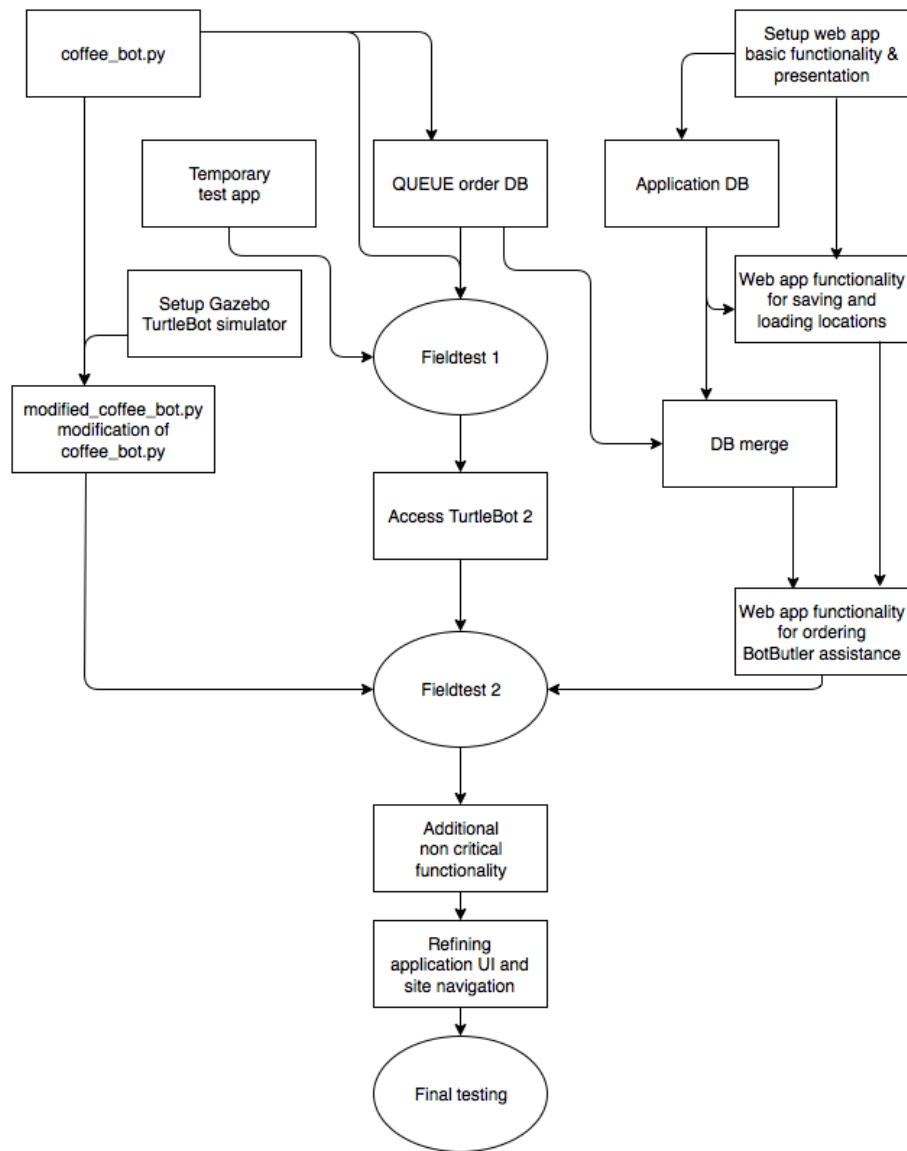


Figure 1 - Work flow and dependencies

- **Setup Web app basic functionality & presentation**
Rough planning of the web application, A Home page, navigation elements, mobile adaptability
- **Web app functionality for saving and loading locations**
Functionality to save coordinates for a location, List saved locations, Remove locations etc. as well as Creating routes, listing them, deleting them etc.
- **Web app Functionality for ordering BotButler assistance**
Functionality to call the Botbutler to a requested location or to go a specific route
- **Application DB**
Tables for, locations, routes etc.
- **DB merge**
Merge of the two databases. The new database stores locations routes etc. and handles the botbutlers order queue.
- **QUEUE order DB**
Tracks the Botbutlers orders, used to communicate coordinates to the Botbutler.
- **Temporary test app**
Very simple application with functionality to manually send coordinates to the robot.
- **Fieldtest 1**
Test of the Coffee_bot.py script, mainly to confirm the TurtleBot 2 is working as expected and forming a deeper understanding of what is necessary to fulfill our requirements.
- **Fieldtest 2**
Test the collaboration between the web application, the merged database, the new script and the TurtleBot 2.
- **coffee_bot.py**
Review and testing of the code. Learn where modifications need to be made.
- **modified_coffee_bot.py - Modification of coffee_bot.py**

Modification of coffee_bot.py to work with new merged database and with additional functionality.

- **Access TurtleBot 2**

Since we are not the only ones working with the TurtleBot 2, getting access and setting up the TurtleBot 2 required some work

- **Setup Gazebo - TurtleBot simulator**

Gazebo was the program used to simulate the TurtleBot 2 when it was not available

- **Additional non critical functionality**

Some work on application functionality and writing additional software for the TurtleBot

- **Refining application UI and site navigation**

Work on UI and navigation. New navigation bar etc.

- **Final testing**

Final testing and observation of of the entire project.

2.5. Source criticism

2.5.1. Online examples

The information gathered from forums or examples online had to be thoroughly tested before being applied to the project therefore credibility of the source was less important. When researching solutions at forums eg. stackoverflow, comments, the posters reputation and upvotes weighed in if the presented solution would be considered to further research and testing to minimize time wasted on unsuccessful tests.

2.5.2. Lynda

Lynda.com was used to learn some of the necessary tools like, HTML, Bootstrap, CSS, Python etc. Lynda is a well-established company that offers online education for subscribers. Lynda is since 2005 owned by Linked In and is considered a very trustworthy source where the content is relatively up to date.

2.5.3. Online documentation

Most of the documentation and references has been from the official sites developing or maintaining the tools.

2.5.4. WC3School

Wc3School was used as a reference when developing the application. The site is well established and has a good reputation. But most of the resources were validated by tests before implementation into the application

2.5.5. Designing for interaction 2nd ed. - Creating Innovative Applications and Devices

Book written by Dan Saffer principal at the San Francisco based product design consultancy Kicker Studio. The book has been used as literature in Courses like Introduction to Interaction Design (DA157A) given at Malmö högskola and is therefore considered verified and trustworthy.

2.5.6. TurtleBot tutorial sites and ros.org

The official tutorial site for the TurtleBot, learn.turtlebot.com, and the linked repositories along with the homepage and wikis for the Robot Operating System are all part of an open source community. Although parts of the resources provided might be in a development stage these sources were considered trustworthy for the purposes of this project.

2.6. Related Work

2.6.1. Coffee bot

One very similar project is the coffee bot from <http://learn.turtlebot.com/> building on the same principle, controlling the TurtleBot 2 from a web application. The coffee bot is able to send the TurtleBot 2 to a specific location using coordinates given from RViz. The coffee bot has to a certain degree been the foundation that

this project has built upon, it has also been a great project to learn and understand the fundamentals of how the TurtleBot 2 works.

Some of the coffee bots open source code has been reused and modified to work in the Botbutler, while other parts such as the web application had to be written from scratch to better fulfill the requirements on the Botbutler project.

3. Technical background

3.1.1. TurtleBot

The TurtleBot is an open source robot built in collaboration with the original makers of ROS, with a focus on education and early-stage development. The TurtleBot consists of a mobile base, a 3D sensor (Kinect), a laptop computer, and the TurtleBot mounting hardware kit as can be seen in Figure 2.



Figure 2 - The TurtleBot 2

On the ROS homepage (Wiki.ros.org, 2016) the following can be read regarding the TurtleBot:

“TurtleBot combines popular off-the-shelf robot components like the iRobot Create, Yujin Robot's Kobuki, Microsoft's Kinect and Asus' Xtion Pro into an integrated development platform for ROS applications.”

