CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

OS	Operating System
SD card	Secure Digital card
GUI	Graphical User Interface

IoT	Internet of Things
IDE	Integrated Development Environment
PC	Personal Computer
SSH	Secure Shell
pip	Pip Installs Packages
RPC	Remote Procedure Call
SFTP	Secure File Transfer Protocol

A

PYTHON MANUAL

A.1 INTRODUCTION

This document is written in collaboration between the group of Michael Simonsen & Stefan Madsen, and Shuo Hua. It serves as a part of their master thesis. This document describes the installation and workings of an OS for LEGO® MINDSTORM EV3, called ev3dev [61]. As well as descriptions of some initial programming experiences with Python for the EV3 system. The software packages described are the ev3dev kernel version 18 [133] and the Python library python-ev3dev version 0.8.1 as of 17th of February 2017 [137].

A.2 THE EV3DEV OPERATING SYSTEM

The ev3dev is a Debian Linux-based OS for the LEGO® MINDSTORM EV3 system. The ev3dev works as a standalone OS that can be booted from external storage on EV3 e.g. a Secure Digital card (SD card). There is no need for flashing the original EV3 internal memory and OS that comes with the EV3. The ev3dev OS gives the ability to program the EV3 with different programming languages such as Python, Javascript, GO, C and C++, for which there already exist libraries that assists in controlling the EV3 system [58]. Since ev3dev is a full Linux backend, it's also possible to extend current libraries to accomplish additional goals. The ev3dev OS is also capable of running on platforms like Raspberry-Pi, though this topic will not be covered by this appendix.

A.2.1 *Installing the ev3dev OS*

The guide to getting started with the ev3dev OS can be found at [59]. To get started a microSD or microSDHC card is needed with a storage size between 2GB and 32GB to store the ev3dev OS [108]. APCwith a SD card adapter and card-reader are needed to flash the ev3dev OS onto the microSD or microSDHC card. The card reader can be internal or external.

1. Download the ev3dev Linux distribution from [126].
2. Use a program like Etcher as suggested by [59] to flash the SD card with ev3dev.

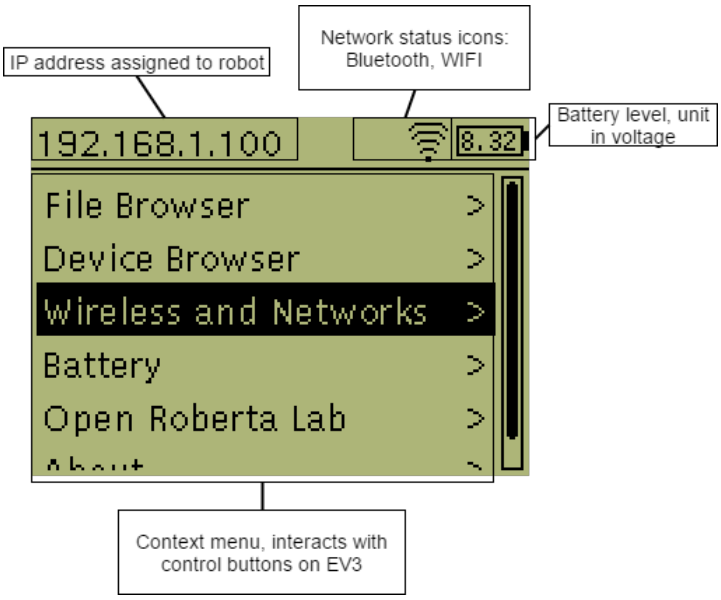


Figure A.1: Brickman, the GUI for ev3dev.

Figure A.2: A SanDisk 32 GB HC MicroSD and a Netgear N150 - WNA1000 Wi-Fidongle in front of their respective ports on the EV3.

3. Once the SD card is flashed, plug it into the EV3, see Figure A.2, and turn on the EV3.

On the first boot the new OS will do some initial setups, which might take a few minutes, but subsequent boots are faster.

When the initial setup is completed, the ev3dev OS will start the ev3dev's GUI *Brickman* [138], and the installation of ev3dev OS is complete, Figure A.1.

Once the ev3dev OS is booted, a connection between EV3 and the PC has to be established. There are several options to choose from, USB, Bluetooth and Wi-Fi, see Figure A.3. Through these connections full Internet connectivity can be established on the EV3, if Internet connectivity is not required, tethering [128, 129], which only allows

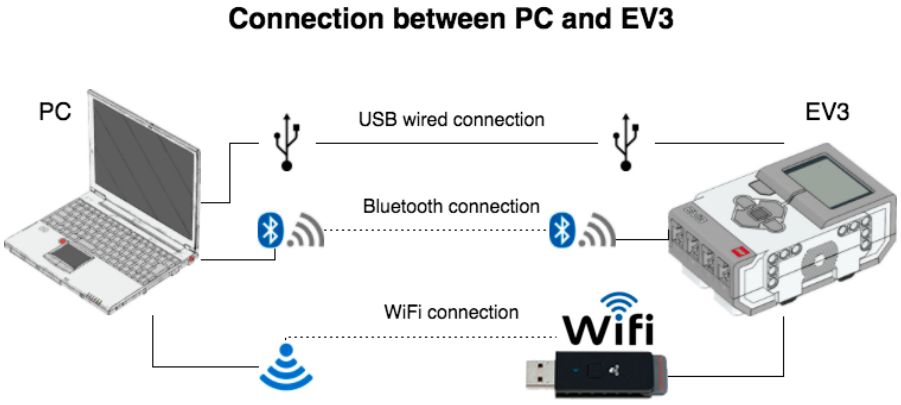


Figure A.3: There are three options to connect the EV3 from a PC: Wired USB, Bluetooth and Wi-Fi.

for local communication see Section A.2.1.2, can be used with these options as well.

A.2.1.1 *Establish Internet connectivity on the EV3*

With Internet connectivity the EV3 becomes a full blown part of the Internet of Things (IoT) [98], and it enables various TCP/IP connections [99] for communicating with the EV3. Also it allows for updating the ev3dev OS [60] without having to flash the SD card all over again. The EV3 can be connected to the Internet either with a Wi-Fidongle or by connecting it to a PC via USB or Bluetooth and then sharing the PC's Internet connection.

The simplest way to connect to the Internet, is to use a Wi-Fidongle connected through the EV3's USB port, see **??**. See Section A.2.1.3 for compatible Wi-Fidongles. Once the Wi-Fidongle is plugged in, use the Brickman GUI, Figure A.1, *Wireless and Networks* → *Wi-Fi* menu to connect to a network [57].

It is difficult to establish an Internet connection though Bluetooth on Windows 10, due to the lack of documentation for Windows installations [125]. Instead a USB connection can be used and an Internet connection can be established through it. For Linux and OSX installations, it is quite straight forward to connect to the Internet, by following the guides in [123, 124].

A.2.1.2 *Tethering to a Host PC*

If internet connectivity is not required, tethering can be used, but this only allows to communicate and transfer files to the EV3 locally. For Windows installations, one needs to join a Bluetooth personal network in order to resolve the IP address correctly, there's a guide about that in [127], other than that the official documentation on tethering, with both USB and bluetooth, is a bit sparse according to [128, 129].

A.2.1.3 *Wi-Fi dongles compatible with ev3dev*

The recommended Wi-Fidongle by the LEGO® Group is the Netgear N150-WNA1100 model [50], which is guaranteed to be compatible with the standard OS running on the EV3 [39]. The N150-WNA1100 model is also one of several Wi-Fidongles recommended by ev3dev developers for the ev3dev OS. Many other USB Wi-Fidongles which support Linux will work too but are not officially supported by ev3dev [57]. It was not possible to obtain the recommended dongle, but it was possible to test the Proware PW-DN4210D Wi-Fi dongle [1] that has the same chipset and wireless chip as the N150-WNA1100. The Proware PW-DN4210D was confirmed working with the ev3dev OS. The Netgear N150-WNA1000 Wi-Fidongle [49] was the closest model to the Netgear N150-WNA1100 that was obtainable, it was tested and confirmed working with the ev3dev OS without any issues. The N150-WNA1000 dongle has a smaller size than the PW-DN4210D and the N150-WNA1100 so it might be preferable over the two others. Other Wi-Fidongles was tested and confirmed working i.e. the dongle was inserted into the EV3, **??**, and the EV3 could subsequently connect to a wireless network and connect to the Internet through that network, see Table A.1 for more information about the dongles.

Model	Chipset	Wireless chip	Status
Netgear N150 - WNA1000	Atheros AR9001U-NG	Atheros AR9170	Confirmed working
Netgear N150 - WNA1100	Atheros AR9002U	Atheros AR9271	Not tested
Linksys WUSB100 v2	Unknown	Ralink RT3070	Confirmed working
Proware PW-DN4210D	Atheros AR9002U	Atheros AR9271	Confirmed working

Table A.1: Wi-Fi dongles compatible with ev3dev OS. The different Wi-Fi dongles are listed with the model name, the chipset [84], the wireless chip and status of the current dongle running with the ev3dev operation system. The dongles marked *Confirmed working* have been tested as described in Section A.2.1.3.

A.3 REMOTE TERMINAL TO THE EV3

Once the EV3 is connected to a PC, as described in Section A.2.1.1 and Section A.2.1.2, it might take a few minutes for the PC to discover the host name of the EV3. If the PC is unable to resolve the host name as *ev3dev* or *ev3dev.local*, then the local IP, which can be seen on the EV3's display, see Figure A.1, can be used as a substitute for the host name.

