Table of Contents

1. Willy

1.1. Project background and progress

In 2015 a 'waste competition' was organized at a primary school, with the aim to provide the pupils of this primary school with a solution to the question 'how to keep the Grote Markt in Zwolle clean'.

Eventually, six students won this competition with their idea to keep the Grote Markt clean with the help of a 'cleaning robot (Willy)'. An important aspect of Willy had to be that he would be able to interact with bystanders and thus influence them on their waste behavior. For example, Willy might point out to people that it would be better to deposit waste in a waste bin instead of throwing it on the floor; this would enable Willy to influence people positively. In short, Willy must be able to act in a corrective as well as a preventive (i.e. interaction) manner.

The primary school contacted the township of Zwolle to see if the 'Willy concept' could be realized. The township of Zwolle reacted positively to the Willy concept and then looked into whether it could involve a partner who could take on Willy's realization.

Eventually the township of Zwolle found 'The Art of Robotics (TAoR)'; a foundation that works to increase the awareness of robot technology in today's society. ' The team of TAoR responded enthusiastically to this proposition and subsequently made a plan on how they could best realize Willy. Ruud van der Burg contacted Hogeschool Windesheim on behalf of TAoR with the proposal to see if it would be possible to let students (of different disciplines) work at Willy. Windesheim has agreed to this, with which the realization process of Willy started.

In order to be able to actually realize Willy, the development process was chopped into various iterations, with the result that different training disciplines would start working on the creation of Willy. In principle, students from IPO (Industrial Product Design) worked on a graphic design of Willy, which would make it clear what Willy should look like. They have also been involved in making technical drawings with regard to the frame to be realized and other design aspects.

The students (mechanical engineering), of the second iteration, have occupied themselves with the realization of a moving chassis for Willy. In the end they decided that the undercarriage of an electric wheelchair would be the best option and so they purchased and prepared this for further developments.

During the third iteration, HBO-ICT students have been working on realizing the autonomous functionality of Willy. An important characteristic of Willy is, that he must be able to function completely autonomously on the Grote Markt in Zwolle (this is a further feature of functionality that Willy should also be able to function in buildings). This meant in the first instance that the project group had to deal with determining the right kind of sensors. Without sensors it would be impossible to recognize objects and therefore avoid them. These sensors eventually had to be linked to Willy's Operating System (ROS). This project group has also been involved in writing an algorithm, which Willy will be able to drive a fixed pattern within a defined area.

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

2. Publicity

2.1. Stentor

Jesse Bouwman got in touch with a redactor of the Zwolse courant de Stentor. After some email contact we planned a meeting to have an interview about Willy and the background of Willy. Joep Boerboom came and we have had some nice conversations about Willy and the future of our robot. The article and the photo of Willy can be found on the website of de Stentor:

https://www.destentor.nl/zwolle/studenten-windesheim-maken-pratende-schoonmaakrobot~aa713925/

2.2. Windesheim newspaper Win'

After the article from de Stentor we got response from the Win' newspaper by email. The Win' crew read our article and they became intrested to write something about Willy too. After some email contact we planned a meeting and gave an interview. Also the photograph took some pictures of our Willy. The article can be read on the following web page.

https://www.hogeschoolkrantwin.nl/2018/05/17/afvalrobot/

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu

- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design

- (2017/1) Base & Functionalities
- (2017/2) Research

3. Sponsors

This document contains a list of all the companies sponsoring Willy.

3.1. Windesheim

Windesheim provides students and financial support for the project.

[image]

3.2. The Art of Robotics

The art of Robotics foundation is the product owner, financial support and a way of getting new sponsors.

[image]

3.3. Sick

Sick provided the project with a very expensive LIDAR sensor. Sick provided this in cooperation with our product owner.

[image]

3.4. Multibat

The battery company Multibat showed via our product owner interest in our project. After a conversation with the company they proposed a deal. Multibat now provides six batteries to us.

[image]

3.5. Automaterialenland

When the batteries were sponsored by Multibat, there were unfortunately no cables to connect the batteries. Then we got Automaterialenland as contact to provide free cables.

[image]

3.6. Koers Polyestertechniek

Our product owner found a company for us to create the outer plating with. After creating a foam mold, we drove it to Koers in Nieuwleusen and made the outer skin of Willy from fiberglass.

[image]

3.7. Multimate Vollenhove

Multimate provided the project with building materials and small accessories.

[image] :numbered: :toc: :toclevels: 5 :icons: font

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

4. Willy Git setup

Commands used to set up the git repos in willy.

```
cd /home/willy/Documents
git clone https://github.com/ArtOfRobotics/WWEB
cd WWEB
git checkout -b test
git push -u origin test
-   Username: willythegarbagedisposal@gmail.com
-   Password:
cd ..
git clone https://github.com/ArtOfRobotics/WTGD
cd WTGD
git checkout -b test
git push -u origin test
-   Username: willythegarbagedisposal@gmail.com
-   Password:
```

4.1. Github synchronisation

A crontab has been created which ensures that Willy stays up-to-date with Github. Because Willy does not have a fixed external-IP or domain, it is unfortunately not possible to use Github Webhooks. That is why a crontab job has been set up, which is configured as follows:

```
sudo crontab -e

*/1 * * * * su -s /bin/sh -c '/home/willy/Documents/WTGD/bin/gitsync.sh ' >>
/home/willy/Documents/logs/git.log 2>&1
```

The script used for syncing github can be found on Github. The main purpose of the script is to do the following:

- Update the WTGD code for driving
- Set permissions for script folder
- (In the future): automatic building
- Update the WWEB code for web platform
- Set permisions for specific folders

Currently this line creates a pull of the test branch, later it is desirable to change this to the master branch for production code. Logs can be found in: /home/willy/Documents/logs/git.log

To track the status of git sync live use the following command

```
tail -f /home/willy/Documents/logs/git.log
```

4.2. Github Repo Willy

Can be found in /home/willy/Documents/WTGD

4.3. Github Repo Web platform

Can be found in /home/willy/Documents/WWEB

4.4. Build

Web interface For building the web interface refer to the Development Guide of the Web section.

Willy For building the software used by Willy refer to the Development Guide of the Technical section.

4.5. Accounts

All the accounts coupled to the email adress: willythegarbagedisposal@gmail.com can be found on the Google Drive

References

- link: https://artofrobotics.github.io/WillyWiki/Documents/#_willy_git_setup

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

5. Operation system of Willy

5.1. Configuration

Currently, Willy is operating using ‘Linux Ubuntu’ as operation system. The ROS-framework is used for centralized communication between nodes.

For more information about nodes and the working of it, see our ROS Introduction wiki page.

Nodes are referred as different software/hardware components of Willy: think of; GPS, sensors, compass and software functions. Some nodes may require

```
dependent-ROS-packages
```

To execute and compile software nodes, these packages are required. Both ROS and dependencies require the same version. The current version of ROS is

```
‘ROS-kinetic’
```

5.2. Current installed packages

The first step in updating the OS is to determine which packages and dependencies are installed. Because previous teams made a lot of changes to Willy, not all Linux packages may still be required. To list the manually installed packages, the following command was used.

```
_comm -23 <(apt-mark showmanual | sort -u) <(gzip -dc
/var/log/installer/initial-status.gz | sed -n 's/^Package: //p' | sort
-u)_
```

The following packages where manually installed;

```
brightness-controller
dhcpcd5
dotnet-sdk-2.0.0
git
google-chrome-stable
htop
nmap
openssh-server
pgadmin3
postgresql
python-pip
ros-kinetic-desktop-full
ros-kinetic-joystick-drivers
ros-kinetic-rosbridge-server
ros-kinetic-rosserial-python
ros-kinetic-move-base
ros-kinetic-hector-mapping
ros-kinetic-sicktim
ros-kinetic-opencv
ros-kinetic-openni
ros-kinetic-rosserial-server
ros-kinetic-teleop-twist-joy
ros-kinetic-teleop-twist-keyboard
ros-lunar-catkin
screen
vsftpd
x11vnc
xfce4
xrdp
```

Some of the above packages are required to compile and execute the WTGD code that is available from GIT. These packages are listed bold. Other packages may be required for the web platform or may have another goal than compiling and executing ROS code. In this project there will be major changes in the web platform, because some of the code will be changed. Dependencies will be determined during the development of the code. Other packages are explained in documentation that will be available with the final delivery.

5.3. Determined packages for Ubuntu 16.04 and ROS-kinetic

To create a clear view about the current ‘WTGD’ code that Willy contains, and how this works on Ubuntu 16.04 with ROS-Kinetic, a test environment was created. None of the previous listed packages where installed, only Ubuntu 16.04 was installed and the ROS-framework on top of this. The code was not able to build successful in this test environment, however based on the error messages, dependencies where determined. Every time a dependency was missing, the error massage was inspected. We concluded that the following packages are required to execute the ‘WTGD’ code on Willy.

```
ROS-kinetic-desktop-full
Screen
ROS-kinetic-rosserial
ROS-kinetic-rosserial-arduino
ROS-kinetic-rosapi
```

5.4. Install dependencies

To install above dependencies, the following commands are required.

Install ROS:

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

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80
--recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

```
sudo apt-get update
sudo apt-get install -y ros-kinetic-desktop-full
sudo rosdep init
rosdep update
```

Link the ROS framework to the ubuntu bash:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Start a new bash prompt and run (test):

```
roscore
```

Install ROS-dependencies:

```
sudo apt-get install Screen
sudo apt-get install ros-kinetic-rosserial
sudo apt-get install ros-kinetic-rosserial-arduino
sudo apt-get install ros-kinetic-rosapi
sudo apt-get install ros-kinetic-rosbridge-server
```

Give user permission to access USB ports:

```
sudo usermod -a -G dialout willy
```

With the above installation changes made to Ubuntu, the current WTGD code was able to run.

5.5. Devices network

The following information can be used to connect with the 'devices' wifi-network at Windesheim.

```
IP-adres: 145.44.211.32
Subnet mask: 255.255.255.192
Default gateway: 145.44.211.1
DNS: 8.8.8.8
```

An connection can only be made with the following MAC address. If this is changed, please ask the support-desk of Windesheim to change the MAC-address.