The operating system on the laptop that sits on the TurtleBot used in this project, is Ubuntu 14.04. The Robot Operating System (ROS) and the TurtleBot packages were pre-installed on the image file used

to install Ubuntu. The image file was found on the ROS homepage (Wiki.ros.org, 2016).

For more information on the technical specifications of the TurtleBot, see Appendix B.

3.1.2. ROS

The Robot Operating System (ROS) was originally developed by the Stanford Artificial Intelligence Laboratory. In 2007, Willow Garage, a robotics research institute, provided major contributions and created well-tested implementations. ROS is a framework for writing robot software. It consists of a collection of tools and libraries that were created with an intent of simplifying the task of creating complex and robust robot behaviour across many different robotic platforms. ROS runs on Linux, is written in C++ and Python, and provides operating system-like functionality. (Ros.org, 2016)

Examples of what the Robot Operating System provides includes hardware abstraction, device drivers, libraries, visualizers, message-passing, and package management. ROS is licensed under an open source, BSD license. (Wiki.ros.org, 2016)

3.1.3. Programming languages

C++

The program used to start and stop the programs on the TurtleBot was written in C++. That programming language was used because the members of the project have programmed with C++ before and felt comfortable using it. The program was written specifically for this project and the need for portability was considered negligible.

Python

The program that is used to handle the communication between the database and the robot, `coffee_bot.py`, was downloaded from Mark Silliman's GitHub page (Silliman, 2016). It was written in Python which is a widely used interpreted language (Python.org, 2016) with libraries for ROS and TurtleBot (Wiki.ros.org, 2016). The

coffee_bot.py script contains a lot of the functionality needed and was therefore incorporated in the project.

HTML

HyperText Markup Language commonly used to create web pages. HTML is considered a cornerstone when making web pages together with CSS and javascript. (W3schools.com, 2016)

With previous experience it was known that this would get the job done and that there was no need to research other alternatives. HTML 5 was the version used as it was the latest stable version at the time of the development and no reason to explore other releases was considered necessary.

Online courses at lynda.com were taken to get further understanding of the tool.

jQuery

For the javascript jQuery was used. jQuery is a common javascript library that works on multiple browser and handles the client side scripting for events, animations, navigating the HTML document etc. (jquery.org, 2016)

jQuery 2.1.4 was used since it was the version recommended by the bootstrap when the development was initialized. Other alternatives such as AngularJS was considered but got discarded because no needs for other functionality emerged. Online courses at lynda.com were taken to get further understanding of the tool.

CSS

Cascading style sheets is a style sheet language, and is together with HTML and Javascript one of the cornerstones used on most web pages. CSS describes how the HTML document is to be presented by the web browser. (W3schools.com, 2016)

Online courses at lynda.com were taken to get further understanding of the tool.

Ajax

When writing the web application there were parts where communication between the client and the server was needed but reloading the page was undesired or would even negate other already implemented functionality. Using Ajax (Asynchronous JavaScript + XML) seemed like a good approach to achieve this.

Ajax introduces an Ajax engine between the client and the server. The client speaks with the engine and the engine communicates with the server on behalf of the client.

As seen in Figure 3 the user uses JavaScript to communicate with the engine locally and delegates the remote communication to the engine.

Not refreshing the page, thus not having to request the entire page from the server has other nice side effects such as increasing the bandwidth efficiency by just transferring essential data and not the entire html document.

That way the user can asynchronously interact with the application without having to wait for the server to process its requests.

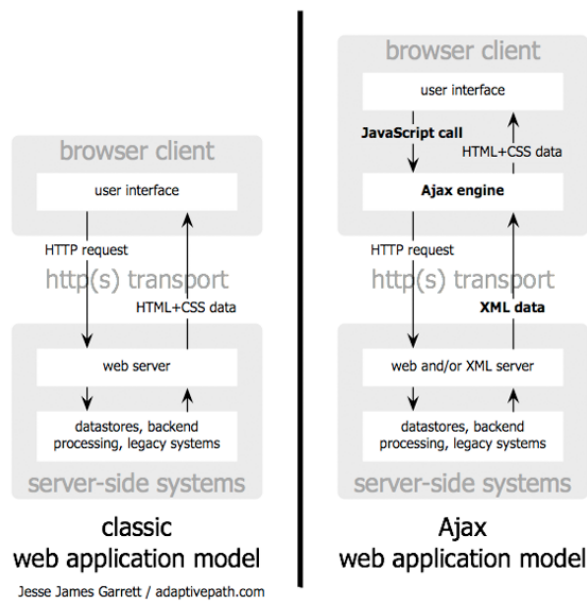


Figure 3 - Ajax web application model

As seen in Figure 4, in a classic web application the user would have to wait for the server to process its requests while in the Ajax model the user can interact with the application while the server is processing its requests. (Garrett and Garrett, 2005)

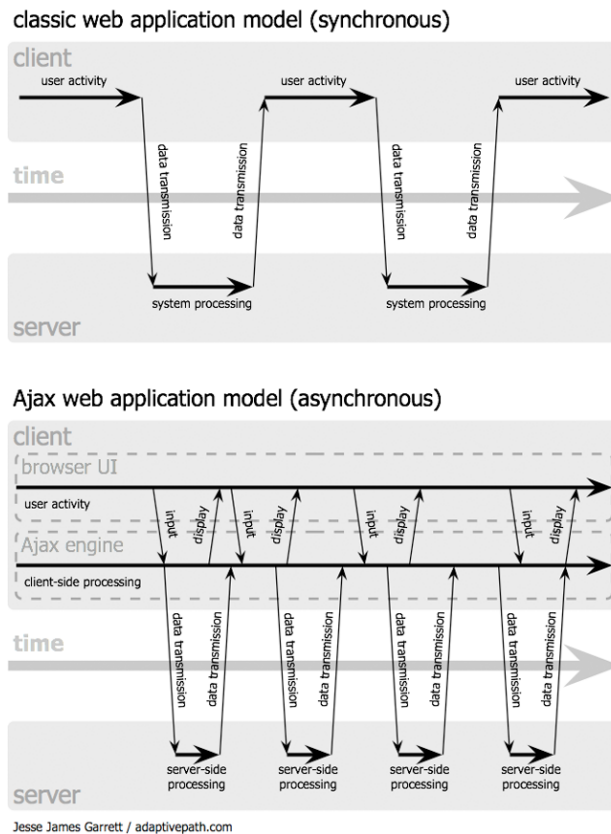


Figure 4 - Asynchronous Ajax web model

Bootstrap

While planning the design of the application one of the requirements were that the application would work well on devices with smaller displays such as smartphones as well as on normal sized desktop displays. It was known that at least to some point this was

achievable through bootstrap. This lead to further researching of the bootstrap framework before deciding to use it as a tool for the implementation of the application. Some online courses were taken at lynda.com to get acquainted with how to use the framework and some of its best practices. Bootstrap 3.3.5 was used at first since it was the latest stable version when the implementation started and it proved to be a powerful tool for the front end implementation.

At later stages of the implementation there was some desired functionality that was not achievable through Bootstrap 3.3.5. Further research showed that this was available through Bootstrap 4.0.0 alpha-2. This was not a stable version and some functionality from 3.3.5 had been removed from Bootstrap 4.0.0 alpha-2. But after some consideration the decision was made to move on with Bootstrap 4.0.0 alpha-2 even though some of the existing code had to be rewritten. (Mark Otto, 2016)

Bootstrap is an open source front end framework, a collection of HTML and CSS based templates for forms buttons navigation and much more as well as optional java script plugins. It is also compatible with most of the modern browsers including Firefox, Chrome and Safari.

Online courses at lynda.com were taken to get further understanding of the tool.