By using the ev3dev OS standard login:

USER NAME: robot

PASSWORD: maker

HOST NAME: ev3dev or ev3dev.local

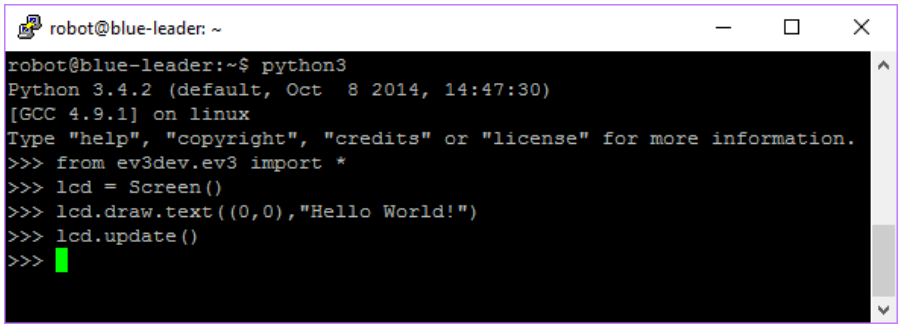


Figure A.4: A terminal running on an EV3 with a host name changed to *blue-leader*.
1. line: Starts the Python3 interpreter.
5. line: Importing the *ev3dev-python-lang* language bindings.
5. line: Getting an instance of the EV3 screen.
6. line: Setup the desired text in the upper left hand corner of the screen.
7. line: Telling the EV3's screen to update, after which the screen on the EV3 will shortly display the text *Hello World!*.

A SSH connection [109] can now be established to remotely log into the EV3 and start a remote terminal on the PC, see Section A.3.1 and Section A.3.2 for how to do this on Windows and Linux/OSX respectively.

The first time the PC makes a SSH connection to the EV3, a security prompt concerning the authenticity of the host will be displayed on the PC, this should just be answered with a yes.

Once logged on to a remote terminal running on the EV3, it is possible to start the Python3 interpreter by typing *python3* into the terminal and hitting enter. In Figure A.4, it is shown how to use the Python3 interpreter to quickly write a text to the screen on the EV3.

The small *Hello World!* example in Figure A.4 also highlights a problem regarding the screen on the EV3. The ev3dev OS comes with a menu system called Brickman [138], and whenever a program or a command wants to use the screen on the EV3, that program or command is competing with Brickman for control. So when running the commands in Figure A.4 the text *Hello World!* is potentially only shown on the screen for a few seconds before Brickman retakes control. This issue and some different solutions are further discussed in Section A.5.2 and Section A.5.7.

If multiple EV3's are connected to the same network or PC, it becomes difficult to tell which EV3 the SSH session is connected to. To alleviate this, the standard host name can be changed with a configuration tool. To do so, first connect to the EV3 as described in Section A.3.1 and Section A.3.2, then start the configuration tool by running the following command in the remote terminal:

```
sudo ev3dev-config
```

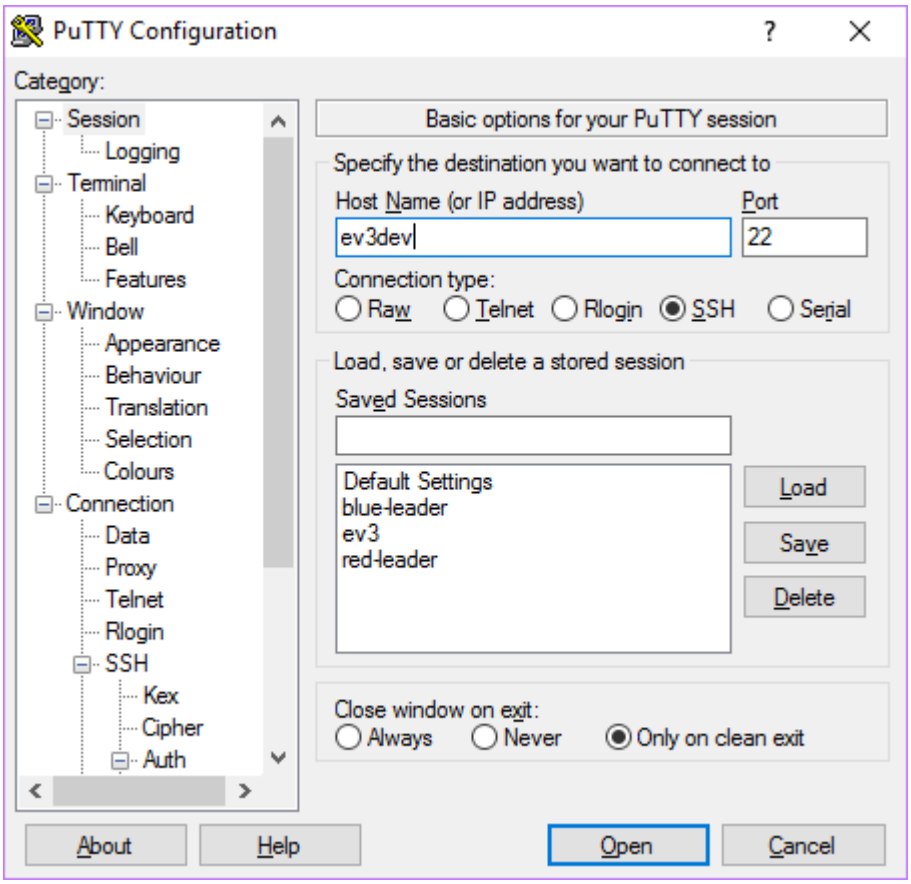


Figure A.5: The PuTTY [67] program setup to connect to the EV3 via the ev3dev OS's standard hostname: *ev3dev* on the SSH standard port: 22 [109].

A.3.1 *Remote terminal on Windows*

On windows a program like PuTTY [67] can be used to remotely log onto the EV3. Pushing *Open* in Figure A.5 will open a terminal window on the PC of a terminal running on the EV3.

A.3.2 *Remote terminal on OSX/Linux*

On OSX or Linux OS a SSH connection can be established directly from a terminal [33]. Connect by using the host name *ev3dev.local*:

```
ssh robot@ev3dev.local
```

Running the above command will open a terminal window on the PC of a terminal running on the EV3.

A.4 UPLOADING FILES TO THE EV3

Uploading files to the EV3 can either be done manually through *scp* [110], e.g. using WinSCP [121] or directly from a terminal [81]. Alternatively the EV3 can be setup as host of a git repository [20], or the PyCharm Professional Edition Integrated Development Environment (IDE) can be setup to manage the deployment of files to the EV3.

A.4.1 *Git*

Git [20] is er version control system, and the rest of this section assumes that the reader already posses a basic understanding of Git and how it works.

To setup the EV3 as host of a git repository, connect to the EV3 via SSH, Section A.3, and execute the following commands in the remote terminal to make sure that git is installed and up to date:

```
sudo apt-get update
sudo apt-get install git
git config --global user.email "your@example.com"
git config --global user.name "Your Name"
```

Then setup the project folder and initialise the git repository:

```
mkdir myproject.git
git init --bare myproject.git
mkdir myproject
```

After setting up the project, edit the git's post-receive hook [19] to automatically deploy code pushed to the repository, by editing the hook file in the text editor *nano*, which is pre-installed with the ev3dev distribution.

```
nano myproject.git/hooks/post-receive
```

Add the following lines to the post-receive hook file, then save (Write-Out) and exit. The second line deploys the project on the EV3, and the last line gives all Python files execution permissions.

```
#!/bin/sh
git --work-tree=/home/robot/myproject --git-dir=/home/
    ↪ robot/myproject.git checkout -f
chmod +x /home/robot/myproject/*.py
```

Finally the hook file needs to be made executable by:

```
chmod +x myproject.git/hooks/post-receive
```

The repository can now be cloned onto the local PC by using the URL *robot@ev3dev:myproject*, see Figure A.6. On OSx and Linux it may

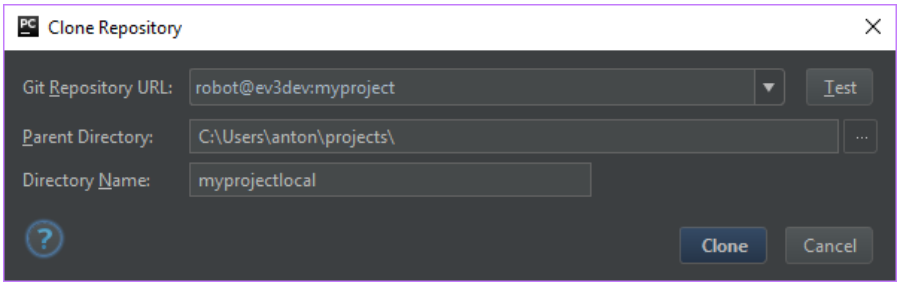


Figure A.6: An example of cloning a repository from an EV3 running ev3dev on windows using PyCharm.

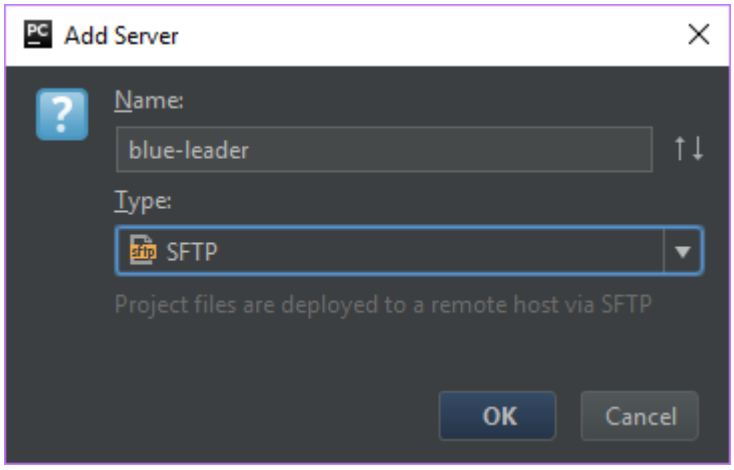


Figure A.7: To add a new deployment server, choose SFTP and give it a name.

need to be *robot@ev3dev.local:myproject* or failing that, the IP from the display can be used as a substitute for *ev3dev.local*, see Section A.3.

Once all this has been setup, assuming the PC and EV3 is on the same network, then getting programs onto the EV3 is a matter of creating a commit and pushing it to the repository that has been created on the EV3, and due to the edited hook file, all uploaded programs will be given executable permissions, meaning the programs can be run from Brickman, assuming the programs have a correct shebang.