MAC: 10:4a:7d:21:f2:d2

The configuration on Willy is as shown below

[ipconfig] | *media/Ipconfig.png*

[wifisettings] | *media/wifisettings.png*

5.6. Leap

Download the Leap package from https://www.leapmotion.com/setup/desktop/linux

Save the package and unzip the 64-bit version

Go to the folder the .deb was unzipped to and run `sudo dpkg --install Leap-*-x64.deb`

6. Remote access

6.1. RDP

The following configuration was used when RDP was deployed by using XRDP and XFCE4

```
sudo apt-get install xrdp
sudo apt-get install xfce4
```

Edit .Xsesion file in home directory

```
echo xfce4-session >~/.xsession
```

Edit XRDP configfile: “/etc/xrdp/starwm.sh” for using XFCE4

```
#!/bin/sh
if [ -r /etc/default/locale ]; then
  . /etc/default/locale
export LANG LANGUAGE
fi
startxfce4
```

Fix the Tab button by editing:

```
~/.config/xfce4/xfconf/xfce-perchannel-xml/xfce4-keyboard-shortcuts.xml
```

Replacing

```
<property name="&lt;Super&gt;Tab" type="string" value="switch_window_key"/>
```

By

```
<property name="&lt;Super&gt;Tab" type="string" value="empty"/>
```

6.2. Openssh server

Install openssh

```
sudo apt-get install openssh-server
```

Enable ssh on boot

```
sudo systemctl enable ssh
```

References

- link: https://artofrobotics.github.io/WillyWiki/Archive/Research/Alternative-interaction.adoc

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

[Build Status]

7. Willy Wiki

This wiki is setup to increase the transferability of the project. Everything you need to start is documented here. Detailed documents are referenced trough the wiki if you need more information.

7.1. Introduction

The wiki is setup using AsciiDoc, TravisCI & Github pages. In practice we use AsciiDoc as source code, TravisCI to convert to html/pdf and Github Pages for publishing the website. This is visually shown in the image below, this is taken from the tutorial we followed.

[AsciiDoc to Github Pages with Travis and Docker Asciidoctor] |

http://mgreau.com/posts/images/cover-asciidoc-ghpages.svg

7.2. Why AsciiDoc

Why AsciiDoc is chosen as our markdown language. During the search for a good Wiki tool, we eventually stumbled upon Github Pages. Github pages in intended to automatically publish markdown for you as a html site and you can add more functions by Using Jekyll. However quickly limitations in Markdown where found and the Jekyll implementation made if far more complex due to different plugins. That's why it is decided to use AsciiDoc at it's raw form.

This is directly the main advantage of using Markdown, while you can use plugins it's main functionality makes it the perfect language for creating a Wiki. In terms of markdown languages you can follow this list as a rule of thumb:

- Markdown (MD)
  - Is the most simplest markdown language out there, but is also it's main weakness
- AsciiDoc (Adoc)
  - Is more versatile in the basics and much more rich in terms of formatting and plugins
- LaTex (Tex)
  - Is more professional focussed and contains a lot of functions at its core where in other markdown languages you need plugins

This should also clarify the reason that AsciiDoc is chosen as the source language of this Wiki. [Markdown vs AsciiDoc]

7.3. How the Wiki is setup using AsciiDoc with TravisCI and Github Pages

The wiki is setup fairly easy, especially when you know your way around Github and TravisCI. So it is important to read into these topics if you don't know what these tools mean. And for AsciiDoc you'll learn it along the way as we did.

7.4. Conversion

As told in the [introduction] AsciiDoc is used as a source language, which then can be converted to whatever format you like. Most commonly HTML and PDF.

7.4.1. Travis

Travis is setup to convert all documents recursively to HTML and to PDF.

Setup

Before you can use Travis you must give travis acces to the repository, this is already done using Willy's Github account. The following environment variables must be set for the script to work

properly.

[env]

Config

The config file used by TravisCI is travis.yml. If you are not familiar with Travis it basically asks you for the following:

- Some config elements (Setup)
  - as of what kind of acces methods and what services you'd like to use
- What to do before installation (Before)
  - Defined under before_installation
- What script to run (Execution)
  - Defined under script:
- What to do after the script has run (After)
  - Defining after_error:, after_failure:, after_success:

For this script a Docker container is used specially made for Asciidoctor, the tool used for AsciiDoc conversion. You do not need any docker knowlegde to use this script because it uses a readymade Docker container. In this container the asciidoctor commands are executed.

Currently the Travis Config is setup as follows:

Setup:
Use of docker and sudo acces

```
sudo: required
```

services: - docker **Before:**
Make a output directory and pull the readymade Docker container.

```
before_install:
  - mkdir -p output
  - docker pull asciidoctor/docker-asciidoctor
```

Execution:
Here is where all the documents are converted what basically comes down to this command:

```
asciidoctor -a allow-uri-read *.adoc
```

With docker it looks like this in the travis.yml

```
script:
  - docker run -v $TRAVIS_BUILD_DIR:/documents/ --name asciidoc-to-html
asciidoctor/docker-asciidoctor asciidoctor -a allow-uri-read **/*.adoc
  - docker run -v $TRAVIS_BUILD_DIR:/documents/ --name asciidoc-to-html-root
asciidoctor/docker-asciidoctor asciidoctor -a allow-uri-read *.adoc
```

Because these commands do not allow for recursive generation more than one folder deep some more lines are added to make sure the Archive folder is converted. A better fix stil need to be implemented.

If the build is failing probably a root heading is used somewhere. This can cause conflicts with the sidebar configuration and is only used in the welcome document.

```
= This is a root heading
== This heading should be used troughout the wiki for the main chapters
```

After:
Your general logging

```
after_error:
  - docker logs
after_failure:
  - docker logs
```

The publishing to Github Pages

```
after_success:
  - find . -name '*html' | cpio -pdm output ;
  - find . -name '*png' | cpio -pdm output ;
  - cd output ;
  - git push
```

Here the output folder contains all the converted documents in html files, but still the images need to be copied, else the images would not be shown. Everything that is copied into the output folder is then pushed to the gh-pages branch on GitHub. As you can see in the travis.yml used some more actions are done by Travis to ensure everything works properly.

For the sidebar to scale correctly we had to manualy add the toc classes because we do not use Docbook (yet). Also see [Recursive replace].

```
  - find -name "*.html" -exec sed -i 's/class="article"/class="article
toc2 toc-left"/g' {} +
```

For conversion from Word to AsciiDoc you can use Pandoc to convert word documents or any other format to AsciiDoc fairly easy. The following command can then be used after Pandoc is installed:

```
pandoc -f "input.docx" "output.adoc"
```

To convert all documents recursively in the current folder you can use the following script: (Windows) [source, BATCH

```
for /r %%v in (*.docx) do pandoc -f "%%v" "%%v.adoc"
```

7.5. Publishing

The end result of the Travis script is a folder with html files which can then be hosted on any server even offline.

7.5.1. Github Pages

To do this we use Github Pages, as this is a free service and is perfect for hosting a static html site. It also helps keeping a history of changes.

This is configured as following: . Go to Github, WillyWiki Settings page . Scroll down to Github Pages . Set branch to 'gh-pages' and you are done

[github]

7.6. Further reading

The Asciidoctor wiki is a good source, you also might notice the familiar look https://asciidoctor.org/docs/user-manual/#introduction-to-asciidoctor

A hardcoded sidebar is used to avoid using Docbook, it might be worth to take a look at this format as it is being used by the official sources as well. The main disadvantage of this is that it makes the Wiki one long HTML page with a large index, however for PDF export this would be fabiolous. See Asciidoctor Wiki as an example.

References

- [Tutorial] Maxime Gréau. Convert AsciiDoc to HTML/PDF & publish to GitHub Pages with Travis CI and Asciidoctor Docker containers
- [[[Markdown vs AsciiDoc]]] Asciidoctor, Markdown vs AsciiDoc
- [[[Recursive replace]]] Stackoverflow, Recursive replace :numbered: :toc: :toclevels: 5 :icons:

font

8. Background

The original design is made by students of Industrial Product Design. One of the first things they produced was a morphological chart. The chart is made to create an overview of all options and functions of the robot.

[morph] | *media/morph.jpg*

With the morphological chart in mind they started brainstorming and made sketches from the ideas they had.

[sketches] | *media/sketches.jpg*

One of the early designs is the one below. In this design a large tv screen is already installed. The idea of a butler like robot came to mind. Willy has to be polite, informative and have a strong character.

[early] | *media/early.jpg*

Because of the complexity the idea above was not realistic to make as a prototyping project. This is one of the reasons why they made a less complex version which certainly is realistic to make.

[willydesign] | *media/willydesign.jpg*

- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

9. Design guide

9.1. Prerequisite

SolidWorks is a 3D CAD software system. The program is used to design and test parts. SolidWorks has a render extension which makes it possible to make picture like images from the 3D models.

Student edition: https://store.solidworks.com/studentstore/default.php?command=Step1

Download: https://www.solidworks.com/sw/education/SDL_form.html

For a student license: Mechanical Engineering has licenses which can be used. Ask at T1 for more information.

Install SolidWorks following their install guide.

It might be useful to look at http://www.3dleerlijn.nl/solidworks-leren-gebruiken/ for learning the basics.

10. Technical

10.1. Definition List

Part	SolidWorks file which is a 3D drawing of a part of the robot.
Drawing	2D drawing of a Part which contains dimensions
Assembly	3D SolidWorks file which contains multiple Parts which are tied together with ‘mates’
Render	JPEG image file of a Part or Assembly in high resolution.

10.2. System overview

The render below contains all the parts and sub-assemblies that are included in our design. The 42” Display is used for interaction with people and as a way to bring information about the garbage

problem and the robot. The speakers in the front are used to let Willy speak and can be used for the sound when displaying a video.

[image1] | *media/image1.jpg*

The bins on each side of Willy will be used to encourage people to throw away their garbage. Willy will ask for trash.

[image2] | *media/image2.jpg*

The Kinect is used to look for people (skeletons) so that Willy can interact with them. The LiDAR which is placed on the front of the robot is used for indoor navigation and might be used for outdoor obstacle detection.

Behind the LiDAR there is a LEAP-motion controller which can be used to control the display with gestures. The bumpers in the front and in the back of the robot is filled with ultrasonic sensors. Six in the front, six in the back and four downwards to look for kerbstones and potholes. Inside the robot there are six 33Ah car batteries which supply the necessary power. In the rear and top of Willy there are fans to cool the whole robot.