Database

When implementing the database MYSQL was used, mainly because we have previous experience using it and thus know it is capable of fulfilling the requirements planned for the database. MYSQL is a relational database management system. A relational database is based on the relational model of data. The data is stored in tables where each row is an instance of data and each column being its attributes. If two or more tables interact they have a logical relationship. (Docs.oracle.com, 2016)

A diagram of the database can be seen in Figure 5.



Figure 5 - Diagram of the database

MYSQL in PHP

There was a necessity to interact with the database through the application and since MYSQL was being used for the database the requirement was to be able to use MYSQL in PHP. The three main APIs for this purpose turned out to be PHP-MYSQL extension, PHP-MYSQLi extension and PDO.

The decision to go with PHP-MYSQLi was based on that the current database uses MYSQL 5.6.17 and neither PHP-MYSQL

extension or PDO fully supports all of its features. There was also no intention so switch database during for the foreseeable future of the applications lifecycle.

PHP-MYSQL extension

This is the predecessor to the PHP-MYSQLi extension and is intended for use with MYSQL 4.1.3 and older. Thus this seemed a little outdated.

PHP-MYSQLi extension

PHP-MYSQL improved extension is intended for use with MYSQL 4.1.3 and newer to fully take advantage of new features in these versions. The most important enhancements from the predecessor PHP-MYSQL extension are:

- Object-oriented Interface
- Support for prepared statements
- Support for multiple statements
- Support for transactions
- Enhanced debugging capabilities
- Embedded server support

As well as the object oriented interface the procedural interface from its predecessor is also available.

PHP-MYSQLi extension

This extension is included in PHP5 and newer.

PDO

PHP Data Objects is a database layer that provides a consistent API for the application regardless of the type of database server the application connects to. This makes it easy if there for any reason is a need to switch database from one type to another. One disadvantage is that it does not fully support all the functionality that PHP-MYSQLi offers eg. multiple statements (Php.net, 2016)

3.1.4. Web server

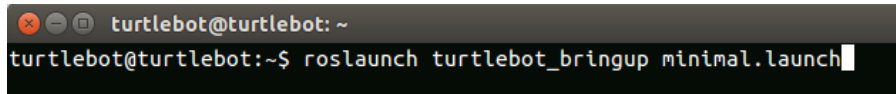
The web server used in this project was Apache 2.4.7. Apache is the world's most popular Web server software according to their own homepage. (Apache.org, 2016)

4. Result

4.1. TurtleBot software

4.1.1. Starting the TurtleBot

To start the TurtleBot, the `turtlebot_bringup` package is used as seen in Figure 6. The `turtlebot_bringup` package has to be run on the on-board computer which is connected to the Kobuki base.

A terminal window with a dark background. The prompt is `turtlebot@turtlebot: ~`. The command entered is `roslaunch turtlebot_bringup minimal.launch`.

```
turtlebot@turtlebot: ~  
turtlebot@turtlebot:~$ roslaunch turtlebot_bringup minimal.launch
```

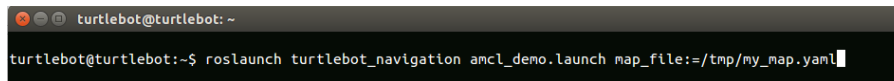
Figure 6 - Starting the TurtleBot

The `turtlebot_bringup` package provides `roslaunch` scripts for starting the TurtleBot base functionality. The base launch file, called ‘`minimal.launch`’, starts the basic node (`kobuki_node`), the `laptop_battery_monitor`, the `robot_state_publisher`, the `diagnostic_aggregator`, and the `robot_pose_ekf`. It also includes `app_manager.launch` which starts the TurtleBot app managers and loads the TurtleBot app list. (Wiki.ros.org, 2016)

4.1.2. Navigation

The TurtleBot relies on Adaptive Monte Carlo Localization (AMCL) for its navigation. AMCL is an algorithm used to localize a robot in a known map. To estimate the robot’s pose (position and orientation), the algorithm begins with requiring an initial external estimation of the robot’s position and orientation on the map. The robot then uses its sensor data to compare the input with the estimated pose on the map. As the robot moves and receives new input data to evaluate, the accuracy of the pose estimation increases. (Se.mathworks.com, 2016)

The `turtlebot_navigation` package (Wiki.ros.org, 2016) takes an `amcl` launch file and a map file as parameters on startup as seen in Figure 7.



```
turtlebot@turtlebot: ~  
turtlebot@turtlebot:~$ roslaunch turtlebot_navigation amcl_demo.launch map_file:=/tmp/my_map.yaml
```

Figure 7 - Starting turtlebot_navigation

The launch file is included in the TurtleBot packages that are installed during the initial setup of the TurtleBot software on the onboard computer. To set the estimated initial pose, the amcl launch file can be edited. As seen in Figure 8 on line 17, 18 and 19 the default x and y values have been set, as well as an angular value to set the TurtleBot's orientation.

```
13 <!-- Map server -->  
14 <arg name="map_file" default="$(env TURTLEBOT_MAP_FILE)"/>  
15 <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />  
16  
17 <arg name="initial_pose_x" default="13.3"/> <!-- Use 17.0 for willow's map in simulation -->  
18 <arg name="initial_pose_y" default="2.545"/> <!-- Use 17.0 for willow's map in simulation -->  
19 <arg name="initial_pose_a" default="-1.000"/>  
20 <include file="$(find turtlebot_navigation)/launch/includes/amcl.launch.xml">  
21   <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>  
22   <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>  
23   <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>  
24 </include>  
25  
26 <include file="$(find turtlebot_navigation)/launch/includes/move_base.launch.xml">  
27   <arg name="custom_param_file" value="$(arg custom_param_file)"/>  
28 </include>  
29  
30 </launch>
```

Figure 8 - Editing the turtlebot_navigation launch file

The values entered in the launch file can be obtained from starting RViz with turtlebot_navigation running. Rviz (ROS Visualization) is a 3D visualizer for displaying sensor data and state information from ROS. Live representations of sensor values can also be displayed. This includes camera data, infrared distance measurements, sonar data, and more. (Sdk.rethinkrobotics.com, 2016)

4.1.3. Creating a map

The map file required for amcl is also created using RViz and turtlebot_navigation along with turtlebot_teleop (Wiki.ros.org, 2016). The turtlebot_navigation is started with a mapping launch file as argument and RViz, being a visualization program, is mostly used to follow the progress of the map creation process. The mapping launch file, "gmapping_demo.launch", is included in the TurtleBot packages installed during the initial setup of the TurtleBot software. When turtlebot_navigation has been started with the gmapping launch file, it is necessary to make the TurtleBot move in the area over which the

map is to be created. This can be achieved by using the turtlebot_teleop package. The turtlebot_teleop package provides launch files for teleoperation with different input devices. For the purposes of this project, the keyboard_teleop.launch file was used. A running instance of turtlebot_teleop with the keyboard_teleop.launch file can be seen in Figure 9.

```

/opt/ros/hydro/share/turtlebot_teleop/launch/keyboard_teleop.launch http://localhost:11311
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 64a2a95e-3559-11e3-aba6-240a641bea79
process[rosout-1]: started with pid [7673]
started core service [/rosout]
process[turtlebot_teleop_keyboard-2]: started with pid [7685]

Control Your Turtlebot!
-----
Moving around:
  u   i   o
  j   k   l
  m   ,   .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2      turn 1

```

Figure 9 - Controlling the TurtleBot with a keyboard and turtlebot_teleop

When the TurtleBot has moved over the entire area over which the map is to be created, the map file can be saved to the onboard computer. This is accomplished using the map_server in combination with the map_saver utilities as seen in Figure 10. The “map_server provides the map_server ROS Node, which offers map data as a ROS Service. It also provides the map_saver command-line utility, which allows dynamically generated maps to be saved to file.” (Wiki.ros.org, 2016)

```

turtlebot@turtlebot: ~
turtlebot@turtlebot:~$ roslaunch map_server map_saver -f /tmp/my_map