A.4.2 *PyCharm and deployment of files*

PyCharm Professional edition can be set up to deploy files directly to *ev3dev* [2]. First make sure the EV3 is turned on, then in PyCharm go to Tools → Deployment → Configuration..., in the deployment window hit the plus sign in the upper left corner. In the *add server* window, Figure A.7, choose type SFTP[88] and give the new deployment server a name.

In the next window, Figure A.8:

1. Enter *SFTP host* as *ev3dev*, unless host name has been changed.
2. Enter *robot* in user name.

3. Enter *maker* in password
4. Hit *Test SFTP connection* to see if PyCharm can establish a connection.
5. Push *Autodetect* to set the rootfolder.
6. Go to the *Mappings* tab.

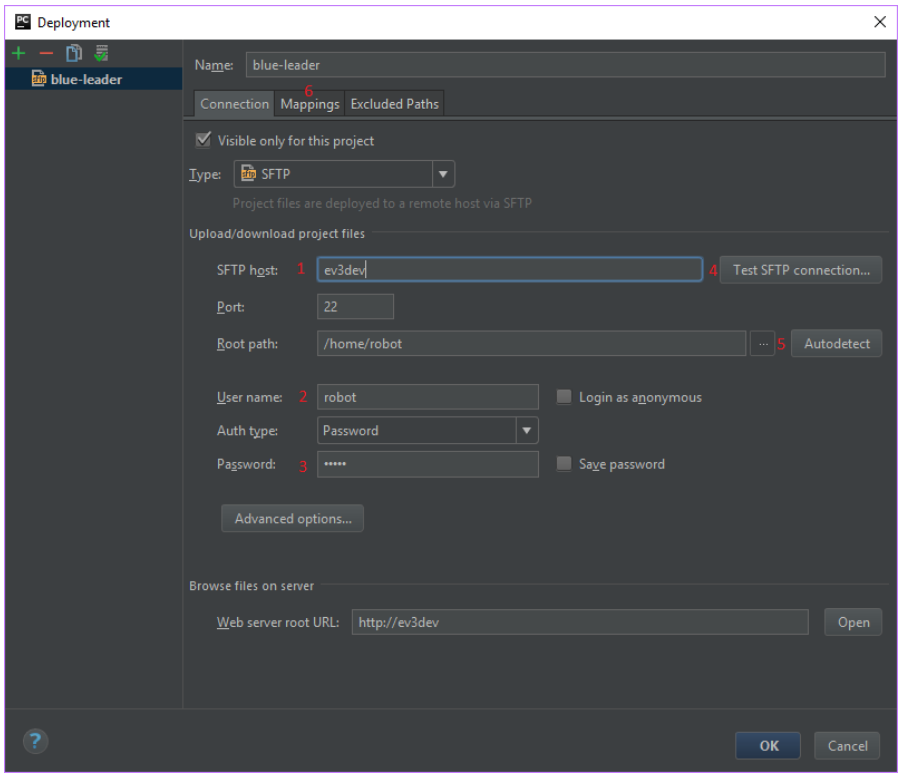


Figure A.8: To setup a connection to a deployment server, e.g. an EV3 , enter host name, user name and password.
(1) Host name of deployment server, ev3dev.
(2) Deployment server user name, robot.
(3) Deployment server password, maker.
(4) Test deployment server setup.
(5) Autodetect root folder.

In the *Mappings* tab, Figure A.9, the *local path* (1) has already been set, assuming the correct project was open when starting the setup process. *Deployment path...* (2) can be used if deployed files should be placed in a specific folder on the EV3, hitting the button with three dots (3) opens a view of available folders on the EV3.

In PyCharm going to Tools → Deployment → Options, Figure A.10, the option *Override default permissions on files* (1) can be used to make uploaded files executable, and in *Upload changed files automatically...* (2) selecting *On explicit save action*, makes PyCharm upload a file to the EV3 whenever the file is explicitly saved, e.g. File → Save All.

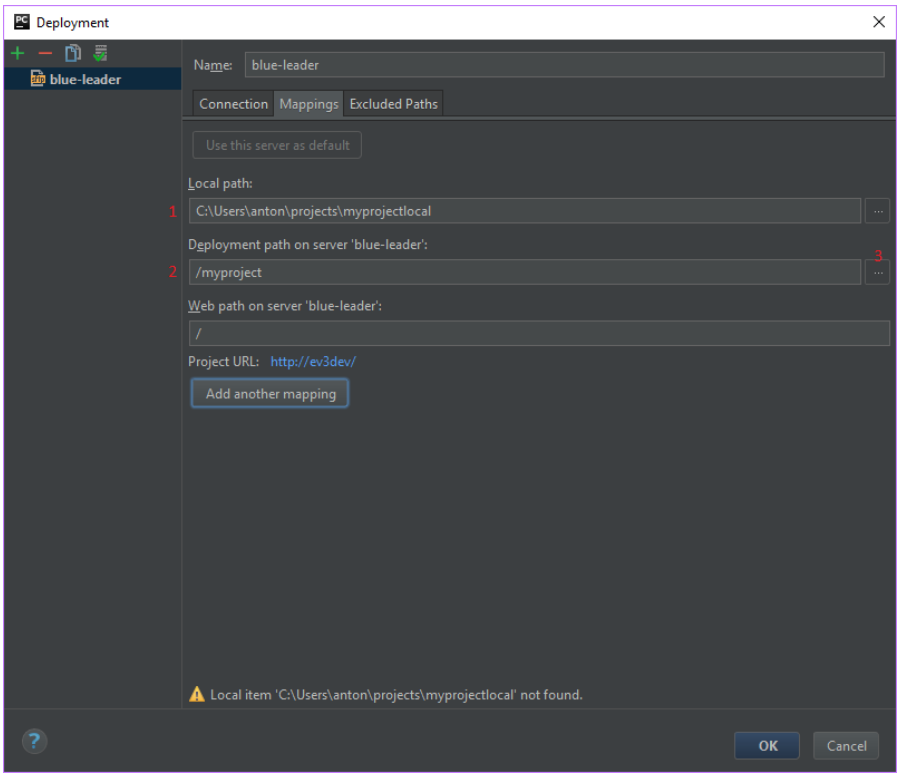


Figure A.9: It is possible to choose a deployment folder.
(1) Local path should be set to project folder on the PC.
(2) Choose a specific folder on the EV3 to deploy files to.
(3) See a list of folders already present on the EV3.

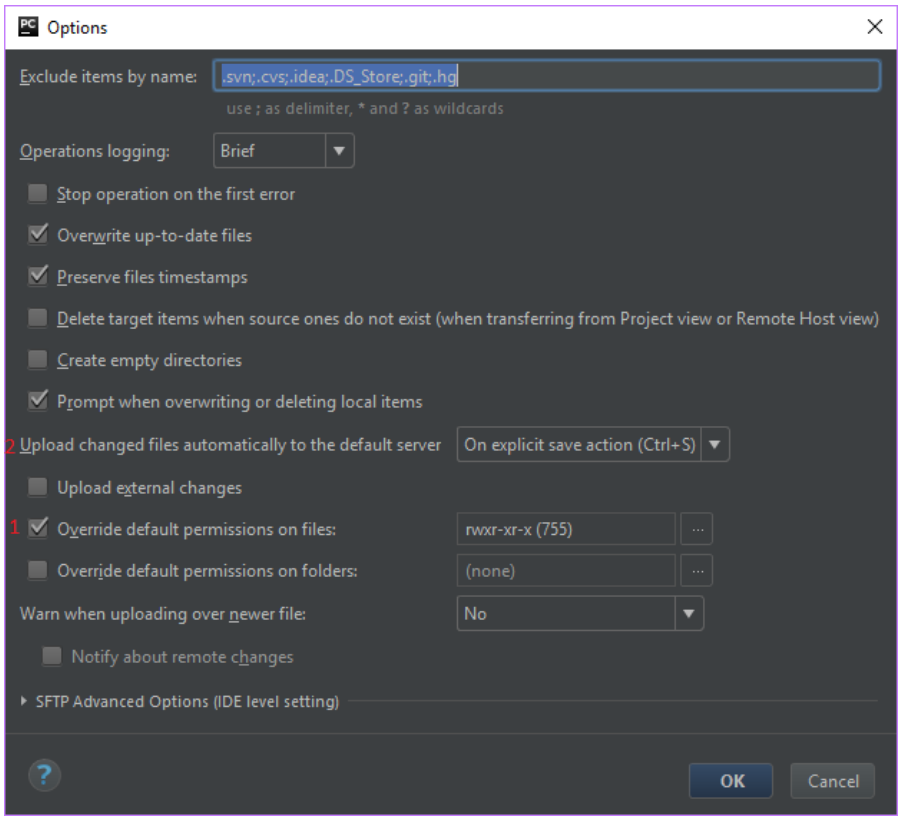


Figure A.10: In Deployment options, (1) files can be given executable permissions on the deployment server, and PyCharm can be configured to (2) upload files when they are explicitly saved.

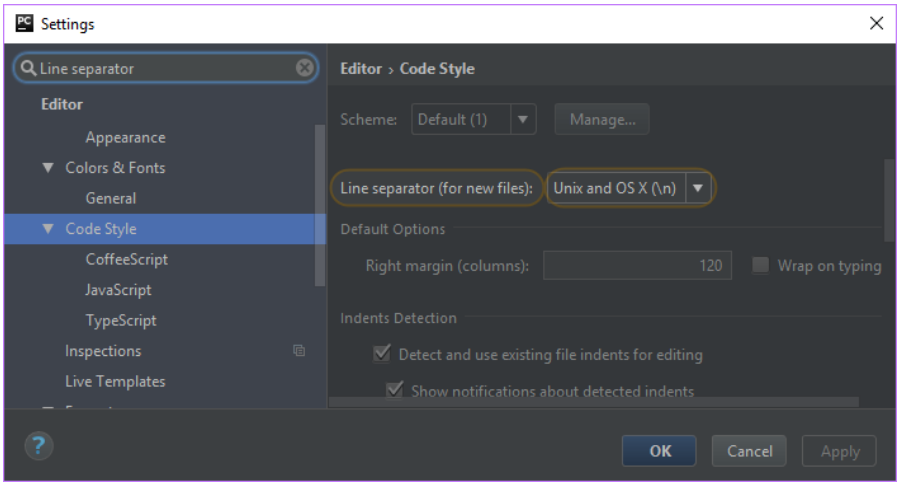


Figure A.11: As ev3dev is a Linux based OS, when PyCharm deploys files to the EV3 it needs to be told specifically to use Unix line separators.