[image3] | *media/image3.jpg*

10.3. Part & assembly overview

10.3.1. Part 1 front plane

The front plane is one of three pieces which are made out of glass-fibre in combination with a polyester resin. A mould is made out of Styrofoam plates. The front plane contains the tv screen, place for a Kinect sensor, a LEAP motion controller, a SICK TIM551 LiDAR and two speakers with grills. The frame goes through the two holes at the bottom of the front plane.

[frontplane] | *media/frontplane.jpg*

This part is made together with the midplane and backplane using the same mould. The front plane is fixed by the front of the frame.

10.3.2. Part 2 midplane

The mid plane is a frame on which the front and back plane are fixed. There are possibilities to fix the tv screen as well. At the bottom there are notches for the swivel wheels on both sides.

[midplane] | *media/midplane.jpg*

The midplane is made together with the front and backplane with the same mould.

10.3.3. Part 3 backplane

The backplane contains slots where the fans can be placed. The design of the backplane improves the flow of air from the bottom to the top of the robot where the air is blown out.

[backplane] | *media/backplane.jpg*

The slanted part is used as a hood so there is still a possibility to work on the electronics. The backplane is made together with the front plane and midplane with the same mould.

10.3.4. Part 4 bin

The bin is a part which is placed on both sides of the robot. It will be used to encourage people to throw away their garbage. There is a special bin opening part which is fitted on top of this bin.

[bin] | *media/bin.jpg*

10.3.5. Part 5 bin opening

The bin opening part is made to have access at the front of the robot.

[bin opening] | *media/bin_opening.jpg*

10.3.6. Part 6 speaker grill

The speaker grill is a part that protects the speaker from most of the larger items like rocks, cans and bottles. Besides that it protects the vulnerable speaker material from peoples' fingers.

[fangrid] | *media/fangrid.jpg*

10.3.7. Assembly 1 computer housing

This is a example of how the housing of the computer could be made. The brackets fit between the frame and the holes provide the computer with a cooler airflow.

[computerhousing] | *media/computerhousing.jpg*

10.3.8. Assembly 2 fan

These fans are standard 140mm case fans which suck up air at the bottom rear of the robot while at the top they blow the hotter air back out.

[fan] | *media/fan.jpg*

10.3.9. Assembly 3 frame

The frame is a part that provides the whole robot with a certain firmness. Parts like the batteries and the outer housing are fixed on the frame. The frame is in that case fixed on top of the wheelchair. The frame is made out of 25x25mm square steel tubes with a thickness of 1.5mm. The tubes are welded in place.

[frame] | *media/frame.jpg*

10.3.10. Assembly 4 front bumper

The front bumper contains six ultrasonic sensors for a wide view. It also has two sensors aimed to the ground. With these sensors things like a sidewalk or potholes can be detected.

[frontbumper] | *media/frontbumper.jpg*

10.3.11. Assembly 5 rear bumper

The rear bumper also has six sensors placed outwards and two aimed at the ground. With the front and rear bumper combined there is an almost full 360 degrees view around the robot.

[rearbumper] | *media/rearbumper.jpg*

10.3.12. Assembly 6 wheelchair

The wheelchair assembly contains the motors and wheels combined with a frame that holds these parts fixed together. This is the base of Willy at the current state.

[wheelchair] | *media/wheelchair.jpg*

10.4. Design choices

The first choice we had to make with the design is which design we wanted to use. We made a design ourselves and there was a design available. The design that was available was not yet usable on the current frame. The design we made was usable in the current frame.

Together with the productowner we concluded that the available design should be the one. The reasons for that are that there was a team behind that design that had researched that idea and that it is a quite simple design to build.

10.4.1. Parts design choices

The bin had a problem that the opening was too small for bottles and cans. Because of that we made it a bit bulkier than the original design. The bin opening part is adjusted accordingly.

10.4.2. Assembly design choices

The front bumper and rear bumper were an aluminium plate in which the ultrasonic sensors were mounted. Because of the design this had to be changed. We made it in the shape of the plating itself. In this way there is only a little gap between the plating and the bumper. This also ment that it needed a sixth ultrasonic sensor because of the widened front part.

Besides the 360 degrees ultrasonic view it needed sensors which had to be aimed at the ground. This helps with finding curbs and unevennesses in the road. Both the front and rear bumper are adjusted to fit two sensors downwards.

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

11. Realisation

11.1. Steel work

11.1.1. Frame

[st01] | *media/st01.jpg*

11.1.2. Mounting

[st02] | *media/st02.jpg*

11.1.3. Bumper preparation

[bmp01] | *media/bmp01.jpg*

11.2. Mold

[maldesign] | *media/maldesign.jpg*

11.2.1. Wooden frame

[wd01] | *media/wd01.jpg*

[wd02] | *media/wd02.jpg*

11.2.2. Styrofoam board

[sch01] | *media/sch01.jpg*

[sch02] | *media/sch02.jpg*

[sch03] | *media/sch03.jpg*

[sch04] | *media/sch04.jpg*

11.3. Polyester work

We used the workshop of Koers Polyester Techniek in Nieuwleusen. http://koerspolyestertechniek.nl/

[koers logo] | *media/koers-logo.jpg*

The mold is covered with glass fiber and polyester resin. Starting with one base layer with a 225 gr/m2 and following with four layers of 600 gr/m2.

11.3.1. Preparation

[po01] | *media/po01.jpg*

[po02] | *media/po02.jpg*

11.3.2. Polyestering

[po03] | *media/po03.jpg*

[po04] | *media/po04.jpg*

[po05] | *media/po05.jpg*

11.3.3. Sanding/Sawing/Filling

[po06] | *media/po06.jpg*

[po07] | *media/po07.jpg*

11.4. Cooling

Cooling is done by 9 140mm fans in sets of 3 across the robot, at every arrow a set can be found. We have not yet found a sponsor for fans after many tries.

[cooling1] | *media/cooling1.JPG*

[cooling2] | *media/cooling2.JPG*

[cooling3] | *media/cooling3.JPG*

- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs

- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

12. Hardware

12.1. Batteries

Research Batteries

Welcome

Project Willy

- Willy

- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

13. Peripherals

13.1. Screen

Research Screen

13.2. Alternative interaction

Research Alternative interaction

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

14. Sensors

14.1. Leap

Research Leap

14.2. Lidar, Kinect & Ultrasonic

Research Indoor navigation

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide

- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

15. Social interaction

Research Social interaction

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status

- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

16. Software

16.1. Backup

Research Back-up

16.2. Ubuntu

Research Current status

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware

- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

17. Web-Interface

Research Current status

Welcome

Project Willy

- Willy
- Publicity

- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background

- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

18. ROS Introduction

18.1. An introduction

As a requirement from the product owner, **ROS** is used as framework on Willy. ROS, the Robot Operating System, is a flexible software **framework** for use in robots. It consists of a collection of libraries, tools and conventions that provide basic infrastructure to communicate between different parts of the robot.

In the case of Willy, ROS is especially handy because Willy is made with a modular design. All modules can be removed without disrupting the other functionalities of Willy. For example, when the web interface is removed, Willy is still able to drive, but with another module as for example

the keyboard controller. Or the removal of the motor driver makes Willy still able to interact with public.

18.2. Nodes

A **node** in ROS can be seen as a module. It is an executable that communicates through ROS to other nodes to send and receive data. A node can be for example a c++ application, or a piece of Python code, or even an Arduino connected with USB running code. A piece of information a node receives or sends is called a **message**.

More information can be found at http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

18.3. Topics

A topic is a bus over which nodes can exchange data messages. A topic always has a name, so all topics can be identified.

[ros]

More information can be found at http://wiki.ros.org/Topics

To interact with a topic, two methods are used, subscribing and publishing.

18.3.1. Subscribing

Subscribing is getting data from a topic. Everytime the data in a topic is updated, a message will be passed to all subscribing nodes. This way a node can use this information.

18.3.2. Publishing

Publishing is sending data to a topic. When a node has new information, a message will automatically be sent to the linked topic, so this data is updated.

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status

- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

19. ROS navigation

19.1. Navigation stack

Willy uses the ROS navigation stack to navigate indoor. The ROS navigation stack is a complex collection of multiple ROS packages. These packages read the sensor data we provided from our sensors to create a map of the environment. When the user provides a goal from rviz, the global planner will provide a path to that goal and calculate the costs in that map. costs can be compared with obstacles. The bigger the obstacle, the higher the cost.

[navigationStack]

19.2. Path calculation

In the image above you can see the overview of the ROS navigation stack in our project. We use the move_base from ROS. We send goals from rviz or C++ to the move_base. The move_base converts the goal into a path and calculates a route. This proces is shown below.

[rviz planner]

19.3. Autonomous driving

When the path and costs are calculated, the move_base will send geometry::Twist messages to the mobile base. In our case this is the motorcontroller with the cmd_vel topic. When the motorcontroller receives a geometry::Twist message, the motor will go driving on the given X, Y and Z velocity from the Twist message.

The ROS navigation stack has a lot of parameters. These parameters change the way the path is calculated and the maximum speed etc. For more information about these parameters see:

https://artofrobotics.github.io/WillyWiki/Willy/Parameters.html

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide

- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

20. Driving of willy

20.1. Steps for autonomous driving

These steps will start the autonomous driving of willy. Do you want to drive manually by using the keyboard? Then take a look at the steps for manual driving underneath.

1. **Deploy the brakes**
   To start Willy, the brakes need to be deployed. Otherwise, Willy will not be able to drive. To deploy the brakes you have to use the levers on the front of the motors. You can check if the brakes are deployed by trying to push Willy. The brakes are applied when the wheels can't rotate anymore.

2. **Turn the power on**
   Willy needs power to be able to operate. Make sure the power levers on the side of Willy are both turned on (under the voltage display).
3. **Turn the motor controller off**
   Willy cannot start when the motors are turned on. To turn the motor controller off, use the switch on the bottom of the pc case. (or the emergency button for convenience)
4. **Start Willy by running the startup command**
   Willy is now able to start. You can easily start willy by running our startup package. To run this package, navigate to the following directory:
   `/home/willy/Documents/WTGD/willy`
   Run the following command to build the package:
   `catkin_make`
   Then make the terminal point to the source code by running the following command:
   `source devel/setup.bash`
   Finally run the package using the following command:
   `roslaunch willy_navigation willy_navigation.launch`
   Willy will now start the software and hardware to start the autonomous driving project.