```

Figure 10 - Saving a map

Two map files are created with the extensions .pgm and .yaml as seen in Figure 11. The .yaml file is written in YAML which “is a human-friendly, cross language, Unicode based data serialization language” (Netpbm.sourceforge.net, 2016). It contains, among other things, information on where to find the .pgm file. "PGM" is an acronym derived from "Portable Gray Map". A PGM image represents a grayscale graphic image and is a lowest common denominator grayscale file format. (Netpbm.sourceforge.net, 2016)

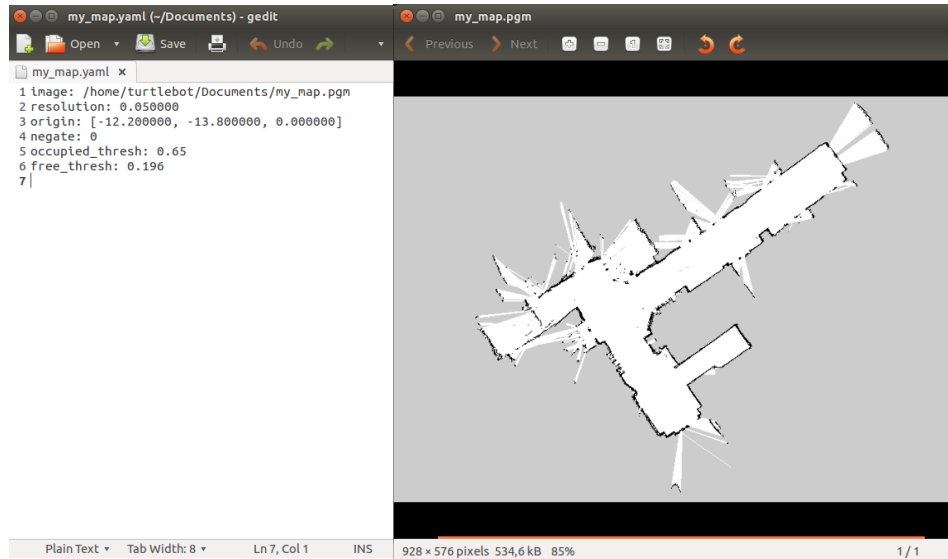


Figure 11 - The .yaml file to the left and the .pgm file to the right

4.1.4. Coffee_bot.py

The python script coffee_bot.py was written by Mark Silliman and is publicly available on GitHub (GitHub, 2016). Mark Silliman is one of the main contributors to learn.turtlebot.com, which is the official tutorial site for the TurtleBot. In one of the tutorials the reader is encouraged to download the coffee_bot.py script.

Early on in the project it was decided to try to find and use prewritten code and incorporate it in the project. The general functionality of the coffee_bot.py script is to get coordinates from a

database and, by using ROS and TurtleBot specific python libraries, send the TurtleBot to those coordinates.

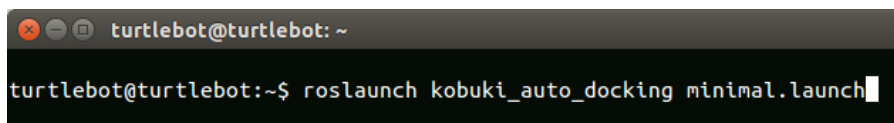
The `coffee_bot.py` script makes use of the “rospy” library to control the TurtleBot. “rospy is a pure Python client library for ROS. The rospy client API enables Python programmers to quickly interface with ROS Topics, Services, and Parameters.” (Wiki.ros.org, 2016) The `coffee_bot.py` script also makes use of some TurtleBot specific libraries to handle information sent from the kobuki base regarding power system events, auto docking actions and general sensor states.

The power system events are used to keep track of the power state of the kobuki base, to decide whether the robot should continue driving or go back to the docking station and charge the battery.

Auto docking

The auto docking actions are used to enable the functionality of the `kobuki_auto_docking` package (Wiki.ros.org, 2016). When the `coffee_bot.py` script receives the information that the power is low and the robot needs to charge, it first sends it to some hard coded coordinates. The coordinates represent a pose approximately one meter away from the docking station. When the TurtleBot reaches the coordinates, the kobuki auto docking tool tries to dock to the docking station and returns a statement regarding if it succeeded or not. The auto docking tool makes use of infrared emitters and receivers on the Kobuki base and on the docking station to steer the robot onto the docking station.

The auto docking tool needs to be started separately for the auto docking to work. Figure 12 shows how the auto docking tool is used.

A terminal window with a dark background and light text. The title bar shows a red close button, a yellow minimize button, and a green maximize button, followed by the text 'turtlebot@turtlebot: ~'. The terminal content shows the prompt 'turtlebot@turtlebot:~\$' followed by the command 'roslaunch kobuki_auto_docking minimal.launch' and a cursor at the end of the line.

```
turtlebot@turtlebot: ~  
turtlebot@turtlebot:~$ roslaunch kobuki_auto_docking minimal.launch
```

Figure 12 - Starting the auto docking tool

The script

```
293 if __name__ == '__main__':
294     delivery_checks = 0 #just for troubleshooting to see how many times we called the server to check for pending coffee
295     try:
296         coffebot = turtlebot_coffee()
297         #keep checking for deliver_coffee until we shutdown the script with ctrl + c
298         while(coffebot.deliver_coffee() and not rospy.is_shutdown()):
299             delivery_checks = delivery_checks + 1
300
301     except rospy.ROSInterruptException:
302         rospy.loginfo("Exception thrown")
```

Figure 13 - The main function of coffee_bot.py

The main function of coffee_bot.py, seen in Figure 13, contains a while loop which call the deliver_coffee() function. The deliver_coffee() function handles all the major functionality of the script. The function is divided into if statements that all contain some part of the principal functionality, like the “if(self.INeedPower())” statement for example. All the if statements return “true” when they have executed and the while loop in the main function loops again.

The python script reads values from a MySQL database through GET requests to a PHP file. The PHP file along with helper files are also publicly available on Mark Silliman’s GitHub page (Silliman, 2016).

4.1.5. TurtleBot server

The files provided by Mark Silliman in the folder “turtlebot-server” (Silliman, 2016) consists of three PHP files and one readme file. The files “config.php” and “db.php” are written for the configuration and setup of a MySQL database. The database used in this project utilizes the table “QUEUE” that is set up by the db.php file but the setup and the configuration of the project database happens elsewhere, as described on page 14.

The file “coffee_queue.php” is what mediates the connection between the python script (coffee_bot.py) and the database. The PHP file contains functions such as push() and pop(), along with some other functions to handle the “QUEUE” table in the database. The python script sends GET requests and the “coffee_queue.php” file reads and returns database values or writes to the database depending on the request.

4.1.6. Modifications of coffee_bot.py and coffee_queue.php

To add functionality, some modifications were made to the coffee_bot.py script. When the TurtleBot has arrived at a destination it will stay there until the “B0” button on the Kobuki base is pressed. To make it continue driving also when a button on the homepage is pressed the additions in Figure 14 were made.

```
84 #if someone is currently making coffee don't move!
85 if(self.cannot move until b0 is pressed):
86     rospy.loginfo("Waiting for button B0 to be pressed.")
87     value = json.load(urllib2.urlopen(self.server_public_dns + "/turtlebot-server/coffee_queue.php?b0"))
88     if(value == 0):
89         self.cannot_move_until_b0_is_pressed = False;
90         time.sleep(2)
91         return True
92
```

Figure 14 - Modifying what it takes to make the TurtleBot continue driving after reaching a destination

The code on line 87, 88 and 89 in Figure 14 was added. The code on line 87 will send a GET request to coffee_queue.php and store the return value in the variable “value”. If the button on the homepage has been pressed the boolean variable “cannot_move_until_b0_is_pressed” will be set to false and the robot will continue to the next destination. To make this work, the additions to coffee_queue.php that can be seen in Figure 15 and 16 were made.

```
53
54 if(isset($_GET['b0'])) {
55     $n = new coffee_queue();
56     $n->b0();
57 }
--
```