To make files executable on the EV3 from a Windows machine, it is necessary to choose Unix line endings, this is done by marking a file in the project view, then File → Line Separators → LF. Or it can be done for the entire project by going to settings, File → Settings..., and under *Code Style* setting the line seperator to *Unix and OS X*, see Figure A.11.

Now uploading files to the EV3 can be done from PyCharms project view, by right clicking a file or folder, Deployment → Upload.

A.5 PYTHON ON EV3

There exists two Python libraries for the EV3 system. The *ev3dev-lang-python* [29] maintained by Ralph Hempel, is recommended by the ev3dev team. The other one is *python-ev3* [21], maintained by GongYi, but according to [58] *python-ev3* is either out of date, unfinished or abandoned.

We have choosen to use *ev3dev-lang-python* as Python library, as it is more stable and have documentation. The API of *ev3dev-lang-python* can be found together with the documentation [32].

Generally the API is split into 3 groups of classes, that handles the motor ports, sensor ports, and miscellaneous functions such as display, sound and buttons. Every class of *ev3dev-lang-python* lies in the *ev3dev* module and inherits from the base *Device* class.

- Motor classes [26]: These classes handles the action of motors connected through motor ports. For each kind of motor connected, there exists different Motor class implementations, such as *ev3dev.core.ServoMotor* or *ev3dev.core.DcMotor*. The motor classes are described in more detail in Section A.5.4.

- Sensor classes [27]: These classes handles the received input from sensors connected through sensor ports. For each connected sensor it creates a *LegoPort* device to communicate with it, and thus it is possible to customize sensors. For standard sensors shipped with the EV3 system, there already exists classes in the library such as *ev3dev.core.TouchSensor* or *ev3dev.core.LightSensor*. The sensor classes are described in more detail in Section A.5.5.
- Miscellaneous classes [25]: These classes provide functionality and control to different part of the EV3 system, such as power management, display, sound and remote control.

A.5.1 *Developing Python programs for ev3dev using PyCharm IDE*

A working environment for developing Python programs on ev3dev can be setup by using the free version of PyCharm IDE or the Professional version [55], which is free for students [18]. PyCharm is a Python IDE made by JETBRAINS and can be downloaded from [55]. The IDE provides helpful tools for coding Python, such as syntax checking, code completion, built-in version control and many other features. A tutorial for setting up PyCharm with ev3dev can be found on [3].

The most important part of the tutorial is to install the site-package *python-ev3dev* for Python3, as this is what allows the PyCharm IDE to perform syntax checking and code completion.

The package can be installed through Pip Installs Packages (pip) [135], a Python package manager [136], by running the following command in a terminal on the PC.

```
pip3 install python-ev3dev
```

To check for updated versions of the package, a pip upgrade command can be executed.

```
pip3 install --upgrade python-ev3dev
```

A.5.2 *Running a program*

We have found 4 different ways of running a program on ev3dev [11].

1. Brickman
2. SSH
3. SSH and chvt
4. SSH, chvt and conspy

Each method is described in greater detail below.

A.5.2.1 *Brickman*

Programs can be started directly from the EV3 using Brickman, ev3dev's standard menu system, by going to the *File Browser*, see Figure A.1, and navigating down to the program. This requires that the program has been marked as executable on the EV3 by running the following command in a remote terminal connected to the EV3, as described in Section A.3:

```
chmod +x /home/robot/myproject/yourprogram.py
```

This requires the Python program file to contain a correct shebang [89], which for python3 looks like:

```
#!/usr/bin/env python3
```

This method gives the program control over the screen and buttons.

A.5.2.2 *SSH*

Programs can also be run through the remote terminal once a SSH connection to the EV3 is establish, as described in Section A.2.1. A Python program can be explicitly run by the Python interpreter in a remote terminal by the command:

```
python3 yourprogram.py
```

This does not require a shebang or the use of the *chmod* command, as described in Section A.5.2.1 since the program is loaded with python3.

A drawback to this method is that the program have to fight for control of the screen with Brickman. The program is able to utilize the screen, however, any output to the screen will disappear behind the Brickman menu system fairly quickly, as described in Section A.5.1.

A.5.2.3 *SSH and chvt*

With a SSH connection to the EV3, as described in Section A.2.1, in the remote terminal run the command:

```
sudo chvt 6
```

This changes the screen on the EV3 to a new terminal window, see Figure A.12, one that Brickman does not use. However, as the new window is still just a terminal this method still suffers from the problems described in Section A.5.7, except that the solution proposed does not work with this method.

A.5.2.4 *SSH, chvt and conspy*

This is the best solution we have found to gain control over the EV3 screen. Like the method described above in Section A.5.2.3, first cre-

Figure A.12: The EV3 screen once the terminal has been changed with the *chvt 6* command

ate a SSH connection and change the window to one not used by Brickman, but for this method also use the command *conspy* [65]:

```
sudo chvt 6
sudo conspy
```

The *conspy* command will make the terminal window on the PC, Figure A.13a, mirror the terminal running on the EV3, Figure A.13b. This way the problem and solution mentioned in Section A.5.7 will work, as the OS commands used are now run directly on the terminal running on the EV3.

A.5.3 *Remote procedure call*

It is possible to set up RPC for Python in the ev3dev. This requires installation of *RPYC* [62] on both the PC and the EV3. With *RPYC* the EV3 can be set as a RPC server, which enables the execution of scripts on EV3, that are stored in a remote position. This gives the ability to perform distributed computing, where heavy computational tasks can be run locally and then send the results to the remote which uses these to execute some lightweight procedures [122].

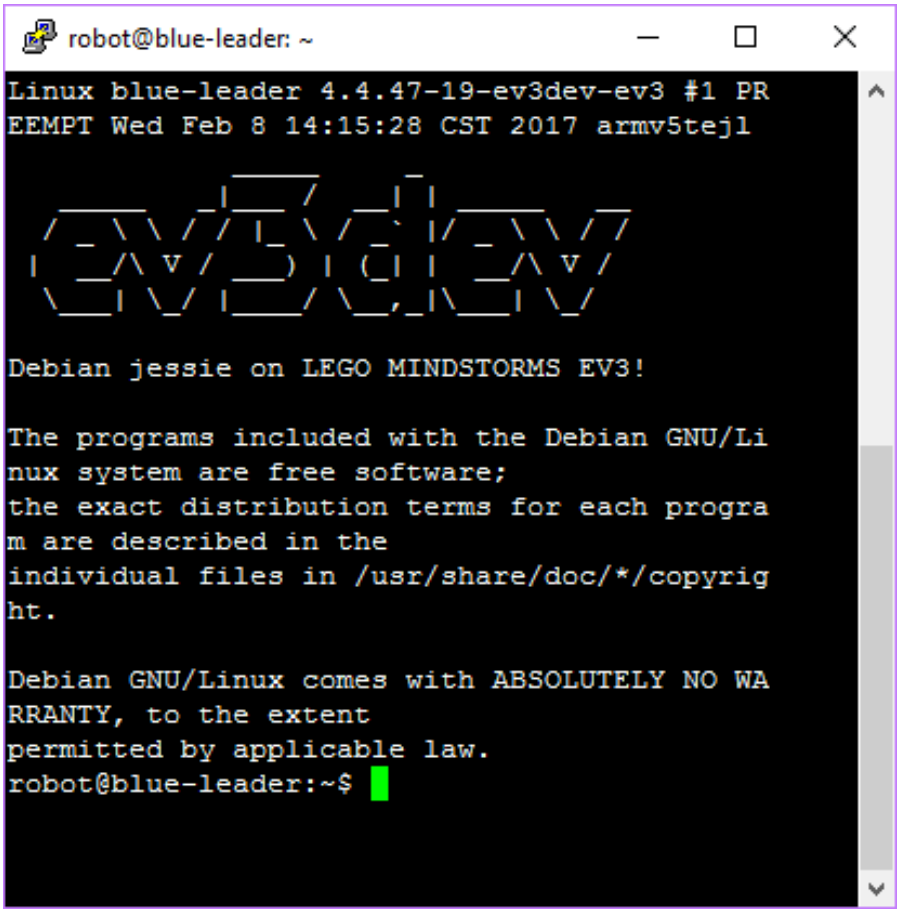
To install *RPYC* on the EV3, run the following command in a remote terminal, Section A.3:

```
sudo easy_install3 rpyc
```

While on PC it is

```
pip install rpyc
```

Afterward it is required to create a shell-script on the EV3 with the following lines:



(a) Local terminal mirroring terminal running on the EV3 after using first the *chvt 6* and then the *conspy* command.

(b) Running terminal being shown on the EV3 screen after using first the *chvt 6* and then the *conspy* command.

Figure A.13: A terminal running on the EV3 can be mirrored in a terminal running on a PC using the *conspy* command.

```
#!/bin/bash
python3 `which rpyc_classic.py`
```

And remember to give it the execution privileges with the *chmod* command.

```
chmod +x rpc_server.sh
```

By executing this shell-script, it initiates a classic RPC server, if successful it should output something like:

```
INFO:SLAVE/18812:server started on [0.0.0.0]:18812
```

Then to execute scripts remotely, it is required to specify the connection to the RPC server by giving the host name or IP address. Then import the core module from *RPYC* remotely, so the scripts should look like the following:

```
import rpyc
conn = rpyc.classic.connect('ev3dev')
ev3 = conn.modules['ev3dev.ev3']

## your code ##
```

Since *RPYC 3.0*, it's possible to define services that expose a well-defined set of capabilities to the other party, which makes RPYC a generic RPC platform. Such that *RPYC* can be setup as a server in the traditional server-client architecture, rather than just a remote interpreter. A tutorial of how to achieve that can be found at [80].