Do not forget to release the emergency stop when the steps above are finished. Otherwise willy will not drive.

20.2. Steps for manual driving

These steps are optional. Only execute the following steps if you want to drive Willy manually.

1. **Deploy the brakes**
   To start Willy, the brakes need to be deployed. Otherwise, Willy will not be able to drive. To deploy the brakes you have to use the levers on the front of the motors. You can check if the brakes are deployed by trying to push Willy. The brakes are appled when the wheels can't rotate anymore.
2. **Turn the power on**
   Willy needs power to be able to operate. Make sure the power levers on the side of Willy are both turned on (under the voltage display).
3. **Turn the motor controller off**
   Willy cannot start when the motors are turned on. To turn the motor controller off, use the switch on the bottom of the pc case. (or the emergency button for convenience)
4. **Start Willy by running the startup command**
   Willy is now able to start. You can easily start willy by running our startup package. To run this package, navigate to the following directory:
   `/home/willy/Documents/WTGD/willy`
   Run the following command to build the package:
   `catkin_make`
   Then make the terminal point to the source code by running the following command:
   `source devel/setup.bash`
   Finally run the package using the following command:

`rosrun driving_willy main`
Willy will now start the software and hardware to start the manual driving project.

Do not forget to release the emergency stop when the steps above are finished. Otherwise willy will not drive.

Keys for the control of Willy

```
Q -> Stop moving
```

```
W -> Move forward
```

```
A -> Move left
```

```
S -> Move backward
```

```
D -> Move right
```

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

21. Remote

21.1. Setup Wifi connection

Before any form of connection can be realized, a wireless connection is required. When the power switch is turned on, the router will setup a wifi access point. Connection with the access point can be set with the following data.

SSID: Willy WPA2-password: See password file.

21.2. Remote Desktop connection

If a wifi connection is established. A RDP connection can be realized. The following working method applies to a Windows environment.

Start → Run → MSTSC → 192.168.0.100 0 → Connect

[mstsc] | *media/mstsc.png*

Local login credentials can be used when logging in.

21.3. SSH

If a wifi connection is established. An SSH connection can be realized. SSH can be setup through a SSH client such as 'Putty'. The default port 22 is required with the IP of '192.168.0.100'. See the screenshot below for details.

Local login credentials can be used when logging in.

[putty] | *media/putty.png*

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

22. Willy-Web

22.1. Setup Wifi connection

Before any form of connection can be realized, a wireless connection is required. When the power switch is turned on, the router will setup a wifi access point. Connection with the access point can be set with the following data.

SSID: Willy WPA2-password: See password file.

22.2. Web platform

Visit http://192.168.0.100 in your browser. Login using the password found in the password file to authenticate. After this authentication you can enter your name and start using the platform.

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters

- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status

- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

23. Progress and quality

Milestone	Researched	Realized	Tested	Category
Assessment current status	Done	Done	Done	Documentation
Willy Improvisation	Done	Done	Done	Hardware Software Documentation
Design	Done	Done	Done	Software
WillyWiki	Done	Done	Done	Software
Willy web	Done	Done	Done	Software
Indoor navigation	Done	Done	In progress (fine tuning)	Software Hardware
Plating	Done	In progress		Hardware
Social interaction	Done	In progress		Software

23.1. Progress report

Assigment current status

Requirement	F/Q/P	Test case	Result by ..
Documentation corresponds to the current status of Willy	functional	compare documentation with the current status of Willy	Positive
Documentation meets the DOD	Principle	Documentation is reviewed conform the DOD	
Components of Willy are explained	functional	Hardware, Software and Documentation is understandable and traceable	Positive
Configuration can be traced through Documentation	Quality manageability	Willy can be installed from scratch without back-up	Positive

F/Q/P = functional / Quality / Principle

Willy Improvisation

Requirement	F/Q/P	Test case	Result
Current status of Willy correspondents to the current documentation and expectations of the client	Functional	Documentation and expectations of the client are compared with the current status of Willy	Positive
Hardware, Software and Documentation satisfies the DOD	Principle	The Current Documentation had been tested on the DOD	
Current functionality is reliable and stable	Quality (availability)	Current Software, Code and hardware is tested and is operational while tested multiple times	Positive
Design decisions are substantiated	Principle	Design decisions are underpinned by research or customer preference	Positive

F/Q/P = functional / Quality / Principle

Design

Requirement	F/Q/P	Test case	Result by ..
The design meets the expectations of the client	Principle	The design has been approved by the client	Positive
Dimensions are traceable from the design	Principle	Dimensions can be digitally exported from the design	Positive
The design is flexible	Quality scalability	New adjustments can be made by future groups by editing a flexible file format (sldprt - Solidworks)	Positive

F/Q/P = functional / Quality / Principle

WillyWiki

Requirement	F/Q/P	Test case	Result by ..
The WillyWikki is simple to understand	Quality manageability	Design and user documentation is available.	Positive

Requirement	F/Q/P	Test case	Result by ..
Documentation is up-to-date	Principle	All documentation is trasfered into .adoc format and added to GIT	Positive
The WillyWiki is modular	Principle	Information on the WillyWiki is exportable to multiple file formats.	Positive
New information can be added	Quality Managebility	The Willywiki automatic updates based on a git repository	Positive

F/Q/P = functional / Quality / Principle

WillyWeb

Requirement	F/Q/P	Test case	Result by ..
The WillyWeb must be dynamic and scalable	Quality scalability	The WillyWeb must be wide compatible with other software components of Willy	Positive
The WillyWeb must able to display real time information about Willy	Quality availability	The WillyWeb is compatible with ROS (previous requirement) and have a delay less that a second	Positive
The WillyWeb can Read and Read data to ROS	Principle	The WillyWeb can subscribe to ROS topics	Positive

F/Q/P = functional / Quality / Principle

- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design

- (2017/1) Base & Functionalities
- (2017/2) Research

24. ToDo and Advice

24.1. ToDo

24.1.1. Indoor navigation

Indoor navigation was realized with the use of a lidar and ultrasonic sensors. Willy can map a space and plan routes within it. The following features and improvements are recommended:

- Fine-tune indoor navigation
- Indoor navigation localization registration; Remember where Willy was in a certain room so he explores new places first.
- The storage and loading of previously mapped spaces and recognizing Willy's current location for further navigation.
- Navigating in spaces with a large audience. Recognizing moving objects and making decisions based on this.

24.1.2. Outdoor navigation

The only thing that has been realized so far with regard to outdoor navigation is GPS and the compass. GPS coordinates can be received inside ROS when subscribed on the /GPS topic.

- Location based on GPS
- Visualize GPS location.
- Accessibility determination based on location
- Route determination based on GPS and location determination
- Indoor navigation localization registration to keep Willy within a certain area.

24.1.3. Social interaction

Social interaction has been investigated. As a group we have focused on offline functionality at the request of the client. Offline functionality has been investigated; Research Social Interaction

Offline wakeword recognition has been realized through the use of 'Snowboy'. Documentation about Snowboy can be found on: Snowboy Documentation.

Implementation of Snowboy on Willy is described here: https://artofrobotics.github.io/WillyWiki/Config/index.html

24.2. Advice

First, it's important to try to understand Willy and get trusted with the robot. Reading of all the documentation and trying to understand it is a must. Use the documentation to drive Willy. Another essential part of understanding Willy is the understanding of the underlying framework. Understanding how ROS works makes understanding Willy much easier.

Willy is a large and complicated project. There are a lot of things that can go wrong. At the start of the project this can give a lot of frustation. When this happens it is essential to know Willy. Dealing with such situations can help understanding Willy even more.

Another advice is that it is essential to keep updating this wiki. That way contunation of the project keeps guaranteed. :numbered: :toc: :toclevels: 5 :icons: font

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status

- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Group of 2018 S1

1. Specialities

todo

2. Members

Jesse Bouwman BIM Vincent van Dijk IDS Jonathan ten Hove SE Martijn van Olst ESA Gerard Zeeman ESA

3. Main work

todo

4. Archive

Unresolved directive in Archive/2018S1.adoc - include::2018S1/index.adoc[]

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote

- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities

- (2017/2) Research

Group of 2017 S2

1. Specialities

2. Members

3. Main work

4. Archive

Google Drive 2017S2

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware

- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Group of 2017 S1

1. Specialities

2. Members

3. Main work

4. Archive

Google Drive 2017S1

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD

- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

Group of 2016 S2

1. Specialities

2. Members

3. Main work

4. Archive

Google Drive 2016S2

- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

5. Development Guide

5.1. Prerequisite

- Basic knowlegde of NodeJs, Git and ROS
- Install Git & Editor

https://code.visualstudio.com/

- Install NodeJS
  https://nodejs.org/en/download/
- Install Sails

```
npm install sails -g
```

- If on Windows:
  1. Install Ubuntu
     - Got to Microsoft store
     - Search for 'Ubuntu'
     - Click get/install
  2. Install ROS
     - Follow ROS Kinetic installation + (this might take some time) http://wiki.ros.org/kinetic/Installation/Ubuntu

```
sudo apt-get install ros-kinetic-desktop-full
```

     - And enviroment setup

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

     - Test the installation running ROS

```
roscore
```

     - Done for now

```
(ctrl+c)
```

5.2. Github

- Invite your personal Github account to acces ArtOfRobotics repos https://github.com/orgs/ArtOfRobotics/people
- Clone Git Repo

```
git clone https://github.com/ArtOfRobotics/WWEB
```

- Switch to test branch

```
git checkout -b origin/test
```

5.3. Compilation

Compilation is done via catkin, this is done to create a rospackage so that nodejs can run in a ROS enviroment.

```
cd WWEB/src
npm install
cd ..
source /opt/ros/kinetic/setup.bash
catkin_make
```

5.4. Testing/Debugging

Run without ros:

```
cd WWEB/src
sails lift (or node app.js)
```

Run with Ros:

1. Start Roscore
    - Open a terminal (Ubuntu app on windows) -

    ```
    cd WWEB
    source devel/setup.bash
    roscore
    ```

2. Run webplatform
    - Open a terminal (Ubuntu app on windows) -

    ```
    cd WWEB
    source devel/setup.bash
    rosrun willyweb start.sh
    ```

The rosrun command might not have acces to port 80 for this to work use sudo -s

```
sudo -s
rosrun willyweb start.sh
```

5.5. Running Scripts

In the same manner as you would do Testing/Debugging you can also run scripts. Scripts are located in the folder 'WWEB/src/scripts'.