Figure 15 - coffee_queue.php receiving a GET request

```
197
198 public function b0() {
199     $result = $this->conn->query("SELECT setting FROM Settings WHERE name='WaitForButton'") or die("Value failed");
200     $this->conn->query("UPDATE Settings SET setting = 1 WHERE name = 'WaitForButton'") or die("Settings insert failed");
201     $row = $result->fetch_array();
202     $result = $row['setting'];
203     echo $result;
204 }
205
```

Figure 16 - Returning the result retrieved from the database

The button on the homepage sets a value in the database. That value is read from the python script and if the value is set the TurtleBot will continue driving. To make the button on the homepage only able to set the database value if the TurtleBot is at a destination, the addition on line 131 in Figure 17 was made to the python script. A GET request is sent to a PHP file that will set a database value when the TurtleBot has reached its goal. That value enables functionality on the homepage to set the other database value, which will tell the TurtleBot to continue on its path.

```

129 if state == GoalStatus.SUCCEEDED:
130     json.load(urllib2.urlopen(self.server_public_dns + "/turtlebot-server/coffee_queue.php?b0"))
131     urllib2.urlopen(self.server_public_dns + "/botbutler/otherPhpFunctions.php?goal_reached")
132     rospy.loginfo("Mooray, reached the desired pose! Press B0 to allow Turtlebot to continue.")
133     #tell Turtlebot not to move until the customer presses B0
134     self.cannot_move_until_b0_is_pressed = True
135     self.count_no_one_needs_coffee_in_a_row = 0 #reset to 0
136     #tell the server that the post was completed
137     data = json.load(urllib2.urlopen(self.server_public_dns + "/turtlebot-server/coffee_queue.php?updateId=" + data["id"] + "&status=complete"))
138

```

Figure 17 - Setting the database value to "location reached"

4.1.7. Turtlepower

To be able to communicate with the TurtleBot through the homepage the robot first had to be started, then the navigation had to be turned on, and the auto docking tool, and then the coffee_bot.py script had to run. This could be achieved by opening terminal windows and executing the commands seen in Figure 18.

```

turtlebot@turtlebot: ~
turtlebot@turtlebot:~$ roslaunch turtlebot_bringup minimal.launch

turtlebot@turtlebot: ~
turtlebot@turtlebot:~$ roslaunch turtlebot_navigation amcl_demo.launch map_file:=/tmp/my_map.yaml

turtlebot@turtlebot: ~
turtlebot@turtlebot:~$ roslaunch kobuki_auto_docking minimal.launch

turtlebot@turtlebot: ~
turtlebot@turtlebot:~$ python modified_coffee_bot.py

```

Figure 18 - Startup sequence. Each in a new terminal window.

To simplify the process of starting the TurtleBot, a shell script was created. The shell script runs each command in a new terminal window and waits ten seconds between each command. Each command is dependent on the previous command running and therefore the wait, sleep(10), was inserted.

To be able to start, restart and stop the TurtleBot from the homepage a small program was written in C++. The program checks for three different files in a specific location. The files will contain either the words “stop”, “start” or “restart”. Depending on the content of the files the program will either kill the processes started by the shell script, run the shell script that starts the robot, or both.

4.2. Application

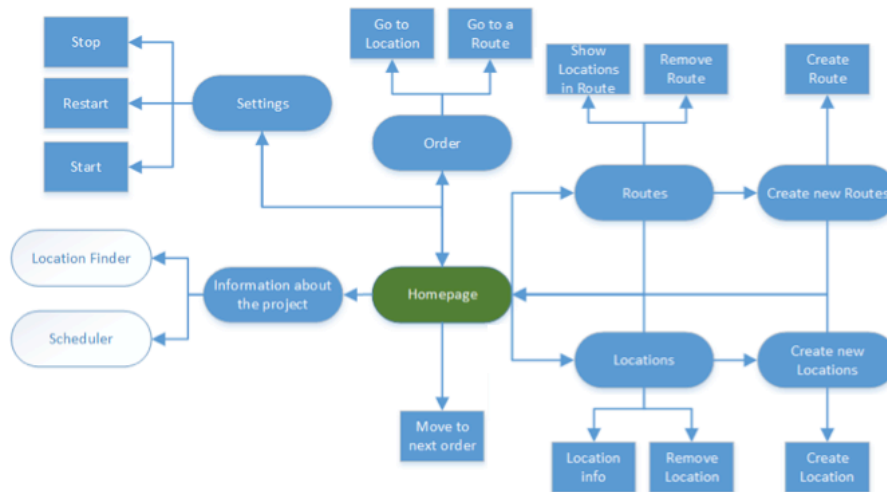


Figure 19 - Site map

4.2.1. Home

The home view is designed to be the starting point of the application and it is possible to return to the home view with a single click from any of the other application views. On the home view there are 6 buttons, 5 of them direct the user to another view and the 6th has functionality added to it but does not change the view.

Buttons:

1. Location List seen in Figure 20
2. Route List seen in Figure 21
3. Order seen in Figure 22
4. Power seen in Figure 23

5. About seen in Figure 24
6. Continue seen in Figure 25

The appearance of the home view may vary depending on the device the user is using.

4.2.2. Location List

Clicking on the first button in the home view opens up the Location List view. From this view all stored locations are presented in alphabetical order. There are 2 buttons for each location, the first button Figure 26 will show the 'X' and 'Y' coordinates for that location and the second button Figure 27 will remove the stored location. Before the location is removed a confirmation popup will show to avoid removing locations by accident. If the location is removed it will disappear from the list.

There is a button on the top Figure 28, clicking this will direct the user to the Add Location view.

Add Location

From the Location List view the user can navigate to the Add Location view. Here the user can add New Locations to the Location List by filling out the 3 fields, Location Name, X and Y then clicking the button Figure 29. A successful addition will be confirmed by a popup and if the addition is unsuccessful the popup will state what the problem is.

Acquiring the X and Y coordinates has to be done through RViz because the functionality to acquire coordinates through the application is not fully working yet.

4.2.3. Route List

Clicking on the second button in the home view opens up the Location List view. From this view all the stored routes are presented by name in the order they were created with the oldest at the top and the newest at the bottom. Clicking on a route will expand it, showing all the locations included in the route. The locations are shown in the order they are intended to be visited in. Clicking on a route will also show a button Figure 27, clicking this will remove the expanded route

from the route list. Before the route is removed a confirmation popup will show to avoid removing routes by accident. If the route is removed it will disappear from the list.

There is a button on the top Figure 30, clicking this will direct the user to the Add Route view.

Add Route

From the Route List view the user can navigate to the Add Route view. Here the user can create new routes by selecting a location in the left box and clicking the ‘>’ button, the location will then be moved to the right box. The right box represents the new route with locations in it in the order they will be visited with the first location at the top. To remove a location from the route being created select the location and click ‘<’. The ‘>>’ and ‘<<’ buttons will add all and remove all locations from the new route. Fill in a name in the Route name box and click the button Figure 29. A successful addition will be confirmed by a popup and if the addition is unsuccessful the popup will state what the problem is.

Adding the same location several times to the route is not possible in the current iteration.

4.2.4. Order

Clicking the third button on the home view will open up the Order view. From this view the user can queue an order for the TurtleBot to move to a stored location or to move through a route stopping at all its locations in the right order.

To queue a location click on the first button Figure 31 and a list of all the stored locations will show, then select the desired destination. A confirmation popup will show before the order is queued to avoid accidental orders.

To queue a route click on the second button Figure 32 and a list of all the stored routes will show, then select the desired route. A confirmation popup will show before the order is queued to avoid accidental orders.

4.2.5. Power

Clicking on the fourth button on the home view will open up the power view. From this view the user can start, stop or restart the BotButler process by clicking on one of the three buttons.

The first (Figure 33) will start the Botbutler process on the TurtleBot.

The second (Figure 34) will stop the Botbutler process on the TurtleBot.