A.5.3.1 *Latency test on RPC*

Like in all distributed systems, the system response time will be worsened due to the latency introduced by the network connection, this would limit the usefulness of the RPC, especially for systems where reaction speed is essential [122].

To further investigate the effect of how the response time affects the robot, a small experiment is conducted. The experiment lets a EV3 robot equipped with a color sensor drive forward on a dark surface, the robot stops when the color sensor detects a lighter surface, while time is measured to see if there is a difference in running time between a program stored on EV3, or a program issued through RPC. The experimental code of both the locally stored program and the RPC program can be found as Listing A.1 and Listing A.2.

Three experiments are conducted, one by direct bluetooth connection, another by Wi-Ficonnection through a local router, and one without RPC. The results can be seen in Table A.2.

1 These numbers are severely affected by network traffic, thus the response time is inconsistent. The number showed here are the best results obtained. It is advised to establish a direct Wi-Fi connection between the EV3 and the local machine instead.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
import time

#initialis the motor and sensor devices
left = LargeMotor('outB')
right = LargeMotor('outC')
sensor = ColorSensor('in1')

#notify that initialisation is completed and get the
    ↪ start time in ms
Sound.tone([(440,100)])
st = time.time()*1000

#get the time before starting the left motor
t1 = time.time() * 1000
left.run_forever(speed_sp=300)
#timing the execution of the statement above
left_t = time.time() * 1000 - t1

#get the time before starting the right motor
t2 = time.time()*1000
right.run_forever(speed_sp=300)
#timing the execution of the statement above
right_t = time.time() * 1000 - t2

#the exit statement
while not Button().enter:
    if sensor.reflected_light_intensity > 50:
        left.stop()
        right.stop()
        break

 # get the end time in ms and notify the end of program
et=time.time()*1000
Sound.tone([(440,50),(220,100)])

#print the results to the console
print('time used on starting the left motor: {}'.format
    ↪ (left_t))
print('time used on starting the right motor: {}'.
    ↪ format(right_t))
print('running time: {}'.format(et-st))
```

Listing A.1: The experimentation code of the locally stored program

```
#!/usr/bin/env python3
import rpyc
import time

#initialise the RPC connection
conn = rpyc.classic.connect('192.168.1.103')
ev3 = conn.modules['ev3dev.ev3']

#initialis the motor and sensor devices
left = ev3.LargeMotor('outB')
right = ev3.LargeMotor('outC')
sensor = ev3.ColorSensor('in1')

#notify that initialisation is completed and get the
    ↪ start time in ms
ev3.Sound.tone([(440,100)])
st = time.time()*1000

#get the time before starting the left motor
t1 = time.time() * 1000
left.run_forever(speed_sp=300)
#timing the execution of the statement above
left_t = time.time() * 1000 - t1

#get the time before starting the right motor
t2 = time.time() * 1000
right.run_forever(speed_sp=300)
#timing the execution of the statement above
right_t = time.time() * 1000 - t2

#the exit statement
while not ev3.Button().enter:
    if sensor.reflected_light_intensity > 50:
        left.stop()
        right.stop()
        break

# get the end time in ms and notify the end of program
et = time.time()*1000
ev3.Sound.tone([(440,50),(220,100)])

#print the results to the console
print('time used on starting the left motor: {}'.format
    ↪ (left_t))
print('time used on starting the right motor: {}'.
    ↪ format(right_t))
print('running time: {}'.format(et-st))
```

Listing A.2: The experimentation code of the remotely stored program

Figure A.14: The response time experiment setup

Connection type	total running time	latency between commands
Local	~3000 ms	~20 ms
Bluetooth	~3500 ms	~100 ms
WIFI[1]	~5000 - ∞ ms	~40 - ∞ ms

Table A.2: Results of the response time experiment

The results show that the network latency introduced by RPC affects the response time severely. Not only does the latency make the robot react slowly, but the latency between each command affects the robots behavior. During the experimentation, as to move the robot forward, the robot starts its left motor before the right motor as shown in Listing A.3. But it is observed that during RPC, due to the latency, this have resulted in the robot twitching a little to the right side before going forward.

A.5.4 *Python program controls motors*

There are five motor classes in the API [32]. The class `ev3dev.core.Motor` controls a motor using the tacho counter, the speed of the motor has to be set before or on calling a command by setting the `speed_sp` attribute. The speed unit is tacho count per second where the LEGO® motors use degree per second, this can be converted by using the `count_per_rot` attribute. A negative speed value will cause the motor to rotate in reverse except in the *run-to-position* commands, which uses the absolute value of the `speed_sp` attribute. The maximum speed value that can be set depends on which type of motor is connected, use the property `max_speed` to get the maximum value that is accepted by the `speed_sp` attribute. If a value

```
left.run_forever(speed_sp=300)
right.run_forever(speed_sp=300)
```

Listing A.3: Method run_forever used in testing RPC latency

```
m = Motor(OUTPUT_B)

# Print the motors max speed
print("max speed {0}".format(m.max_speed))

# Set motor to run at max speed for 3000 milliseconds
m.run_timed(time_sp=3000, speed_sp=m.max_speed)

# Block as long a motor is running (3000 miliseconds)
m.wait_while("running")
```

Listing A.4: Example of running a motor for a set amount of time.

```
m = Motor(OUTPUT_B)

# Start motor at max speed
m.run_forever(time_sp=m.max_speed)

# Wait for back button to be pushed
while not btn.backspace:
    sleep(0.01)

# Stop the motor
# This has to be done explicitly, otherwise the motor
   ↪ will continue running, even after program
   ↪ termination
m.stop()
```

Listing A.5: Example of running a motor until it is explicitly stopped.

greater then `max_speed` is assigned to `speed_sp` the motor class will throw an exception. The maximum speed value is a theoretical maximum speed and can differ slightly from the actual maximum speed of a particular motor. The motor class uses a PID regulation [103] to maintain the correct speed, as defined by the `speed_sp` attribute. The motor class provides a low-level control over the motor in general, where parameters can be tweaked and information about the current state of the motor can be retrieved, e.g. instead of setting the `speed_sp` attribute it is possible to set the `duty_cycle_sp` [105] attribute instead, see an example in Listing A.6.

The two different motor classes `ev3dev.core.LargeMotor` and `ev3dev.core.MediumMotor` serves the purpose of loading the right drivers for the EV3 large motor and EV3 medium motor respectively, and both use the `ev3dev.core.Motor` class described above to control the motor.

```
motor = Motor(OUTPUT_B)

motor.duty_cycle_sp = 0

motor.run_direct()

for x in range(0,100):
    sleep(1)
    motor.duty_cycle_sp = x

sleep(3)
motor.duty_cycle_sp = 0
```

Listing A.6: Example of using the low level `duty_cycle_sp` attribute and the `run_direct()` method. Slowly ramps up the duty cycle from 0% to 100% over 100 seconds.

The class `ev3dev.core.DcMotor` provides a simple interface for using a regular DC motor this includes the LEGO® MINDSTORMS RCX motors and the LEGO® Power Functions motors. The class does not provide complicated controls or feedback. Similar to the `speed_sp` attribute in the motor class a `duty_cycle_sp` attribute has to be set, this sets the PWM signal [105] sent to the motor, then the motor can simply be started and stopped. Using a DC motor is almost identical with the example in Listing A.6, but older DC motors cannot be auto detected by ev3dev, so it is necessary to explicitly tell ev3dev that a DC motor is connected to a specific port [28], and after the DC motor is initialised it is necessary to wait a second before attempting to set any of the attributes or running any of the methods. This is probably related to the way ev3dev handles motors and sensors, i.e. ev3dev uses the file system to control sensors and motors, so these files need time to be created. The example in Listing A.7 has the same effect on a DC motor as Listing A.6 has on a standard EV3 motor.

The last motor class is `ev3dev.core.ServoMotor` which can be used for hobby type servo motors, more information about this class can be found in the documentation [32].

A.5.4.1 *Motor.wait_while('running')*

The EV3 motor has a method `wait_while(s)`, [30], which blocks as long as the motors state includes `s`, so this could be used like `Motor().wait_while('running')` to wait for a motor to stop running. However due to the way this has been implemented, see Listing A.9, it can block program execution even though the motor is not running, e. g. running the code in Listing A.8 will block even though the motor is not running.

```
# Tell ev3dev a DC motor is connected to port B
port = LegoPort(OUTPUT_B)
port.mode = "dc-motor"

motor = DcMotor(OUTPUT_B)

# Wait for system files to be created
# In our experience, not waiting causes an exception to
    ↪  be thrown
sleep(1)

# Now the DC motor is ready to be used
motor.duty_cycle_sp = 0
motor.run_direct()

for x in range(0,100):
    sleep(1)
    motor.duty_cycle_sp = x

sleep(3)
motor.duty_cycle_sp = 0
```

Listing A.7: Example of using an older DC motor. Slowly ramps up the speed from 0% to 100% over 100 seconds.

```
from ev3dev.ev3 import Motor
Motor().wait_while('running')
Motor().wait_while('running')
```

Listing A.8: On a freshly added motor, i. e. the motor has received no other commands, running this code will make the second wait_while block until the motors state file is changed.

The `wait_while(s)` method is a wrapper that uses the `wait(cond)` method, so the problem originates in the `wait(cond)`. Specifically the problem occurs because the implementation utilizes that hardware access in *ev3dev* happens through the file system. The `poll` method in Listing A.9 waits for an I/O event on the motors state file, but it does this before checking what the current state actually is, which leads to the wait method blocking even though condition is already met.