1. Start Roscore
   - Open a terminal (Ubuntu app on windows) -

```
cd WWEB
source devel/setup.bash
roscore
```

2. Run sending script
   - Open a terminal (Ubuntu app on windows) -

```
cd WWEB
source devel/setup.bash
rosrun willyweb scripts/send.js
```

3. Run receive script
   - Open a terminal (Ubuntu app on windows) -

```
cd WWEB
source devel/setup.bash
rosrun willyweb scripts/receive.js
```

Rosrun makes it possible to communicate with ROS because it is now run as a ROS package

The 'start.sh' script consist out of a simple run script which launches the webplatform

```
#!/usr/bin/env bash
node src/app.js
```

- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

6. Software architecture document

The Software Architecture Document contains all the functional requirements, use cases and wireframes. It also includes a technical overview and explaination about the written code and how RosNodeJS is implemented. The wiki contains some more info about RosNodeJs because it lacks a proper documentation.

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

7. RosNodeJs

7.1. Introduction

RosNodeJs is a framework in which you can create real ROS Nodes. It is good to have basic knowledge about ROS before reading any further. This can be read on the Wiki and at the official website http://www.ros.org/.

RosNodeJs lacks in a proper documentation but the official wiki is available on ROS's own website, see http://wiki.ros.org/rosnodejs/overview.

RosNodeJs functions as a ROSPackage, the instance of WillyWeb is run as a RosPackage named 'willyweb'. However you are able to run multiple instances of the Rospackage 'willyweb' for example if you run multiple scripts.

7.2. Advantages/disadvantages

7.3. How to use

7.3.1. Declaration

Ros is used as a NPM package and is loaded into a controller using the following code: image::media/sad/image18.png[] Here in the first line of code RosNodeJs is loaded into the constant variable 'rosnodejs'. In the second line the type of message is defined, in this case 'sensor_msgs'.

7.3.2. Node Handle

In the following section of code these variables are used as following: image::media/sad/image19.png[] In the third line of code you can see an if statement that checks if a so called Node Handle already exist. A Node Handle is the Node on which the platform operates, you could also see this as the Rospackage 'willyweb' but then in a active node. You would expect that this is not needed because you require the same RosNodeJs module across the application but research found out this does not work as of the moment of writing.

If a Node Handle already exits the node can be accessed using 'rosnodejs.nh'. If this Handle does not yet exist it awaits the creation of it using 'rosnodejs.initnode('/willyweb')'.

The main reason this is done each time RosNodeJs is used in the code is because the same instance must be used over the whole application. Multiple Node Handles cannot coexist. The official examples given are also written inside one function, this is not practical for a well designed application. Another solution was therefore to create a RosController in which all communication with Ros would be done. The main disadvantage of this is that you'll have to create a function for each and every message type used throughout the application which violates the Single responsibility principle in the SOLID principles.

7.3.3. Subscribing

If the Node Handle is initialised we can use the various functions ROS uses, for example subscribe: image::media/sad/image20.png[] In the first line the subscribe method is called with the first parameter being the topic on which it must subscribe '/sonar' and next up the message type 'msg.LaserEcho' after that the function is declared on what to do when data is published on the topic. This data is stored in the local variable 'data', in the current situation it would blast an update over the socket that new data is available. This update also contains the new data received form the topic.

7.3.4. Advertise

Before being able to publish onto a ROS topic we first need to advertise that we are going to publish data. This is done using the advertise function on the Node Handle: image::media/sad/image21.png[] First a variable is made so that the advertise topic can be reused inside the controller. image::media/sad/image22.png[] After that the variable is filled with a advertise topic, the function advertise consist out of 2 parameters in this case the topic to advertise on '/led' and a message type 'msgs.ColorRGBA'.

7.3.5. Publishing

After the initialisation of the advertise topic the following function is used to publish data onto this ROS topic: image::media/sad/image23.png[] In this example the earlier variable 'pub' is reused to call the function publish on the advertise topic. This function contains the data in JSON format that needs to be published onto the topic. Make sure this data is coherent to the official format used by the specific message type which can be found at ROS official wiki: http://wiki.ros.org/common_msgs?distro=kinetic

7.4. Implementation

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu
- Remote

- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities

- (2017/2) Research

8. Interaction

8.1. Flow Chart

[image]

8.2. Screen Designs

8.2.1. English

1. This is the screen shown when Willy is driving around, and there is not yet a person to interact with.

   [image]

2. This screen is shown when a person is starting to interact with Willy. At the bottom there are two arroys pointing to the garbage bins.

   [image]

3. After that Willy asks permission to ask a question to the person in front of Willy.

   [image]

4. When the person says 'no', then this screen is shown, after wich the robot continues driving.

   [image]

5. When the person says 'yes', a random question is shown with a number of multiple choise answers.

   [image]

6. When the given answer is wrong, this screen is shown.

   [image]

7. When the given answer is right, this screen is shown.

   [image]

8. After the question, Willy starts to play an informative video about garbage.

   [image]

8.2.2. Dutch

For comments at each image, see the English version above.

1. [image]
2. [image]
3. [image]
4. [image]
5. [image]
6. [image]
7. [image]
8. [image]

8.2.3. Speech configuration

```
curl -sL https://deb.nodesource.com/setup_9.x | sudo -E bash -

sudo apt-get install -y nodejs

sudo apt-get install -y build-essential

sudo apt-get install python-pyaudio python3-pyaudio sox

pip install pyaudio

sudo apt-get install libmagic-dev libatlas-base-dev

git clone https://github.com/ArtOfRobotics/Speechtest.git

cd Speechtest

node server.js
```

Welcome

Project Willy

- Willy
- Publicity
- Sponsors

Startup Willy

- Driving Willy
- Remote
- Willy Web

Configuration

- GIT Setup
- Ubuntu

- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design

9. Development Guide

9.1. Prerequisite

To be able to develop Willy there are a few Prerequisites:

- Integrated Development Environment
- Basic knowledge about C++ and C
- Basic knowledge about ROS
- Basic knowledge about GIT

Integrated Development Environment
Visual Studio Code is a Integrated Development Environment. This program is used to write code and use git to send the documents to Willy. Visual studio can be extended with a lot of extensions. These extensions make it possible to align the code and see if the syntax of your code is correct or incorrect.

Basic knowledge about GIT
GIT is used as file transfer and version manager of the current Willy project. Everything starts with setting up the GIT repository and be able to push and pull parts of code. That makes GIT essential in the Willy development.
A tutorial about GIT and the setup of GIT can be found here:

https://www.atlassian.com/git/tutorials/setting-up-a-repository

Basic knowledge about C++ and C
The Willy code is written in C++. To add features and understand the code, some basic knowledge of C++ is required. If you want to learn the basics of C++? Follow a semester programming or see the following tutorial

https://www.studytonight.com/cpp/introduction-to-cpp.php

Basic knowledge about ROS
Because Willy is based on the Robot Operating System (ROS), a basic knowledge of ROS is required to communicate with the different modulair parts of Willy. ROS learing may be difficult, but you can follow the following tutorial to learn the basics:

http://wiki.ros.org/ROS/Tutorials

9.2. Development

To start development, make sure you complies to the requirements above.

1. Make a folder on your local computer where you want the Willy project to take place.
2. Install git.

Git can be downloaded from the following website: https://git-scm.com/downloads

1. Clone the WTGD git repository.
   This can be done with the following command:
   `git clone https://github.com/ArtOfRobotics/WTGD`
2. Checkout at the `test` branch.
   This can be done with the following command:
   `git checkout test`
3. Open Visual Studio Code and open the cloned WTGD folder.
   This can be done by following this steps:
   File → Open Folder → Navigate to WTGD folder → Select Folder
4. In the navigation window on the left, navigate to:
   willy → src

In this folder all the Willy ROS packages are listed.

9.3. Compilation

To compile your changes you have made to the Willy project, a few steps are required.

1. Commit and push your changes to the GIT WTGD repository. This can be done using git bash or Visual Studio Code integrated source control.

2. Startup the computer of Willy, login using the default login credenticals and open a terminal window.
3. Enter the following command: `cd /home/willy/Documents/WTGD/willy`
4. Pull the git changes (There may be no changes because of the automatic git sync) `sudo git pull`
5. Build the package using catkin_make `catkin_make`
6. Make the terminal point to the source code of driving_willy `source devel/setup.bash`
7. Now you can launch the project as described in startup wiki page.

For more information about starting up willy? See our startup guide: https://artofrobotics.github.io/WillyWiki/Startup/Driving-Willy.html

9.4. Testing/Debugging

To test the new builded code, you need to launch the project you builded. Make sure you followed the steps above and leave the terminal open.

Make sure to press the emergency button or the switch on the motor controller that makes the LED turn off. Otherwise the motors cannot start.

To launch the willy_navigation package you can use the roslaunch command. `roslaunch willy_navigation willy_navigation.launch`

Do not forget to depress the emergency button or switch the lever on the motor controller after starting the project. Otherwise Willy will not drive.

Step if Willy will not drive

Willy can sometimes be a little unreliable and stops driving. This can have multiple reasons. Here are some of them and the way to fix the problems:

1. Forgot to press the emergency button before starting up the project.
   → Use the command `pkill ros` in a new terminal to stop ROS.
   → Follow the steps above or the startup guide to restart Willy.
2. Forgot to depress the emergency button after starting the project.
   → Depress the emergency button and willy will start driving
3. Forgot to set a goal to navigate to.
   → Use your mouse to send a new goal in Rviz. This button can be found in the top navigation bar in rviz.
4. Rviz hangs on startup
   → Use the command `pkill ros` in a new terminal to stop ROS.
   → Use the command `pkill rviz` in a new terminal to stop Rviz.
   → Reboot the project as described in the startup guide.
5. Mini pc will not power on.
   → Use the switch on the top of willy to power on Willy.

