Table of Contents

1. Development Guide . . . 2
  1.1. Prerequisites . . . 3
    1.1.1. General . . . 3
    1.1.2. C++ Development . . . 3
    1.1.3. Python Development . . . 5
    1.1.4. Social Interaction Node . . . 5
  1.2. Github . . . 6
  1.3. Compilation . . . 6
  1.4. Testing/Debugging . . . 6
  1.5. Running Scripts . . . 7

Components

- ROS master
- New ROS master on Lubuntu
- Sonar
- Lidar
- Kinect
- Localization and navigation
- Motor controller
- Joystick

Lessons learned

- Todo & Advice
- Lessons Learned

Archive

- Previous Groups
- Research Archive
- Skylab Architecture
- Skylab
- Multi master
- WillyWRT
- Realisation
- Hardware
- Brain
- Design Guild
- Social interaction
- Speech
- Speech recognition
- IMU
- Human Detection
- Radeffect App

1. Development Guide

The software components written for Willy The Robot are diverse but kept to three major languages. These are Python, C++ and Javascript. The following guide shall explain general instructions for developing in each language within the project.

1.1. Prerequisites

1.1.1. General

Knowledge of the Publisher/Subscriber model within ROS is required, as it will simplify the process of reading and sending messages from your new script to the various components such as the navigation stack.

You will also need a coding IDE or (text) editor. Git + a client interface (eg. SourceTree or Gitkraken) is necessary in order to "push" and "pull" the existing code from the Github Repository.

1.1.2. C++ Development

C++ scripts, nodes and/or applications are built using the Catkin environment within ROS. You can find more information within the tutorials upon the ROS wiki.

Generally, you will start with developing a node in a catkin package, which needs to be created via the following commands, this must be done within the catkin workspace;

```
cd ~/catkin_ws/
catkin_create_pkg <your new package>
cd <your new package>
```

To compile the packages within the workspace, it is recommended to run the following commands in order. The `-j4` option tells the compiler how many CPU cores can be used. By default, this is set to 2, but the Raspberry Pi 3b+ has four cores available to utilize.

```
catkin_make clean
catkin_make -j4
catkin_make install
```

An alias for this process is available, type the command: `catbmi` (Build, Make, Install) and the list above will be run automatically in order.

Important to note: Compilation on a Raspberry Pi can often result in the OS running out of (virtual) memory. To mitigate this, it is urged to use a spare 4GB or larger USB drive as swap space. The following commands will let you set this up, but do note that administrative privileges are required and that the USB drive could experience strain when being used as a makeshift RAM stick.

First, connect the USB drive to the Raspberry Pi and use `fdisk` to view all connected storage drives.

```
sudo fdisk -l
```

The USB drive will appear as `\dev\sda` or derivative below the list:

```
Disk /dev/sda: 7.5 GiB, 8068792320 bytes, 15759360 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000001

Device     Boot   Start      End Sectors  Size Id Type
/dev/sda1          2048  7372799 7370752  3.5G  c W95 FAT32 (LBA)
/dev/sda2       7372800 15757311 8384512    4G  c W95 FAT32 (LBA)
```

Note that this USB drive is divided in two partitions, the first is usable for transferring files with, the second one is to be used as swap space and would therefore not be usable on regular PC's.

To initiate a partition as swap space, use the following command:

Any disk or partition manipulation command will risk losing all files on the partition or drive itself, exercise caution and make a back-up of any important files off the drive.

```
sudo mkswap /dev/sda2
```

Then, to initiate the newly created swap space, turn it on:

```
sudo swapon /dev/sda2
```

This setting does not persist through reboots, ergo; you will have to use `swapon` every time the Raspberry Pi has been turned off or restarted.

Using the process monitor `htop`, we can see if the swap space has been successfully added:

```
  1  [                          0.0%]   Tasks: 60, 56 thr; 1 running
  2  [                          0.0%]   Load average: 0.00 0.00 0.03
  3  [||                        4.5%]   Uptime: 00:19:06
  4  [                          0.0%]
  Mem[||||||||||||||      155M/876M]
  Swp[                     0K/4.00G]   <-- That's our swap drive!
```

Once a swap partition has been activated, DO NOT DISCONNECT THE USB DRIVE. If the USB drive is disconnected before turning swap off, the system will become inoperable until it is rebooted.

Once you're finished with compiling your packages, you can use `sudo swapoff /dev/sda2` to safely disconnect the USB drive.

1.1.3. Python Development

The benefit of Python is that it lets you write and integrate small scripts quicker by skipping the need to compile packages, but it is still recommended to do so if you want to integrate the script as a node within ROS.

Running python scripts could simply be done by either navigating to the containing directory or to name the path immediately:

```
# Running from the directory
cd ~/Documents/willy/demo_script
python demo_script.py

# Running from the path
python ~/Documents/willy/demo_script/demo_script.py
```

1.1.4. Social Interaction Node

The following instructions are relevant to development on/for the Social Interaction Node.

- You will need basic knowledge of Node.js, 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
     - Go 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 access 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 node.js 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 of a simple run script which launches the webplatform

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