```
def wait(self, cond, timeout=None):
                                ⋮
    while True:
        self._poll.poll(None if timeout is None else
            ↪ timeout)

        if timeout is not None and time.time() >= tic +
            ↪  timeout / 1000:
            return False

        if cond(self.state):
            return True
```

Listing A.9: Lines 850-875 of core.py, is part of the wait method used in wait_while(). The poll method blocks until an I/O event happens on the underlying state file, and this happens without first checking if the condition has been met. This means that if the state file already meets the condition, then the wait method will still block until the state file changes.

A.5.5 *Python program using sensors*

The API contains classes for all the LEGO® sensors, as documented in [27, 32], and they are fairly well documented. All the sensor classes are derived from the base class *Sensor*, which means that if you do not like the formatted output of the special sensor classes, the raw values can be extracted from the base class, e.g. by calling the *value* function.

The *value* function returns the value or values the sensor is currently measuring in the sensors current mode. The available modes

```
colorSensor = ColorSensor()
colorSensor.mode = "RGB-RAW"
red = colorSensor.value(0)
green = colorSensor.value(1)
blue = colorSensor.value(2)
```

Listing A.10: Example of extracting the raw rgb values from a color sensor.

can be listed with the *modes* attribute, and the mode can be set by setting the *mode* attribute. Different modes for different sensors might have a different amount of sensor values, this can be determined by reading the *num_values* attribute. So extracting the raw color values from a color sensor can be done like in the example in Listing A.10.

One of the benefits of the EV3 is that the standard LEGO® sensors identify themselves, meaning that if a robot only contains one sensor of a specific type, there is no need to identify the port it is connected to, which is why the code in Listing A.10 will work. If a robot does have multiple sensors of the same type, then it is necessary to identify the correct port, e.g. if two touch sensors are attached to ports 1 and 3 then it is required to initiate them as follows:

```
_left = TouchSensor(INPUT_1)
_right = TouchSensor(INPUT_3)
```

If a port has been specified and nothing is connected to that sensor or motor port, then the program will throw an exception.

A.5.6 *Reading values from hardware*

Reading a value from hardware is not thread safe, even if it is a constant like a motors `count_per_rot` attribute, and doing so can produce error like the two shown in Figure A.15.

A solution to this problem is to make a wrapper class, which ensures to protect access to attributes. Listing A.11 shows a wrapper class for a color sensor, the class creates a lock object which is used to protect access to the `color` attribute, meaning that as long as all threads use the some instance of the `PColorSensor` it should not produce concurrency errors.

A.5.7 *Python program using buttons*

The button class is documented in [24, 32] and the code in Listing A.12, will wait for the user to push the back button before continuing, assuming the ev3dev is running at least kernel version 18, 18-ev3dev kernel [133], check this by going to the about menu in Brickman on the EV3. On older versions button checks return true

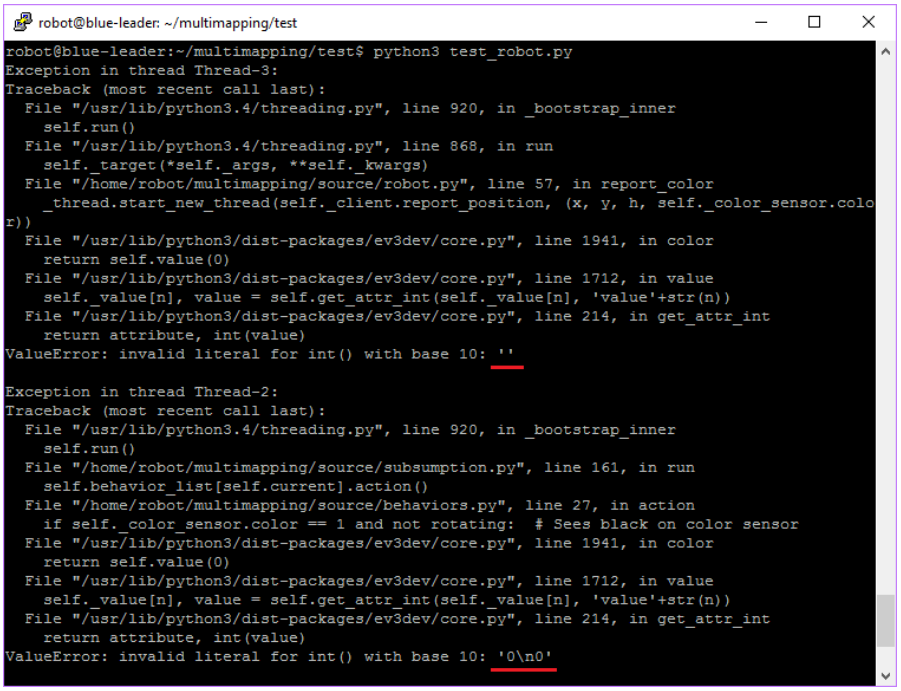


Figure A.15: Concurrently reading from a hardware attribute can causes error where it reads an empty file or file with 2 values in it, neither of which can be converted to an int.

```
class PColorSensor(ColorSensor):
    SYSTEM_CLASS_NAME = Sensor.SYSTEM_CLASS_NAME
    SYSTEM_DEVICE_NAME_CONVENTION = Sensor.
        ↪ SYSTEM_DEVICE_NAME_CONVENTION

    def __init__(self, address=None, name_pattern=
        ↪ SYSTEM_DEVICE_NAME_CONVENTION, name_exact=
        ↪ False, **kwargs):
        super().__init__(address, name_pattern,
            ↪ name_exact, **kwargs)
        self._lock = threading.Lock()

    @property
    def color(self):
        with self._lock:
            return super().color
```

Listing A.11: A thread safe wrapper class for a color sensor.

when the button is up, and false when it is down, but this has been fixed in version 18 according to [134], so that button checks return true when down.

```
btn = Button()
while not btn.backspace:
    sleep(0.01)
```

Listing A.12: Waiting for the back button to be pressed down.

```
#Wait for button down
while not _btn.enter:
    sleep(0.01)

#Wait for button up
while _btn.enter:
    sleep(0.01)
```

Listing A.13: Waiting for a button pressed, e.g. both pressed down and released again, has to be done explicitly.

The Button class does not have a method to wait for a button press, as in button both pressed down and released again, so if the intention is to wait for user input before advancing a program, it has to be done explicitly as in Listing A.13, or different buttons could be used to advance the programs through different stages. An example of this can be seen in the line follower discussed in **??**, when the color sensor is calibrated.

Also button inputs are not fully intercepted by a running program, meaning that if a program is started from a remote terminal as described in Section A.5.2.2, then any button input the program requires, will also navigate the Brickman menu system. Or if a program is started like described in Section A.5.2.3 and Section A.5.2.4, then button presses will cause output to be written to the screen, as shown in Figure A.16.

A possible workaround is to surround the Python program code with some OS commands to turn off input and cursor, see Listing A.14. This works when starting a program from Brickman or by the *conspy* method described in Section A.5.2.4. The use of a try/catch/finally block [16] in Listing A.14 ensures that the terminal works as usual even if *main()* throws an exception.

A.5.8 *Python program using LCD*

Using the LCD display is not optimal and has room for improvements, if a program is started from a terminal, then button input is not intercepted fully by the program and so it writes these to the screen, as also described above in Section A.5.7.

Figure A.16: When a program is running on the EV3 button events are not fully intercepted by the running program, but are also executed in the terminal. This results in input showing up on the EV3 screen, as marked by the red boxes, when the buttons on the EV3 are pushed.

```
if __name__ == '__main__':
    try:
        #Turn of blinking cursor and echo to avoid
            ↪ printing
        #button input
        os.system("stty -echo")
        os.system("setterm -cursor off")
        main() # Your code in main()
    finally:
        #Turn on echo and cursor again
            os.system("setterm -cursor on")
            os.system("stty echo")
```

Listing A.14: A work around to better control the EV3 screen.

```
from ev3dev.ev3 import *

#Get an object instance of the display
lcd = Screen()

#Clears the display
lcd.clear()

#Draws text on the display
lcd.draw.text((25,50),'Every thing is awesome')