10. Findings

This file contains all the things we have found while prototyping and may be usefull to other project groups

10.1. Software

- The motor controller is a device that is used to control the cmd_vel topic created by the ros navigation stack. The ros navigation stack published geometry::Twist messages
- ROS is complicated and difficult to learn. But if everything is setup just right, ROS makes it really easy to communicate between your hardware.
- If you want to create files to start multiple projects? Do not use bash files. ROS can't handle bash files properly. use ros launch files instead.
- The making of the DrivingController was a complex situation. That is because of the fact every ROS node has only one ROS nodehandler. To make it possible to subscribe and advertise everywhere in the DrivingWilly code, we must send the nodehandler to every subcontroller by using pointers. This request a detailed knowledge of C++ and ROS.
- The setup of the ROS navigation stack is difficult because of the fact that every robot is different. A lot of components needs to be setup on your own. As example the move_base, the transformations and rotations and the cmd_vel topic.

10.2. Vision

- When the kinect is used together with the LIDAR for indoor navigation, the kinect can not

handle the fast scan frequency of the LIDAR which will cause the transform of the robot to become unknown. Then the robot will lost its orientation.

- The kinect cannot look further than 10 meters away. Thats makes the kinect not usefull for indoor navigation.

10.3. Hardware

- The current batteries should be powerfull enough to power the current Willy.
- The 230 volt touchscreen in combination with the power converter is replaced by a 19v display.
- The brakes are not easy to deploy. Thats because the levers on the side of willy are to loose to deploy the brake. We can't tight them because the screw is malformed.
- The GPS sensor and compass only work outside and are controlled using the GPSController.
- The kinect cannot be used outside. The IR camera can't handle the bright sunlight.

10.4. Design

- The current battery shell will not fit because of the new motorcontroller. Therefore we created a set of new brackets to mount the 6 batteries onto the chassis.

10.5. Social interaction

- There are a lot of ways to communicate with Willy. The ones we're using are:
  → Leap
  → kinect
  → Voice recognition
  → TTS (Text to Speech)
  → Lights (Led and sirene)
  → Keyboard
  → Mouse
- Offline speech recognition is limited. There are a few solutions for hot-word detection like Snowboy.

Configuration

- GIT Setup
- Ubuntu
- Remote
- Wiki

ROS

- Introduction to ROS
- Navigation

Technical

- Development Guide
- Findings
- Hardware
- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status

- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

11. Hardware

11.1. Overview

A simple overview of all the hardware in Willy is made in the following scheme:

[image]

11.2. Components

This chapter contains all the components with explanation.

11.2.1. Mini pc

This is the dedicated PC for running the ROS Server. Al the calculational work will be done on this PC. Therefore, we chose for a PC with some calculation power as well as enough RAM.

Product	**Dell Optiplex 3020 Micro (3020-9806)**
CPU	Intel Core I5 4670T quad-core
Video Chip	Intel HD Graphics 4600
RAM	16 GB DDR3L Sodimm
Harddisk	Western Digital Black 500GB 2,5" HDD
Amount of USB Ports	• 2x USB3.0 • 4x USB2.0
Other Ports	• 1x VGA • 1x DisplayPort • 1x 1GB/s Network Port • 1x Microphone Jack • 1x Headphone Jack
Wireless Connection options	WLAN chip

11.2.2. Motors and controller

Willy contains 2 motors from a second-hand mobility scooter. A previous project group has chosen this option as documented in the ‘Ontwerp verslag’. (Ontwerpverslag, 2016)

These motors need 24 volts and use a max of 20 amps. One of the previous project groups created the motor controller to control the motors. This is documented in the ‘Systeem dossier’ from a previous project group. (Systeem Dossier, 2016)

[image]

This motor controller is made by using an Arduino Mega and a custom shield with additional custom hardware. The controller recieves data from ROS and sends it to an internal controller. This is part of the wheelchair and is named Penny and Giles Pilot Plus.

The motor controller is built as shown below:

[image]

As can be seen in the image above, there are still cables for the odometry sensors. These are however not used anymore. This has two reasons. The first one is the fact that since using the lidar, the positioning is so accurate, that the wheel encoders are not neccesary anymore. The second reason is that both the sensors don’t function anymore.

The 3 cables to the internal motor controller (built in in the wheelchair frame) are connected to the original cable with a simple circuit. The pinout of the connector attached to the cable is as follows:

[image]

Pin number	Description	Internal Cable Color
1	+ 24V	Red
2	- 0V	Black
3	Driving data	White
4	Unknown (not used)	Yellow
5	Enable	Blue
6	Actuator Data (not used)	Green

The drive data pin (3) is used to send driving commands to the internal controller. The yellow cable is not used and the function is unknown. The enable pin is connected to the emergency button and the switch on the motor controller. The motors only work when the voltage level on this cable is 5 volts. The actuator pin is not used in this project. In the original wheelchair this pin is used for the lighting etc.

The motor controller also contains a connection for charging the batteries. The connector at the side of the box is connected to the + and - of the cable to the internal motor controller.

Brakes

The two engines of willy contain both a brake. This brake is controlled by the engine itself. When the active signal to the motors is lost, the motor will automatically activate the brake. To start the driving_willy project, you NEED to deploy the brakes. Otherwise Willy will not drive!

[image6]

[image7]

Make sure to deploy the brakes. Otherwise Willy won't work!! There is only one way to detect if the brakes are deployed. It's by simply testing if the wheel can't turn (Or a very little bit) anymore.

The wheels should not be able to turn for Willy to function normally. The brakes will automatically be released by Willy when driving.

11.2.3. Batteries

Old situation

In the old situation Willy contained 2 12V batteries connected in serial, adding up to 24V. Both batteries are 52Ah.

[image]

[image]

New situation

In the new situation the 2 batteries are replaced with 6 new 12V batteries. All the batteries are 33Ah. They are connected placing 2 batteries in serial 3 times in parallel. This adds again up to 24V.

[image]

[image]

11.2.4. Power Supply

The 19 volts for the screen is provided by a step-down converter. The module is based on the Xl4015 buch DC to DC converter. The input voltage range is 8-36V and the adjustable output range is 1.25V to 32V. The converter is rated for a max current of 5 amps.

The mini pc is connected to a small 19.5V DC step-down converter. This is needed because the screen needs the full amperage of the other 19.5 step-down converter. The mini pc use 65 Watt and the step-down converter can provide a maximum of 75 watt.

The power for the step-down converters is provided by the batteries. Between the batteries and the step-down converter, there is a fuse placed. The fuse called T5L250V is rated for 250V with 5A. Then a switch is used to turn Willy on or off. To split the 24v of the batteries to all the step-down converters, there is a homemade pcb plate created. This board is placed at the left of the step-down converters.

11.2.5. Sonars

The sonars are currently not connected, because the frame bumpers are not yet made. These sensors have to be placed in this metal piece.

To prevent collisions, ultrasonic sensors are used. These sensors measure distance by using sound. This is made possible by sending out bursts of high frequency noise, and then waiting for a reflection of that sound using the HC-SR04 ultrasonic sensor.

[image]

By using this data Willy is be able to decide if he is able to drive any further in a certain direction. In the event of Willy being not able to drive any further, he will decide if there is a direction where he is able to drive further. This way Willy will be able to drive around autonomously without collisions. How Willy reacts to objects in his navigation is researched by a previous group. (Navigation design v0.1, 2017)

The datasheet [1] for the HC-SR04 is included in the sources at the bottom of this document.

The sensors all use 5V as can be seen in the schematic:

[image]

As shown in the schematic above, all the 16 sensors are connected to the Arduino.

The new sensors don’t have a 'sig' pin, but two seperate pins called 'trig' and 'echo'. However, these two pins are connected to each other, so both pins use the same pin on the Arduino. The code has been adapted to this new situation.

The sensors are connected in the following order:

Arduino Digital Pin Number	Sonar Sensor Number
3	Sensor 0
4	Sensor 1
5	Sensor 2
6	Sensor 3
7	Sensor 4
8	Sensor 5
9	Sensor 6
10	Sensor 7
11	Sensor 8
12	Sensor 9
13	Sensor 10
A0	Sensor 11
A1	Sensor 12

Arduino Digital Pin Number	Sonar Sensor Number
A2	Sensor 13
A3	Sensor 14
A4	Sensor 15

The sensors are placed as follows:

[image]

It should be noted that not all the sensor are read at the same time. All odd numbered sensors are read first, and after a small delay the even numbered sensors are read. This is because sensors placed next to each other can interfere the readings of each other.

11.2.6. GPS / Compass

The GPS and compass are currently implemented but not used in the code on the mini-pc.

To allow Willy to drive outdoor by using a precomputed route, a GPS sensor and a compass are required. The GPS sensor is linked to the GPSController using ROS. In the GPSController the raw data is processed to usable coordinates. These coordinates are only used to draw a line where Willy has driven in the code of the previous group. The Compass is used to determinate the direction of willy. This is needed for navigation when willy is going to drive precomputed routes.

The sensors are connected according to the following scheme:

[image]

As can be seen in the scheme, there are two Arduino's used to measure sensor data. Both Arduino's are connected with I2C (see pins A4 and A5). The second Arduino with the GPS sensor is powered with the 5V pin from the first Arduino. At the moment of writing this has not been fixed and reduced to one Arduino.

The GPS will not send data until a FIX has been established. When the reciever is indoors, this can take a long time. GPS works best when outside.

11.2.7. Screen

The screen used is a KDL-42W815B [2] from Sony. The screen is connected to the mini-pc with a VGA cable. The manual can be found at the bottom of this document.

The current screen is a replacement for the old screen. This screen was very heavy and energy consuming.

11.2.8. LIDAR

The previous group has also done research on a Lidar sensor. Unfortunately the previous documentation stated that it was not possible to link a Lidar to ROS. Also other methods where

somehow researched by a previous group but not in the form of a Lidar. (Research localization system v1.1, 2017)

A LIDAR sensor uses a laser to measure distance. With these measurements the sensor makes a map of all the objects in the environment.

[image]

The previous group has done some research concerning LIDAR and the link to ROS. But came to the conclusion that it is not possible to create a link between LIDAR and ROS. Therefore they decided to not implement the LIDAR sensor.