The third (Figure 35) will restart the Botbutler process on the TurtleBot.

There may be a small delay on up to 1 second for the action to start.

4.2.6. About

Clicking on the fifth button on the home view will open up the navigation view. Here the user can read about the project and other useful information. There are 2 buttons directing the user to the Scheduling view and the acquire graphical coordinates view. This views have been placed in the About view because they are not yet fully implemented.

Acquire Graphical Coordinates

Clicking on the first button in the about view will open up the acquire graphical coordinates view. Here a picture of the map generated by the TurtleBot is presented and the user is able to click anywhere on the map and the coordinates for that specific location will be generated.

This functionality is not fully implemented since the coordinates generated are not perfectly matched with coordinates generated from RViz.

Scheduling

Clicking on the second button in the about view will open up the scheduling view. Here the user can set up a schedule for the TurtleBot to go to a location or go through a route at any given time and day.

This functionality is not fully implemented since the time and day is not stored on or executed by the TurtleBot.

4.2.7. Continue

The sixth button on the home view will not change the view but will make the TurtleBot move to its next order. Before the TurtleBot moves to its next order a popup will show for confirmation. If the TurtleBot has not reached its destination yet the popup will tell the user to wait for the TurtleBot to reach its destination first.

4.2.8. Header

At the top of every view there is a navigation bar that will open up the home view.



Figure 20 - Location list



Figure 22 - Order



Figure 24 - About



Figure 26 - Informaiton

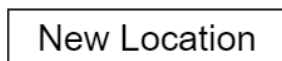


Figure 28 – New location

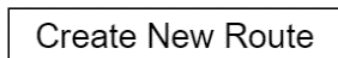


Figure 30 - Create new route



Figure 32 - Go to a route



Figure 34 - Stop



Figure 21 - Route list



Figure 23 - Power



Figure 25 - Continue



Figure 27 - Remove

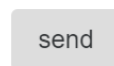


Figure 29 - Send



Figure 31 - Go to location



Figure 33 - Start



Figure 35 - Restart

5. Conclusion

5.1. Application Functionality

Most of the planned features are implemented and have passed testing on several devices and are thus considered done and fully working. This functionality was the most prioritized and is sufficient to control the TurtleBot from the application as requested. However the feature to create a route could need some rework as some of its functionality might have a few bugs even though it is working and currently considered as done.

There are two features (acquire graphical coordinates and scheduling) that never got fully implemented but got included in the application anyway (temporarily available through the About view) because they had passed the concept stage and started implementation already. Because of time limitation the implementation of these features had to be aborted. These were never highly prioritized features but would be classed as “nice to have”.

5.2. Application Design

The design also got to suffer some the time limitations however maybe not as much as the functionality. Overall the design process was mainly driven by the developers taking inspiration from previous experience other applications and examples. The main motivation to this approach was that it would greatly reduce the implementation times because of the reduced time consumptions of the decision process. This felt like a successful approach especially since the development did not occur with close access to the users. During the development it was considered more valuable to have access to the TurtleBot than to develop close to the customer.

The design of the interface and the interaction with the application has the clean and simple feeling that was planned. The design lacks some testing from users outside of the development team and consequently some additional rework has to be accounted for in future iterations.

More explanatory text or feedforward is one of things that never got implemented solely because of time restraints. Some improvements to navigation such as heading back to the homepage after ordering the TurtleBot to go somewhere would have been desirable but were also discarded because of the time restraints. The main thing the design is lacking is testing from a 3rd party user that might give insight to additional flaws in the design. However the application has been tested and feels comfortable in several different screen resolutions

Some unnecessary and nonfunctional buttons in the Route List view also never got removed after the decision was made that that specific functionality was redundant, these were placed out mainly because some of the design was being based on speculation.

5.3. Robot Functionality

The functionality of the robot is mostly handled through `coffee_bot.py`. The project team added a few modifications to make it better fit into the project. The original developer of the `coffee_bot.py` script had more experience with developing for the TurtleBot than the project members. Since it had the functionality that was needed for the project it was decided to use that script and focus on learning web development instead of learning ROS and Python development.

5.4. Problem Statements

5.4.1. What already available functionality can be used in this project?

The tutorial site at learn.turtlebot.com had a lot of the functionality that was needed in the project. The python script that handled the communication between the web site and the TurtleBot was an essential part of the required functionality.

The web site was developed with bootstrap (Mark Otto, 2016), which is an HTML, JavaScript and CSS framework.

The functionality for getting coordinates when clicking on the map in the Acquire Graphical Coordinates view was downloaded from Emanuele Feronato's homepage (Emanuele Feronato, 2006).

5.4.2. How can location information be presented through the web application in a way that makes a user able to control the TurtleBot?

The web application was developed in a way so that when a location is added, it is up to the user to give it a name that is understandable. When adding locations the user have to know the coordinates of the location to be added. The functionality for finding coordinates were supposed to be in the Acquire Graphical Coordinates view, which was not completely developed. This means that the user will have to know how to use the TurtleBot tools, especially RViz, to find the coordinates and to be able to add a new location.

5.4.3. How can coordinates be passed from a web application in a way that the TurtleBot can map them to its internal navigation system?

The library "rospy" used in Mark Sillman's python script, `coffee_bot.py`, contains a class called `MoveBaseGoal()` (Docs.ros.org, 2016) which takes coordinates as arguments and sends the TurtleBot to the corresponding location. The process of sending them from the web application involves the use of a database where the coordinates are stored.

6. Terminology

App	The developed application
Application	The developed application
BotButler	The entire project, including the application and the TurtleBot software
Coordinates	Values instructing the TurtleBot 2 where to go
Customer	The people at Omegapoint giving the assignment
Developers	The two students, Jorge Alas and Anders Holm working on the project
Designers	see developers.
Order	Instructions to the TurtleBot 2 to go to a specific location
RViz	Software used for creating maps and gathering coordinates.
TurtleBot	see TutleBot 2
TurtleBot 2	the Robot used during the project
User	Workers at the Omegapoint offices
View	A single page on the web application

References

Apache.org. (2016). Welcome to The Apache Software Foundation!. [online] Available at: <http://www.apache.org/> [Accessed 11 May 2016].

Docs.oracle.com. (2016). A Relational Database Overview (The Java™ Tutorials > JDBC(TM) Database Access > JDBC Introduction). [online] Available at: <https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html> [Accessed 9 Jun. 2016].

Docs.ros.org. (2016). move_base_msgs: move_base_msgs::msg::_MoveBaseGoal::MoveBaseGoal Class Reference. [online] Available at: http://docs.ros.org/diamondback/api/move_base_msgs/html/classmove_base_msgs_1_1msg_1_1__MoveBaseGoal_1_1MoveBaseGoal.html [Accessed 20 May 2016].

Emanuele Feronato. (2006). Click image and get coordinates with Javascript. [online] Available at: <http://www.emanueleferonato.com/2006/09/02/click-image-and-get-coordinates-with-javascript/> [Accessed 20 Mar 2016].

Garrett, J. and Garrett, J. (2005). Ajax: A New Approach to Web Applications | Adaptive Path. [online] Adaptivepath.org. Available at: <http://adaptivepath.org/ideas/ajax-new-approach-web-applications/> [Accessed 2 Feb. 2016].

GitHub. (2016). markwsilliman/turtlebot. [online] Available at: https://github.com/markwsilliman/turtlebot/blob/master/coffee_bot.py [Accessed 15 May 2016].

GitHub. (2016). markwsilliman/turtlebot-server. [online] Available at: <https://github.com/markwsilliman/turtlebot-server/> [Accessed 15 May 2016].

jquery.org, j. (2016). jQuery. [online] JQuery.com. Available at: <https://jquery.com> [Accessed 22 Apr 2016].

Kniberg, H. and Skarin, M. (2010). Kanban and Scrum. [S.l.]: C4Media, Inc.