#Call update on the display to apply changes
lcd.update()
```

Listing A.15: A sample program printing a text to the screen on the EV3.

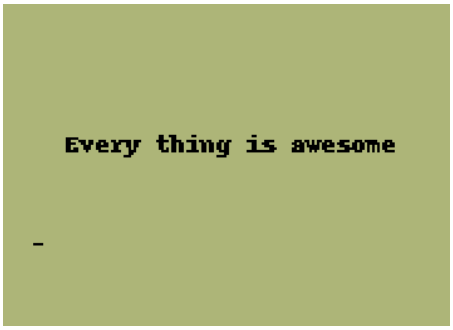
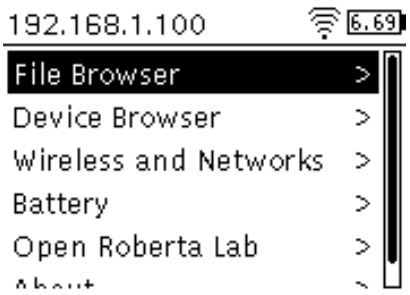


Figure A.17: Output from running the example in Listing A.15.

The EV3 has the capability of displaying 178 x 128 pixels on the monochrome LCD. The top-left corner pixel has the coordinates $\{0, 0\}$ and the bottom-right pixel has the coordinates $\{177, 127\}$ according to [11]. Most of the interaction with the display is through a graphics library called Pillow, which is a standard library in Python, see more documentation of Pillow at [53].

There are two ways to display text on the display, since the programs runs in a terminal all functions that writes output to the terminal will be displayed on the screen where the program was started, in Python the *print()* function does that. The other way is to use the Pillow library to write text on given coordinates, where the coordinates are the top left corner of the text. An example of a simple text written to the display can be seen in Listing A.15 the output of which can be seen in Figure A.17.

It is possible to save a screenshot of the current screen of EV3, by running the command `fbgrab <some-name>.png` in a terminal connected by SSH to the device, which produces an image as seen in Figure A.18a. But the LCD itself is not white, but more of a green



(a) Using the `fbgrab <some-name>.png` command.

(b) Using the `scr.py` program from [38].

Figure A.18: Screenshotting the EV3 LCD.

color, so [38] has a small program which takes a screenshot using `fbgrab` and then changes all the white pixels to a green color, producing an output as in Figure A.18b, which is the method used in this document.

BIBLIOGRAPHY

[1] *150M 1T1R High Gain Wireless N USB Adapter*. 2017. URL: http://www.proware.com.cn/product-detail.asp?productId=1010200 (visited on 03/08/2017).

[2] David Fisher Anton Vanhoucke. *Different programming languages on the ev3dev*. URL: http://www.ev3dev.org/docs/tutorials/setting-up-python-pycharm/#setting-up-an-sftp-remote-server (visited on 03/09/2017).

[3] David Fisher Anton Vanhoucke. *Setting Up a Python Development Environment with PyCharm*. URL: http://www.ev3dev.org/docs/tutorials/setting-up-python-pycharm/ (visited on 03/06/2017).

[4] Brian Bagnall et al. *ev3classes / src / lejos / robotics / navigation / DifferentialPilot.java*. URL: https://sourceforge.net/p/lejos/ev3/code/ci/master/tree/ev3classes/src/lejos/robotics/navigation/DifferentialPilot.java (visited on 03/01/2017).

[5] Victor Bahl and Venkat Padmanabhan. "RADAR: An In-Building RF-based User Location and Tracking System." In: Institute of Electrical and Electronics Engineers, Inc., Mar. 2000. URL: https://www.microsoft.com/en-us/research/publication/radar-an-in-building-rf-based-user-location-and-tracking-system/.

[6] Johann Borenstein, HR Everett, Liqiang Feng, et al. "Where am I? Sensors and methods for mobile robot positioning." In: *University of Michigan* 119.120 (1996), p. 27.

[7] Niels Olof Bouvin. *Lecture notes in Internet of Things/Peer-Networking*. Jan. 2016.

[8] R. Brooks. "A robust layered control system for a mobile robot." In: *IEEE Journal on Robotics and Automation* 2.1 (Mar. 1986), pp. 14–23. ISSN: 0882-4967. DOI: 10.1109/JRA.1986.1087032.

[9] Dídac Busquets. "A Multiagent Approach to Qualitative Navigation in Robotics." PhD thesis. Technical University of Catalonia, July 2003. URL: http://eia.udg.es/~busquets/thesis/thesis-chapters.html (visited on 04/25/2017).

[10] Ole Caprani. *Lesson 9, sumo wrestling robots*. Apr. 23, 2015. URL: http://legolab.cs.au.dk/DigitalControl.dir/NXT/Lesson9.dir/Lesson.html (visited on 03/06/2017).

[11] *EV3 Python*. Feb. 2017. URL: http://ev3python.com (visited on 03/06/2017).

[12] *EV3dev python PID LineFollower*. Youtube. Feb. 10, 2017. URL: https://youtu.be/nPnx78bLehQ.

[13] *EV3dev python Testing Gyro sensor for accumulated error over time*. Youtube. May 3, 2017. URL: https://youtu.be/t0FoIMfTIoc.

[14] R. Faragher and Robert Harle. "An Analysis of the Accuracy of Bluetooth Low Energy for Indoor Positioning Applications." In: 2014. URL: https://www.semanticscholar.org/paper/An-Analysis-of-the-Accuracy-of-Bluetooth-Low-Faragher-Harle/bcbf261b0c98b563b842313d02990e386cad0d24.

[15] The Python Software Foundation. *Errors and Exceptions*. 2017. URL: https://docs.python.org/3.6/library/threading.html#threading.Thread (visited on 03/24/2017).

[16] The Python Software Foundation. *Errors and Exceptions*. 2017. URL: https://docs.python.org/3.6/tutorial/errors.html (visited on 03/10/2017).

[17] Lars Nikander Frandsen. "Collective Intelligence for EV3 Robots." MA thesis. Aarhus University, 2016.

[18] *Free for students: Professional developer tools from JetBrains*. 2017. URL: https://www.jetbrains.com/student/ (visited on 03/08/2017).

[19] *Git - Git Hooks*. 2017. URL: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks (visited on 03/08/2017).

[20] *Git –everything-is-local*. 2017. URL: https://git-scm.com/ (visited on 03/08/2017).

[21] GongYi. *python-ev3*. URL: https://github.com/topikachu/python-ev3 (visited on 03/06/2017).

[22] Ralph Hempel and David Lechner. *Programming Languages*. URL: http://www.ev3dev.org/docs/programming-languages/ (visited on 02/21/2017).

[23] Ralph Hempel et al. *API reference*. URL: http://python-ev3dev.readthedocs.io/en/stable/spec.html (visited on 03/06/2017).

[24] Ralph Hempel et al. *Button API*. URL: http://python-ev3dev.readthedocs.io/en/stable/other.html#button (visited on 03/09/2017).

[25] Ralph Hempel et al. *Miscellaneous Classes*. URL: http://python-ev3dev.readthedocs.io/en/stable/other.html (visited on 03/06/2017).

[26] Ralph Hempel et al. *Motor API*. URL: http://python-ev3dev.readthedocs.io/en/stable/motors.html (visited on 03/06/2017).

[27] Ralph Hempel et al. *Sensor API*. URL: http://python-ev3dev.readthedocs.io/en/stable/sensors.html (visited on 03/06/2017).

[28] Ralph Hempel et al. *ev3dev-lang-python - LegoPort*. URL: http://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/other.html#lego-port (visited on 03/10/2017).

[29] Ralph Hempel et al. *ev3dev-lang-python*. URL: https://github.com/rhempel/ev3dev-lang-python (visited on 03/06/2017).

[30] Ralph Hempel et al. *ev3dev-lang-python*. URL: http://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/motors.html#ev3dev.core.Motor.wait_while (visited on 05/01/2017).

[31] Ralph Hempel et al. *ev3dev-lang-python*. URL: http://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/sensors.html#gyro-sensor (visited on 04/21/2017).

[32] Ralph Hempel et al. *python-ev3dev Documentation*. URL: https://media.readthedocs.org/pdf/python-ev3dev/latest/python-ev3dev.pdf (visited on 03/06/2017).

[33] *How To Use SSH to Connect a Remote Server*. 2017. URL: https://www.digitalocean.com/community/tutorials/how-to-use-ssh-to-connect-to-a-remote-server-in-ubuntu (visited on 03/09/2017).

[34] Lars Jeppesen. *monoev3*. URL: https://github.com/Larsjep/monoev3 (visited on 02/21/2017).

[35] Lars Jeppesen and Anders Søborg. *MonoBrick*. URL: http://www.monobrick.dk/ (visited on 02/21/2017).

[36] Lars Jeppesen and Anders Søborg. *MonoBrick*. URL: http://www.monobrick.dk/software/ev3firmware/ (visited on 02/21/2017).

[37] Mikkel Baun Kjærgaard, Henrik Blunck, Torben Godsk, Thomas Toftkjær, Dan Lund Christensen, and Kaj Grønbæk. "Indoor Positioning Using GPS Revisited." In: *Pervasive Computing: 8th International Conference, Pervasive 2010, Helsinki, Finland, May 17-20, 2010. Proceedings*. Ed. by Patrik Floréen, Antonio Krüger, and Mirjana Spasojevic. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 38–56. ISBN: 978-3-642-12654-3. DOI: 10.1007/978-3-642-12654-3_3. URL: http://dx.doi.org/10.1007/978-3-642-12654-3_3.

[38] *LCD Screen*. 2017. URL: https://sites.google.com/site/ev3python/learn_ev3_python/screen (visited on 04/03/2017).

[39] *LEGO® MINDSTORMS® Support site*. Feb. 2017. URL: https://www.lego.com/da-dk/mindstorms/support (visited on 03/06/2017).

[40] LEGO. *LEGO MINDSTORMS EV3 Home User Guide*. The LEGO Group. 2015.

[41] LEGO. *LEGO MINDSTORMS EV3 Education User Guide*. The LEGO Group. 2016.

[42] LEGO. *EV3 Ultrasonic Sensor*. 2017. URL: https://shop.lego.com/en-CA/EV3-Ultrasonic-Sensor-45504 (visited on 03/15/2017).

[43] LeJOS. *LeJOS API documentation*. URL: http://www.lejos.org/ev3/docs/ (visited on 03/07/2017).

[44] LeJOS. *LeJOS Java for LEGO Mindstorm*. URL: http://www.lejos.org (visited on 03/07/2017).

[45] Charles Liu. *An Experimental Study on EV3 and NXT Ultrasonic Sensors*. 2014. URL: https://www.robofest.net/2014/Liu.pdf (visited on 03/16/2017).

[46] Fred G. Martin. *Robotic Explorations: A Hands-on Introduction to Engineering*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN: 0130895687.

[47] Bartosz Meglicki. *Mapping*. URL: http://www.ev3dev.org/projects/2016/08/07/Mapping/ (visited on 05/24/2017).

[48] Robin Murphy. *Introduction to AI robotics*. MIT press, 2000.

[49] *Netgear N150-WNA1000*. 2017. URL: https://www.netgear.com/home/products/networking/wifi-adapters/WNA1000M.aspx (visited on 03/08/2017).

[50] *Netgear N150-WNA1100*. 2017. URL: https://www.netgear.com/home/products/networking/wifi-adapters/WNA1100.aspx#tab-techspecs (visited on 03/10/2017).

[51] Michael Nygard. *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2007. ISBN: 0978739213.

[52] Michel Pelletier. *PEP 245 – Python Interface Syntax*. 2001. URL: https://www.python.org/dev/peps/pep-0245/ (visited on 03/06/2017).

[53] *Pillow Python Imaging Library*. Feb. 2017. URL: https://pillow.readthedocs.io/en/3.3.x/ (visited on 03/06/2017).

[54] HiTechnic Products. *NXT Compass Sensor (NMC1034)*. URL: http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NMC1034 (visited on 03/16/2017).

[55] *PyCharm*. 2017. URL: https://www.jetbrains.com/pycharm (visited on 03/08/2017).