After doing some research we found that it is possible and supported to link ROS to a LIDAR sensor.

At this stage we use the LIDAR to navigate with Willy. The sensor is placed at the front of the robot.

The LIDAR is connected with an ethernet connection via a router to the pc.

11.2.9. Kinect

[image]

As a method to navigate inside, initially the Kinect was chosen. This is not true anymore. The Kinect is now used to detect if there is a person in front of Willy. A Kinect can create a framework of a person and see all movements of that person. By adding a Kinect to willy, willy will be able to recognize people and interact with them by using the speech recognition of the Kinect.

There are two versions of the Microsoft Kinect: Kinect 1 for Xbox 360 and Kinect 2 for Xbox one. See table 1 which list all the different features of both versions.

Feature	**Kinect 1**	**Kinect 2**
Color Camera	640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	320 x 240	512 x 424
Max Depth Distance	apr. 4.5 m	8 m
Min Depth Distance	40 cm in near mode	50 cm
Depth Horizontal Field of View	57 degrees	70 degrees
Depth Vertical Field of View	43 degrees	60 degrees
Tilt Motor	Yes	no
Skeleton Joints Defined	20 joints	25 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0

The main difference which is most important for us is the Field of View (FoV). The bigger the FoV, the more Willy can see in front of him. The Kinect 2 can also recognize more people and can see further away, which are both nice features to have when the social aspect of Willy will be implemented in the future. This makes the Kinect 2 more futureproof than its precursor. (Kinect 1 vs 2 specifications: , sd) (Kinect 1 specifications, sd) (Main factors/features of most industrial

computer vision hardware., sd)

The Kinect used in Willy is a version 1 Kinect, the old one.

References

- [[[1]]] HCSR04 Datasheet version 1. Retrieved from https://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf
- [[[2]]] Sony KDL42w815 Manual. Retrieved from https://www.sony.nl/electronics/support/res/manuals/4489/44895371M.pdf

- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

12. Common problems

This file contains a list of the most common problems and the problems we found during development.

12.1. Driving

- It is quite hard to deploy and undeploy the brakes.
- When the battery voltage drows below a certain level, Willy will stop driving at irregular intervals. This is probably an internal security system. The brakes are also deployed causing an abrupt stop.
- When driving manually, Willy uses characters pressed on the keyboard to move. When a lot of characters are pressed in a short time, Willy will try to process the complete list of characters, seeming to be uncontrollable.

- Willy has difficulty driving through doors when driving autonomously. This is caused by Willy being quite wide and limitations in the software for autonomous driving.
- The ultrasonic sensors have a lot of blind spots. Since the LIDAR can't scan the back of Willy, the ultrasonic sensors are the only sensors controlling the back of Willy. When Willy is driving backwards, this can cause problems.

12.2. Design

- Because of the width of Willy with garbage bins attached to the side, it is impossible to go through a door without detaching the bins, including the detachable buffers with ultrasonic sensors in it.

12.3. Web Interface

- There are no known bugs at this moment.

The Web Interface is still in development.

- Known Bugs
- Parameters
- Software

Web interface

- Development Guide
- SAD
- RosNodeJs
- Interaction

Research

- Hardware
- Peripherals
- Sensors
- Social interaction
- Software
- Web interface

Design

- Background
- Design Guide
- Technical
- Realisation

Status and Advice

- Status
- Todo & Advice

Archive

- (2016/2) Initial design
- (2017/1) Base & Functionalities
- (2017/2) Research

13. Launch file parameters

13.1. Overview

The autonomous driving of Willy is controlled by external ROS plugins. These plugins are:

Move_base → Used for sending Twist messages to Willy out of goals send to the ros navigation stack.

Hector_Mapping → Used for creating a map based on the LiDaR and transforms.
Sick_tim551 → Used for initilization of the LiDaR and creating the /scan topic for the map.
Transform → Used the send the transformation and rotations from devices on the robot to the rotation point of the robot.
Kinect → Used for the recognition of people and creating the /camera topic.
Rviz → Used for visualization of the map and sensors on the base frame.

All of these plugins have their own launch file. Every launch file has it's own parameters which can be tuned to finetune Willy.

13.2. parameters

Move base

Move_base_param.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/move_base_params.yaml`
It's possible that some parameters are not used yet. These can be added when needed.

- `base_global_planner` (string, default: "navfn/NavfnROS")
  The name of the plugin for the global planner to use with move_base.
- `base_local_planner` (string, default: "base_local_planner/TrajectoryPlannerROS")
  The name of the plugin for the local planner to use with move_base.
- `recovery_behaviors` (list, default: [{name: conservative_reset, type: clear_costmap_recovery/ClearCostmapRecovery}, {name: rotate_recovery, type: rotate_recovery/RotateRecovery}, {name: aggressive_reset, type: clear_costmap_recovery/ClearCostmapRecovery}])
  A list of recovery behavior plugins to use with move_base, see pluginlib documentation for more details on plugins. These behaviors will be run when move_base fails to find a valid plan in the order that they are specified. After each behavior completes, move_base will attempt to make a plan. If planning is successful, move_base will continue normal operation. Otherwise, the next recovery behavior in the list will be executed.
- `controller_frequency` (double, default: 20.0)
  The rate in Hz at which to run the control loop and send velocity commands to the base.
- `planner_patience` (double, default: 5.0)
  How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed.
- `controller_patience` (double, default: 15.0)
  How long the controller will wait in seconds without receiving a valid control before space-clearing operations are performed.
- `conservative_reset_dist` (double, default: 3.0)
  The distance away from the robot in meters at which obstacles will be cleared from the costmap when attempting to clear space in the map. Note, this parameter is only used when the default recovery behaviors are used for move_base.
- `recovery_behavior_enabled` (bool, default: true)

Whether or not to enable the move_base recovery behaviors to attempt to clear out space.

- `clearing_rotation_allowed` (bool, default: true)
  Determines whether or not the robot will attempt an in-place rotation when attempting to clear out space. Note: This parameter is only used when the default recovery behaviors are in use, meaning the user has not set the recovery_behaviors parameter to anything custom.
- `shutdown_costmaps` (bool, default: false)
  Determines whether or not to shutdown the costmaps of the node when move_base is in an inactive state.
- `oscillation_timeout` (double, default: 0.0)
  How long in seconds to allow for oscillation before executing recovery behaviors. A value of 0.0 corresponds to an infinite timeout.
- `oscillation_distance` (double, default: 0.5)
  How far in meters the robot must move to be considered not to be oscillating. Moving this far resets the timer counting up to the `oscillation_timeout`
- `planner_frequency` (double, default: 0.0)
  The rate in Hz at which to run the global planning loop. If the frequency is set to 0.0, the global planner will only run when a new goal is received or the local planner reports that its path is blocked.
- `max_planning_retries` (int32_t, default: -1)
  How many times to allow for planning retries before executing recovery behaviors. A value of -1.0 corresponds to an infinite retries.

13.3. Base local planner

base_local_planner_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/base_local_planner_params.yaml`
It's possible that some parameters are not used yet. These can be added when needed.

--Robot configuration parameters--
- `acc_lim_x` (double, default: 2.5)
The x acceleration limit of the robot in meters/sec^2
- `acc_lim_y` (double, default: 2.5)
The y acceleration limit of the robot in meters/sec^2
- `acc_lim_theta` (double, default: 3.2)
The rotational acceleration limit of the robot in radians/sec^2
- `max_vel_x` (double, default: 0.5)
The maximum forward velocity allowed for the base in meters/sec
- `min_vel_x` (double, default: 0.1)
The minimum forward velocity allowed for the base in meters/sec.
- `max_vel_theta` (double, default: 1.0)
The maximum rotational velocity allowed for the base in radians/sec
- `min_vel_theta` (double, default: -1.0)
The minimum rotational velocity allowed for the base in radians/sec
- `min_in_place_vel_theta` (double, default: 0.4)

The minimum rotational velocity allowed for the base while performing in-place rotations in radians/sec
- `escape_vel` (double, default: -0.1)
Speed used for driving during escapes in meters/sec.
- `holonomic_robot` (bool, default: true)
Determines whether velocity commands are generated for a holonomic or non-holonomic robot. For holonomic robots, strafing velocity commands may be issued to the base. For non-holonomic robots, no strafing velocity commands will be issued.

The following parameter is only used if holonomic_robot is set to true:
- `y_vels` (list, default: [-0.3, -0.1, 0.1, 0.3])
The strafing velocities that a holonomic robot will consider in meters/sec.

--Goal Tolerance Parameters--
- `yaw_goal_tolerance` (double, defaultv: 0.05)
The tolerance in radians for the controller in yaw/rotation when achieving its goal.
- `xy_goal_tolerance` (double, default: 0.10)
The tolerance in meters for the controller in the x & y distance when achieving a goal.
- `latch_xy_goal_tolerance` (bool, default: false)
If goal tolerance is latched, if the robot ever reaches the goal xy location it will simply rotate in place, even if it ends up outside the goal tolerance while it is doing so.

--Forward Simulation Parameters--
- `sim_time` (double, default: 1.0)
The amount of time to forward-simulate trajectories in seconds
- `sim_granularity` (double, default: 0.025)
The step size, in meters, to take between points on a given trajectory
- `angular_sim_granularity` (double, default: `sim_granularity`)
The step size, in radians, to take between angular samples on a given trajectory.
- `vx_samples` (integer, default: 3)
The number of samples to use when exploring the x velocity space
- `vtheta_samples` (integer, default: 20)
The number of samples to use when exploring the theta velocity space
- `controller_frequency` (double, default: 20.0)
The frequency at which this controller will be called in Hz.

--Trajectory Scoring Parameters--
The cost function used to score each trajectory is in the following form:
cost =
pdist_scale * (distance to path from the endpoint of the trajectory in map cells or meters depending on the meter_scoring parameter)
+ gdist_scale * (distance to local goal from the endpoint of the trajectory in map cells or meters depending on the meter_scoring parameter)
+ occdist_scale * (maximum obstacle cost along the trajectory in obstacle cost (0-254))

- `meter_scoring` (bool, default: false)
  Whether the gdist_scale and pdist_scale parameters should assume that goal_distance and path_distance are expressed in units of meters or cells. Cells are assumed by default.

