Table of Contents

1. Development Guide 4
   1.1. Prerequisite 4
   1.2. Github 5
   1.3. Compilation 5
   1.4. Testing/Debugging 6
   1.5. Running Scripts 6
2. Driving of willy 9
   2.1. Stop Willy 9
3. Willy Wiki 11
   3.1. Introduction 11
   3.2. Why AsciiDoc 12
   3.3. How the Wiki is setup using AsciiDoc with TravisCI and Github Pages 12
   3.4. Conversion 12
      3.4.1. Travis 12
         Setup 12
   3.5. Publishing 15
      3.5.1. Github Pages 15
   3.6. Further reading 16
   3.7. References 16

Welcome

Project Willy

- History of Willy
- Project Willy
- Publicity
- Sponsors

Getting started

- Development Guide
- Driving Willy
- Documentation

Build of Willy

- Design history
- Requirements
- Design reference
- Physical build
- Hardware

Robotic Operating System

- Introduction to ROS
- ROS Tutorials
- Multi master

Architecture

- Software Architecture
- Hardware Architecture
- Skylab Architecture
- ROS topic design

Hardware nodes

- sensor node
- si node
- power node
- WillyWRT

Components

- ROS master
- New ROS master on Lubuntu
- Brain

- Sonar
- Lidar
- Localization and navigation
- Motor controller
- Joystick
- Social interaction
- Speech
- Speech recognition

Skylab

- Setup Skylab
- Python scripts
- Webserver
- Functions of the webserver
- Skylab servers
- ROS installation on Ubuntu VMs in Skylab
- DNS,DHCP, pfSense & Ubuntu

Radeffect App

- Radeffect App

Lessons learned

- Todo & Advice
- Lessons Learned

Archive

- Previous Groups
- Research Archive

Welcome

Project Willy

- History of Willy
- Project Willy
- Publicity
- Sponsors

Getting started

- Development Guide
- Driving Willy
- Documentation

Build of Willy

- Design history
- Requirements
- Design reference
- Physical build
- Hardware

Robotic Operating System

- Introduction to ROS
- ROS Tutorials
- Multi master

Architecture

- Software Architecture
- Hardware Architecture
- Skylab Architecture
- ROS topic design

Hardware nodes

- sensor node
- si node
- power node
- WillyWRT

Components

- ROS master
- New ROS master on Lubuntu
- Brain
- Sonar
- Lidar
- Localization and navigation
- Motor controller
- Joystick
- Social interaction
- Speech
- Speech recognition

Skylab

- Setup Skylab

- Python scripts
- Webserver
- Functions of the webserver
- Skylab servers
- ROS installation on Ubuntu VMs in Skylab
- DNS,DHCP, pfSense & Ubuntu

Radeffect App

- Radeffect App

Lessons learned

- Todo & Advice
- Lessons Learned

Archive

- Previous Groups
- Research Archive

1. Development Guide

1.1. 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)
```

1.2. Github

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

```
git clone https://github.com/Windesheim-Willy/repo-name
```

- Switch to test branch

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

1.3. 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
```

1.4. 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
```

1.5. 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
```

Welcome

Project Willy

- History of Willy
- Project Willy
- Publicity
- Sponsors

Getting started

- Development Guide
- Driving Willy
- Documentation

Build of Willy

- Design history
- Requirements
- Design reference
- Physical build
- Hardware

Robotic Operating System

- Introduction to ROS
- ROS Tutorials
- Multi master

Architecture

- Software Architecture
- Hardware Architecture
- Skylab Architecture
- ROS topic design

Hardware nodes

- sensor node
- si node
- power node
- WillyWRT

Components

- ROS master
- New ROS master on Lubuntu
- Brain
- Sonar
- Lidar
- Localization and navigation
- Motor controller
- Joystick
- Social interaction
- Speech
- Speech recognition

Skylab

- Setup Skylab
- Python scripts
- Webserver
- Functions of the webserver
- Skylab servers
- ROS installation on Ubuntu VMs in Skylab
- DNS,DHCP, pfSense & Ubuntu

Radeffect App

- Radeffect App

Lessons learned

- Todo & Advice
- Lessons Learned

Archive

- Previous Groups
- Research Archive

2. Driving of willy

1. **Deploy the brakes**
   Normally a wheelchair has brakes. The brakes of Willy are removed.
2. **Deploy the emergency stop**
3. **Turn the power on**
   Do it in the following order:
   - Start the OpenWRT PI (Which is removed, because this was a private PI)
   - Start the Notebook
   - Start the Senor and SI PI.
4. **Undeploy the emergency stop**

❗ By default the **brain** will takeover the control. Make sure first move Willy with a Joystick to a safe place.

2.1. Stop Willy

1. **Deploy the emergency stop**
2. **Shutdown all raspberry PI's from the notebook**
   a. Open the terminal on the notebook

```
sh /opt/willy/components/ansible/shutdown-pi.sh
```

Fill in the password of the PI's

1. **Shutdown the notebook**
2. **Turn of the power**

- History of Willy
- Project Willy
- Publicity
- Sponsors

Getting started

- Development Guide
- Driving Willy
- Documentation

Build of Willy

- Design history
- Requirements
- Design reference
- Physical build
- Hardware

Robotic Operating System

- Introduction to ROS
- ROS Tutorials
- Multi master

Architecture

- Software Architecture
- Hardware Architecture
- Skylab Architecture
- ROS topic design

Hardware nodes

- sensor node
- si node
- power node
- WillyWRT

Components

- ROS master
- New ROS master on Lubuntu
- Brain
- Sonar
- Lidar

- Localization and navigation
- Motor controller
- Joystick
- Social interaction
- Speech
- Speech recognition

Skylab

- Setup Skylab
- Python scripts
- Webserver
- Functions of the webserver
- Skylab servers
- ROS installation on Ubuntu VMs in Skylab
- DNS,DHCP, pfSense & Ubuntu

Radeffect App

- Radeffect App

Lessons learned

- Todo & Advice
- Lessons Learned

Archive

- Previous Groups
- Research Archive

3. 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.