Mark Otto, a. (2016). Bootstrap · The world's most popular mobile-first and responsive front-end framework.. [online] Getbootstrap.com. Available at: <http://getbootstrap.com/> [Accessed 19 Apr 2016].

Mark Otto, a. (2016). Migrating to v4 · Bootstrap. [online] V4-alpha.getbootstrap.com. Available at: <http://v4-alpha.getbootstrap.com/migration/> [Accessed 19 Apr. 2016].

Netpbm.sourceforge.net. (2016). PGM Format Specification. [online] Available at: <http://netpbm.sourceforge.net/doc/pgm.html> [Accessed 17 May 2016].

Python.org. (2016). Welcome to Python.org. [online] Available at: <https://www.python.org/about/> [Accessed 12 May 2016].

Php.net. (2016). PHP: Overview - Manual. [online] Available at: <http://www.php.net/manual/en/mysqli.overview.php> [Accessed 25 Apr 2016].

Ros.org. (2016). ROS.org | About ROS. [online] Available at: <http://www.ros.org/about-ros/> [Accessed 12 May 2016].

Saffer, D. (2010). Designing for interaction : creating innovative applications and devices.

Sdk.rethinkrobotics.com. (2016). Rviz - sdk-wiki. [online] Available at: <http://sdk.rethinkrobotics.com/wiki/Rviz> [Accessed 16 May 2016].

Se.mathworks.com. (2016). Localize TurtleBot using Monte Carlo Localization - MATLAB & Simulink Example. [online] Available at: <http://se.mathworks.com/help/robotics/examples/localize-turtlebot-using-monte-carlo-localization.html> [Accessed 14 May 2016].

Silliman, M. (2016). markwsilliman (Mark Silliman). [online] GitHub. Available at: <https://github.com/markwsilliman> [Accessed 12 Feb 2016].

Trello.com. (2016). Trello. [online] Available at: <https://trello.com/> [Accessed 12 May 2016].

W3schools.com. (2016). CSS Tutorial. [online] Available at: <http://www.w3schools.com/css/> [Accessed 9 Jun. 2016].

W3schools.com. (2016). Introduction to HTML. [online] Available at: http://www.w3schools.com/html/html_intro.asp [Accessed 9 Jun. 2016].

Wiki.ros.org. (2016). Documentation - ROS Wiki. [online] Available at: <http://wiki.ros.org/> [Accessed 14 Apr 2016].

Wiki.ros.org. (2016). kobuki_auto_docking - ROS Wiki. [online] Available at: http://wiki.ros.org/kobuki_auto_docking [Accessed 5 May 2016].

Wiki.ros.org. (2016). map_server - ROS Wiki. [online] Available at: http://wiki.ros.org/map_server [Accessed 2 May 2016].

Wiki.ros.org. (2016). Robots/TurtleBot - ROS Wiki. [online] Available at: <http://wiki.ros.org/Robots/TurtleBot> [Accessed 16 Apr 2016].

Wiki.ros.org. (2016). rospy - ROS Wiki. [online] Available at: <http://wiki.ros.org/rospy> [Accessed 22 Apr 2016].

Wiki.ros.org. (2016). turtlebot/Tutorials/indigo/Turtlebot Installation - ROS Wiki. [online] Available at: [http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot Installation](http://wiki.ros.org/turtlebot/Tutorials/indigo/Turtlebot%20Installation) [Accessed 20 Mar 2016].

Wiki.ros.org. (2016). turtlebot_bringup - ROS Wiki. [online] Available at: http://wiki.ros.org/turtlebot_bringup [Accessed 20 May 2016].

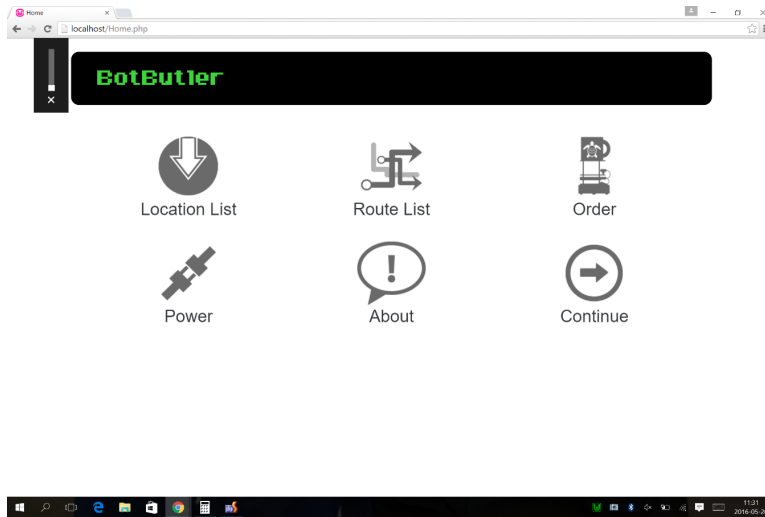
Wiki.ros.org. (2016). turtlebot_navigation - ROS Wiki. [online] Available at: http://wiki.ros.org/turtlebot_navigation [Accessed 12 May 2016].

Wiki.ros.org. (2016). turtlebot_teleop - ROS Wiki. [online] Available at: http://wiki.ros.org/turtlebot_teleop [Accessed 16 May 2016].