- `pdist_scale` (double, default: 0.6)
  The weighting for how much the controller should stay close to the path it was given, maximal possible value is 5.0.
- `gdist_scale` (double, default: 0.8)
  The weighting for how much the controller should attempt to reach its local goal, also controls speed, maximal possible value is 5.0.
- `occdist_scale` (double, default: 0.01)
  The weighting for how much the controller should attempt to avoid obstacles.
- `heading_lookahead` (double, default: 0.325)
  How far to look ahead in meters when scoring different in-place-rotation trajectories.
- `heading_scoring` (bool, default: false)
  Whether to score based on the robot's heading to the path or its distance from the path.
- `heading_scoring_timestep` (double, default: 0.8)
  How far to look ahead in time in seconds along the simulated trajectory when using heading scoring.
- `dwa` (bool, default: true)
  Whether to use the Dynamic Window Approach (DWA)_ or whether to use Trajectory Rollout.
- `publish_cost_grid_pc` (bool, default: false)
  Whether or not to publish the cost grid that the planner will use when planning. When true, a sensor_msgs/PointCloud2 will be available on the `cost_cloud*` topic.
  Each point cloud represents the cost grid and has a field for each individual scoring function component as well as the overall cost for each cell, taking the scoring parameters into account.
- `global_frame_id` (string, default: odom)
  The frame to set for the cost_cloud. Should be set to the same frame as the local costmap's global frame.

--Oscillation Prevention Parameters--
- `oscillation_reset_dist` (double, default: 0.05)
How far the robot must travel in meters before oscillation flags are reset

--Global Plan Parameters--
- `prune_plan*` (bool, default: true)
Defines whether or not to eat up the plan as the robot moves along the path.

13.4. Common costmap

costmap_common_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/costmap_common_params.yaml`
It's possible that some parameters are not used yet. These can be added when needed.

- `max_obstacle_height` (double, default: 1.0)
  The max obstacle height the robot can handle in meters.
- `obstacle_range` (double, default: 2.5)

the maximum range sensor reading that will result in an obstacle being put into the costmap.

- `raytrace_range` (double, default: 3.0)
  determines the range to which we will raytrace freespace given a sensor reading.
- `robot_radius` (double, default: 0.8)
- `footprint` (double array, default: [[0.3, 0.3], [0.3, -0.3], [-1.0, -0.3], [-1.0, 0.3]])
  Here we set either the footprint of the robot or the radius of the robot if it is circular.
- `cost_scaling_factor` (double, default: 2.58)
  A scaling factor to apply to cost values during inflation.
- `inflation_radius` (double, default: 1.75)
  The radius in meters to which the map inflates obstacle cost values.
- `observation_sources` (string, default: scan kinect) The sensor sensors used for the navigation. Listed below.

scan: {data_type: LaserScan, topic: scan, marking: true, clearing: true, min_obstacle_height: -1, max_obstacle_height: 1.0}
kinect: {data_type: PointCloud2, topic: camera/depth_registered/cloud, marking: true, clearing: false, min_obstacle_height: -1, max_obstacle_height: 1}

13.5. Global costmap

global_costmap_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/global_costmap_params.yaml`
It's possible that some parameters are not used yet. These can be added when needed.

- `global_frame` (string, default: /map)
  The topic where the map is published.
- `robot_base_frame` (string, default: /base_link)
  The topic where the robot base frame is published and transformed.
- `update_frequency` (double, default: 1.0)
  Determines the frequency, in Hz, at which the costmap will run its update loop
- `static_map` (bool, default: false)
  Determines whether or not the costmap should initialize itself based on a map served by the map_server.

13.6. Local costmap

local_costmap_params.yaml

This param file can be found in the following directory: `/home/willy/WTGD/willy/src/willy_navigation/param/local_costmap_params.yaml`
It's possible that some parameters are not used yet. These can be added when needed.

- `global_frame` (string, default: /odom)
  The topic where the map is published.

- `robot_base_frame` (string, default: /base_link)
  The topic where the robot base frame is published and transformed.
- `update_frequency` (double, default: 5.0)
  Determines the frequency, in Hz, at which the costmap will run its update loop
- `publish_frequency` (double, default: 2.0)
  Determines the rate, in Hz, at which the costmap will publish visualization information.
- `static_map` (bool, default: false)
  Determines whether or not the costmap should initialize itself based on a map served by the map_server.
- `rolling_window` (bool, default: true)
  Setting the "rolling_window" parameter to true means that the costmap will remain centered around the robot as the robot moves through the world.
- `width` (double, default: 6.0)
  The width in meters of the costmap.
- `height` (double, default: 6.0)
  The height in meters of the costmap.
- `resolution` (double, default: 0.05)
  The resolution in meters / cell of the costmap.

14. Software

14.1. Arduino's

This chapter contains a overview of the code on the different Arduino's in Willy.

The following definition list is to clear up confusion about certain terms used in this chapter.

Term	*Definition*
Code conventions	These are rules that are used for programming and drawn up by the developers of Wake 'Em.
UpperCamelCase	This means that each word is capitalized. Example: SetAlarmOfAndroid.
LowerCamelCase	This means that each word is capitalized except for the first word. Example: setAlarmOfAndroid.
Odometry	The technique of measuring the amount of movement of the wheels with special sensors.
ROS	Robot Operating System, the framework Willy has been built on.

14.1.1. System Overview

The following schematic shows an overview of how the Arduino's communicate with ROS.

[image] ros_gps gps sonar odometry

It should be stated that the code for the odometry is currently removed. The reason for this can be found in the Technical Hardware document.

Code Overview

In this chapter the structure of the C code on the Arduino's can be found. These schemes are not class diagrams, since C is not object-oriented.

[image]

This is the code of the file in ros_gps. It is running on the Arduino connected to ros. The compass sensor is attached to this Arduino.

[image]

This code reads out the GPS sensor and sends it with a serial connection to the other Arduino (the one running ros_gps).

[image]

This is the code running the ultrasonic sensors. All the sensors have a digital pin.

[image]

This is the code of the motor controller. It is called 'Odometry', but this part is removed from the code. The code receives data from ROS and controls the motors of the wheelchair by sending serial data to the built in wheelchair controller.

14.1.2. Design Decisions

In the past the decision has been made to make the hardware modular. Unfortunately the reason behind this decision is not documented.

The motor controller (odometry) is made by the group of the second semester of 2016/17. The odometry code subscribes from the topic “/cmd_vel”. The code writes data it gets from ROS to the motors.

The sonar code was reading the 10 sonar sensors and publishes it to ROS on the topic “sonar”. The code has been written by the group of the second semester of 2016/17.

We updated the code so it uses 16 sensors, 6 for the front, 6 for the back, and 4 for facing down to the ground to detect if Willy is driving above a stair.

The GPS and compass code is written by the group of the first semester of 2017/18. The setup is made ambiguous. The compass has a Arduino and the GPS has a Arduino. The data from the GPS is sent to the Arduino with the compass. From that Arduino the data from the Compass and the data from the GPS is both being published to two separate ROS topics. The topic with compass data is “compass” and the topic with GPS data is “gps”. This setup has not been fixed yet.

14.2. Software

Here goes information about the code of the DrivingWilly ROS package

14.2.1. Overview

[image]

Our driving willy code consists of modulair components named controllers. These controllers controls the functionality of willy. Every piece of hardware has it’s own controller which controls the methods and stores the variables. A global class diagram can be found in the image above.

By using this way of coding, functionality of willy can be easily extended. Just create a new controller and add your methods to it. A detailed explaination of this can be found in the "Extending functionality" paragraph on this page below.

14.2.2. Controller setup

Every controller is created and declared in the main. The main creates the class object and returns a pointer to it.

[image]

This pointer is forwarded to the controller of willy named "DrivingController".

[image]

In this class the subcontrollers are declared. The current subcontrollers of willy are:

1. VisionController
2. GeneralController
3. VisionController

The pointers of the controllers are forwarded to the subcontrollers above in the "DrivingController"

[image]

Now every controller has acces to their subscontroller and the subcontroller has acces to the main where the other controllers are declared. By using this method we can always access all data and methods of the controllers and subcontrollers. To make that possible, we have to send class object pointers between functions. Thats done by using the "static_cast" in C++

[image]

Above is an example of the "static_cast" in the "MovementController"

The "static_cast" make it possible to convert a void pointer back into a class object pointer without declaring the class again.

[rsc]

14.2.3. ROS setup

To make the communication between nodes easy accessibly, we created some advertisers and subscribers in the ROS cloud.

[image] [image]

To see the basics of ROS and the general purpose, please visit our ROS generic wiki page on: //LINK TO ROS GENERAL page

These publishers and advertisers make it possible to push data generated by hardware to a rostopic, as example the sonar topic, and read that data all over the code of willy. Because you can echo the rostopic data anywhere and anytime.

Using ROS we're able to push keyboard characters on the 'keyboard' topic and subscribe on the keyboard topic. This means that when the code is running and you pressed a key, the function that you gave to the keyboard subscriber will be launched. In our code this means that the led lighting will turn red if you press the 'r' button on the keyboard. The flowchart is shown on this page below.

Always running (DrivingController) MovementController → GetKeyboardController() → ReadCharacter()

[image]

Running when char received subLed() → LedCallback() → Advertise ColorRGBA on 'led' topic()

[image]

The following paragraph is about the ros navigation stack. To learn the basics about ros_navigation, see our ros navigation wiki page.

14.2.4. autonomous driving

The autonomous driving of Willy is controlled by external ROS plugins. These plugins are:

Move_base → Used for sending Twist messages to Willy out of goals send to the ros navigation stack.
Hector_Mapping → Used for creating a map based on the LiDaR and transforms.
Sick_tim551 → Used for initilization of the LiDaR and creating the /scan topic for the map.
Transform → Used the send the transformation and rotations from devices on the robot to the rotation point of the robot.
Kinect → Used for the recognition of people and creating the /camera topic.
Rviz → Used for visualization of the map and sensors on the base frame.

All of these external plugin are started using the 'willy_navigation.launch'.

[image]

This file starts all of the external plugin launch files. All of these external launch files can be find in our git repository.

[image]

To learn more about the parameters in the launch files. See our 'parameter' wiki page.