3.1. 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

3.2. 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]

3.3. 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.

3.4. 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.

3.4.1. 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.

Travisci is already configured for the windesheim-willy.github.io/WillyWiki/ repository. No additional configuration is required.

===== How-to 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"
```

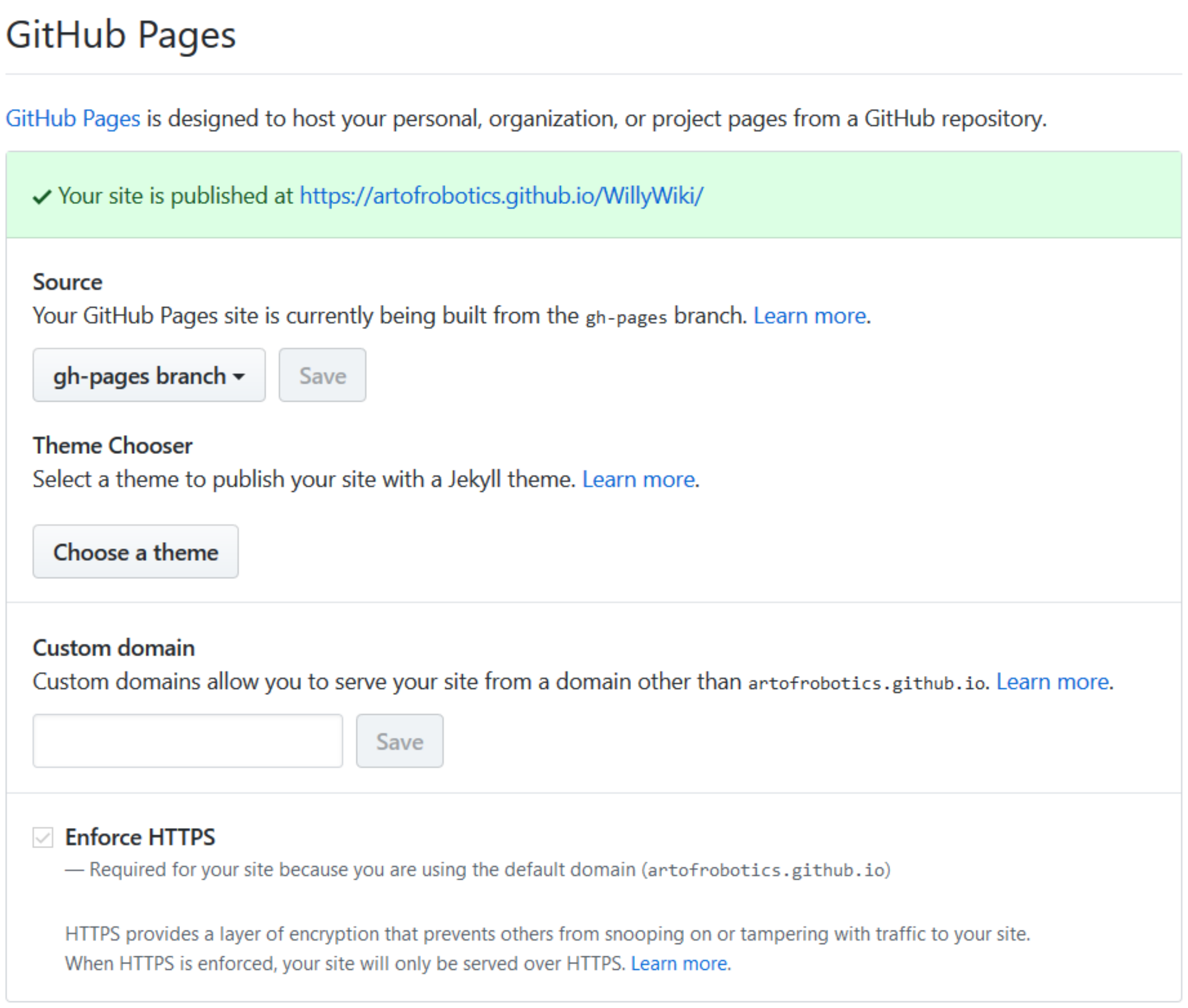
3.5. Publishing

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

3.5.1. 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 Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

✓ Your site is published at https://artofrobotics.github.io/WillyWiki/

Source
Your GitHub Pages site is currently being built from the `gh-pages` branch. Learn more.

Theme Chooser
Select a theme to publish your site with a Jekyll theme. Learn more.

Custom domain
Custom domains allow you to serve your site from a domain other than `artofrobotics.github.io`. Learn more.

Enforce HTTPS
— Required for your site because you are using the default domain (`artofrobotics.github.io`)

HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site.
When HTTPS is enforced, your site will only be served over HTTPS. Learn more.

3.6. 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.

3.7. 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