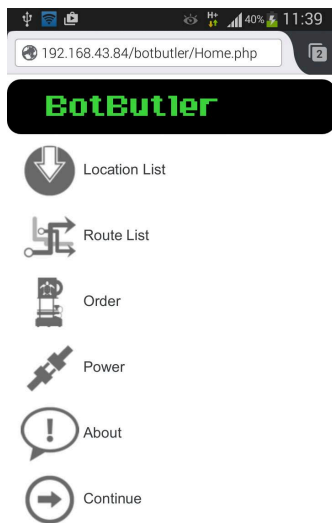
Appendix A: Web site screen shots

Home View

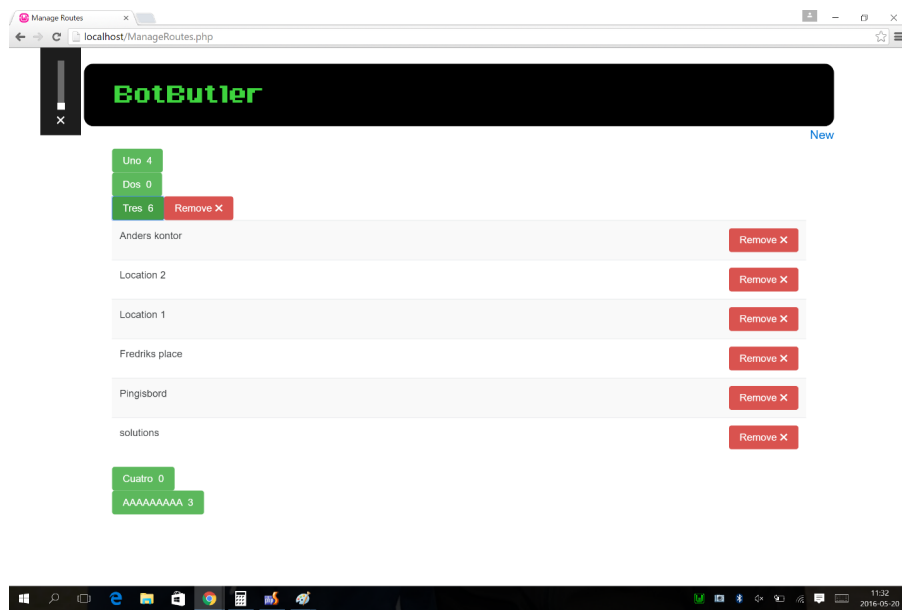
Laptop



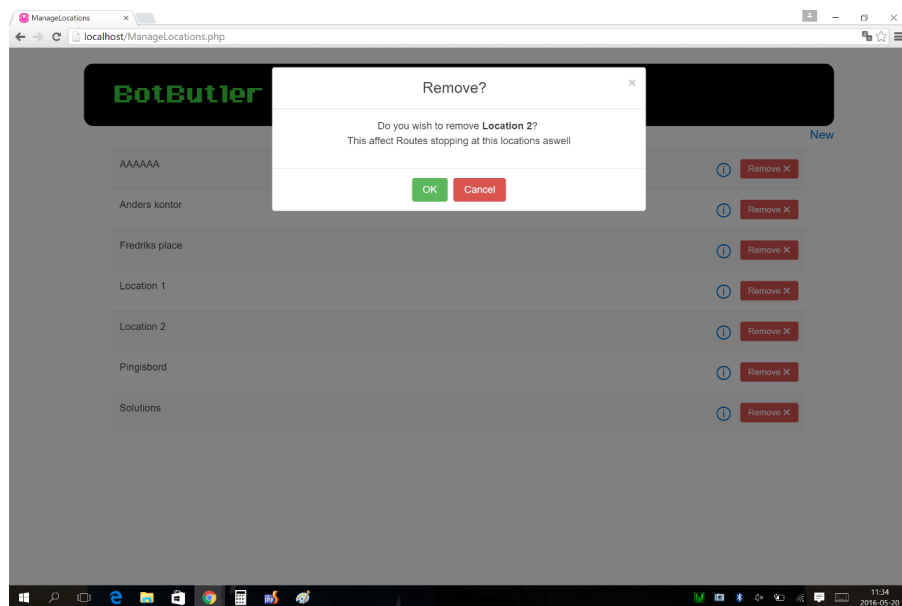
Smartphone



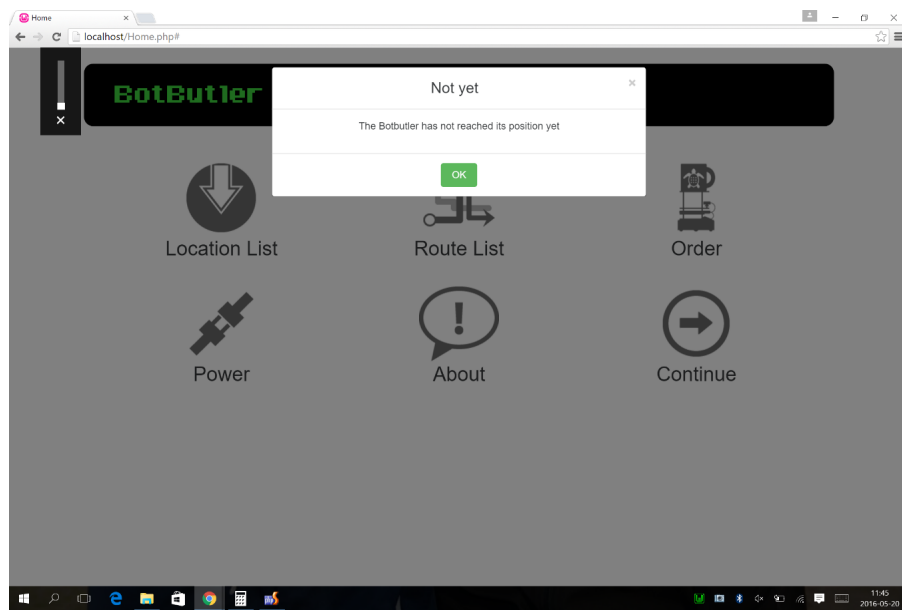
Exapanded routes list



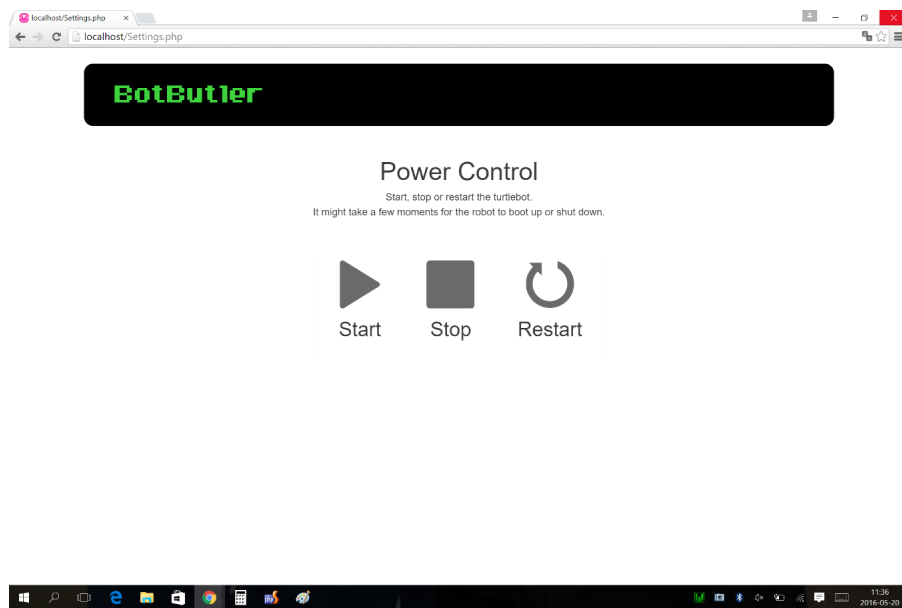
Remove confirmation popup



Continue error popup



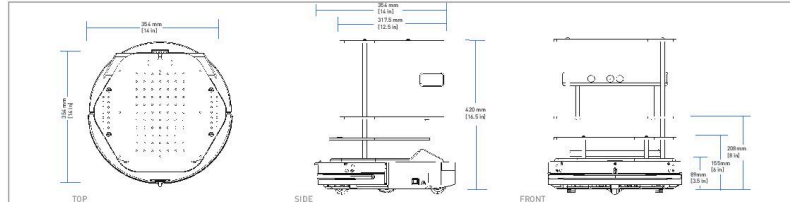
Power



Appendix B: TurtleBot specification sheet

TurtleBot™

PERSONAL ROBOTICS PLATFORM



TECHNICAL SPECIFICATIONS

SIZE AND WEIGHT		
EXTERNAL DIMENSIONS (L x W x H)	354 x 354 x 420 mm (14.0 x 14.0 x 16.5 in)	
WEIGHT	6.3 kg (13.9 lb)	
WHEELS (Diameter)	76 mm (3 in)	
GROUND CLEARANCE	15 mm (0.6 in)	
SPEED AND PERFORMANCE		
MAX. PAYLOAD	5 kg (11 lb)	
MAX. SPEED	0.65 m/s (2.1 ft/s)	
MAX. ROTATIONAL SPEED	180°/S	
BATTERY AND POWER SYSTEM		
STANDARD BATTERY	2200 mAh Li-Ion	
EXTENDED BATTERY	4400 mAh Li-Ion	
USER POWER	5 V and 19V (1A), 12 V (1.5A), 12V (5A)	
SENSORS		
3D VISION SENSOR (ASUS Xtion PRO LIVE)	Color Camera: 640px x 480px, 30 fps.	Depth Camera: 640px 480px, 30 fps
ENCODERS	25700 cps	11.5 ticks/mm
RATE GYRO	110 deg/s Factory Calibrated	
AUXILIARY SENSORS	3x forward bump, 3x cliff, 2x wheel drop	
COMPUTER (subject to change)		
MEMORY	4 GB	
SCREEN	11.6in (1366x768)	
PROCESSOR	Intel Core i3-4010U	
GRAPHICS	Intel® HD Graphics	
INTERNAL HARD DRIVE	500 GB	
WIFI	802.11n	
OPTICAL DRIVE	Not Applicable	

Contact us today for pricing and a free 30 minute technical assessment: 1-800-301-3863

© 2014 Clearpath Robotics, Inc. All Rights Reserved. Clearpath Robotics and clearpathrobotics.com are trademarks of Clearpath Robotics. All other product and company names listed are trademarks or trade names of their respective companies.



LUND
UNIVERSITY

Series of Bachelor's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2016-522

<http://www.eit.lth.se>