[56] *Pygame*. 2017. URL: https://www.pygame.org (visited on 04/20/2017).

[57] David Lechner Ralph Hempel. *Connecting ev3dev to the internet and other devices*. URL: http://www.ev3dev.org/docs/networking/ (visited on 03/06/2017).

[58] David Lechner Ralph Hempel. *Different programming languages on the ev3dev*. URL: http://www.ev3dev.org/docs/programming-languages/ (visited on 03/06/2017).

[59] David Lechner Ralph Hempel. *Getting Started with ev3dev*. URL: http://www.ev3dev.org/docs/getting-started/ (visited on 03/06/2017).

[60] David Lechner Ralph Hempel. *Upgrading Ev3dev*. URL: http://www.ev3dev.org/docs/tutorials/upgrading-ev3dev/ (visited on 03/10/2017).

[61] David Lechner Ralph Hempel. *ev3dev is your EV3 re-imagined*. URL: http://www.ev3dev.org/ (visited on 03/06/2017).

[62] *Remote Python Call*. Feb. 2017. URL: http://rpyc.readthedocs.io/en/latest/index.html# (visited on 03/06/2017).

[63] SimpleTest. *SimpleTest - Mock objects documentation*. 2017. URL: http://www.simpletest.org/en/mock_objects_documentation.html (visited on 03/30/2017).

[64] Xander Soldaat. *Comparing the NXT and EV3 bricks*. Jan. 8, 2013. URL: http://botbench.com/blog/2013/01/08/comparing-the-nxt-and-ev3-bricks/ (visited on 03/07/2017).

[65] Russell Stuart. *Conspy - Remote control of Linux virtual consoles*. 2014. URL: http://conspy.sourceforge.net/ (visited on 03/10/2017).

[66] Daniel Stutzbach and Reza Rejaie. "Understanding Churn in Peer-to-peer Networks." In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: ACM, 2006, pp. 189–202. ISBN: 1-59593-561-4. DOI: 10.1145/1177080.1177105. URL: http://doi.acm.org/10.1145/1177080.1177105.

[67] Simon Tatham et al. *PuTTY: a free SSH and Telnet client*. 2017. URL: http://www.chiark.greenend.org.uk/~sgtatham/putty/ (visited on 03/08/2017).

[68] *Test backward(), forward() og stop()*. Youtube. Feb. 24, 2017. URL: https://youtu.be/ifyXeJEGzNc.

[69] *Test of subsumption, with Arbitrator and Behavior*. Youtube. Mar. 6, 2017. URL: https://youtu.be/u4K__YVqWcc.

[70] *Test rotate(360)*. Youtube. Feb. 24, 2017. URL: https://youtu.be/OptE24O5Wdc.

[71] *Test rotate(720, False)*. Youtube. Feb. 27, 2017. URL: https://youtu.be/jONWSHaVvQE.

[72] *Test rotate(720, True)*. Youtube. Feb. 27, 2017. URL: https://youtu.be/c0tFMur2kGI.

[73] *Test rotate_left() rotate_right()*. Youtube. Feb. 27, 2017. URL: https://youtu.be/Ts1CB2zTOHs.

[74] *Test travel(500, False)*. Youtube. Feb. 27, 2017. URL: https://youtu.be/Qyp6ezxWZ-U.

[75] *Test travel(500, True)*. Youtube. Feb. 27, 2017. URL: https://youtu.be/c67khLcGiiM.

[76] *Test travel(500)*. Youtube. Feb. 24, 2017. URL: https://youtu.be/YPXbbuTQ9ao.

[77] *The leJOS Tutorial, Behavior Programming*. URL: http://www.lejos.org/nxt/nxj/tutorial/Behaviors/BehaviorProgramming.htm (visited on 03/06/2017).

[78] S. Thrun, D. Fox, and W. Burgard. "Probabilistic mapping of an environment by a mobile robot." In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 2. May 1998, 1546–1551 vol.2. DOI: 10.1109/ROBOT.1998.677346.

[79] Sebastian Thrun. "Learning metric-topological maps for indoor mobile robot navigation." In: *Artificial Intelligence* 99.1 (1998), pp. 21 –71. ISSN: 0004-3702. DOI: http://dx.doi.org/10.1016/S0004-3702(97)00078-7. URL: http://www.sciencedirect.com/science/article/pii/S0004370297000787 (visited on 04/26/2017).

[80] Tomer Filiba. *Services and New Style RPyC*. 2016. URL: https://rpyc.readthedocs.io/en/latest/tutorial/tut3.html#tut3 (visited on 03/09/2017).

[81] *Using SSH/SCP on Mac OS X in the Terminal app*. 2017. URL: http://ged.msu.edu/angus/tutorials/using-ssh-scp-terminal-macosx.html (visited on 03/10/2017).

[82] Martin Vang. "Undersøgelse af to modeller til at implementere Behavior Based Control i leJOS på Lego Mindstorms EV3." MA thesis. Aarhus University, 2014.

[83] A. Ward, A. Jones, and A. Hopper. "A new location technique for the active office." In: *IEEE Personal Communications* 4.5 (Oct. 1997), pp. 42–47. ISSN: 1070-9916. DOI: 10.1109/98.626982.

[84] *WikiDevi - user database for computer hardware*. Feb. 2017. URL: https://wikidevi.com/wiki/Main_Page (visited on 03/06/2017).

[85] Wikipedia. *Ground truth — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Ground_truth&oldid=747505862 (visited on 03/15/2017).

[86] Wikipedia. *Monte Carlo localization — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Monte_Carlo_localization&oldid=742503395 (visited on 03/15/2017).

[87] Wikipedia. *Robotic paradigm — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Robotic_paradigm&oldid=755582544 (visited on 03/16/2017).

[88] Wikipedia. *SSH File Transfer Protocol — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=SSH_File_Transfer_Protocol&oldid=748520958 (visited on 03/09/2017).

[89] Wikipedia. *Shebang (Unix) — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Shebang_(Unix)&oldid=756002426 (visited on 03/09/2017).

[90] Wikipedia. *Time of arrival — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Time_of_arrival&oldid=716375708 (visited on 03/15/2017).

[91] Wikipedia. *Trilateration — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Trilateration&oldid=747450095 (visited on 03/14/2017).

[92] Wikipedia. *Bluetooth low energy — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Bluetooth_low_energy&oldid=769864030 (visited on 03/14/2017).

[93] Wikipedia. *Client–server model — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=769944747 (visited on 03/21/2017).

[94] Wikipedia. *Dead reckoning — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Dead_reckoning&oldid=766147652 (visited on 03/15/2017).

[95] Wikipedia. *Differential wheeled robot — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Differential_wheeled_robot&oldid=764130342 (visited on 03/23/2017).

[96] Wikipedia. *Global Positioning System — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Global_Positioning_System&oldid=769612343 (visited on 03/14/2017).

[97] Wikipedia. *Interface (Java) — Wikipedia, The Free Encyclopedia*. 2017. URL: http://en.wikipedia.org/w/index.php?title=Interface%20(Java)&oldid=759850321 (visited on 03/06/2017).

[98] Wikipedia. *Internet of things — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=769078517 (visited on 03/06/2017).

[99] Wikipedia. *Internet protocol suite — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Internet_protocol_suite&oldid=764004256 (visited on 03/06/2017).

[100] Wikipedia. *Lego Mindstorms — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Lego_Mindstorms&oldid=763043494 (visited on 03/07/2017).

[101] Wikipedia. *Lego Mindstorms EV3 — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Lego_Mindstorms_EV3&oldid=758489503 (visited on 03/07/2017).

[102] Wikipedia. *Network topology — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Network_topology&oldid=770944716 (visited on 03/22/2017).

[103] Wikipedia. *PID controller — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=PID_controller&oldid=769341738 (visited on 03/10/2017).

[104] Wikipedia. *Peer-to-peer — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=766873346 (visited on 03/21/2017).

[105] Wikipedia. *Pulse-width modulation — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Pulse-width_modulation&oldid=769088114 (visited on 03/09/2017).

[106] Wikipedia. *Radio propagation — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Radio_propagation&oldid=769891551 (visited on 03/14/2017).

[107] Wikipedia. *Received signal strength indication — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Received_signal_strength_indication&oldid=767913001 (visited on 03/14/2017).

[108] Wikipedia. *Secure Digital cards — Wikipedia, The Free Encyclopedia*. Mar. 2017. URL: https://en.wikipedia.org/wiki/Secure_Digital (visited on 03/06/2017).

[109] Wikipedia. *Secure Shell — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Secure_Shell&oldid=764662849 (visited on 03/08/2017).

[110] Wikipedia. *Secure copy — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Secure_copy&oldid=763460528 (visited on 03/08/2017).

[111] Wikipedia. *Sensor fusion — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Sensor_fusion&oldid=776963158 (visited on 05/25/2017).

[112] Wikipedia. *Simulation — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Simulation (visited on 03/29/2017).

[113] Wikipedia. *Single point of failure — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Single_point_of_failure&oldid=764359341 (visited on 03/21/2017).

[114] Wikipedia. *Standard deviation — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Standard_deviation&oldid=775338963 (visited on 04/20/2017).

[115] Wikipedia. *Subsumption architecture — Wikipedia, The Free Encyclopedia*. 2017. URL: http://en.wikipedia.org/w/index.php?title=Subsumption%20architecture&oldid=755582507 (visited on 03/06/2017).

[116] Wikipedia. *Thread (computing) — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Thread_(computing)&oldid=769356190 (visited on 03/24/2017).

[117] Wikipedia. *Triangulation — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Triangulation&oldid=764708080 (visited on 03/14/2017).

[118] Wikipedia. *Variance — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Variance&oldid=773810269 (visited on 04/21/2017).

[119] Wikipedia. *Wi-Fi positioning system — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/w/index.php?title=Wi-Fi_positioning_system&oldid=765855448 (visited on 03/14/2017).

[120] Wikipedia. *Write once, run anywhere — Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Write_once,_run_anywhere (visited on 03/29/2017).

[121] *WinSCP - Free SFTP, SCP and FTP client for Windows*. 2017. URL: https://winscp.net (visited on 03/08/2017).

[122] *Working with ev3dev remotely using RPyC*. Feb. 2017. URL: http://python-ev3dev.readthedocs.io/en/stable/rpyc.html (visited on 03/06/2017).

[123] ev3dev. *Connecting to the Internet via Bluetooth*. URL: http://www.ev3dev.org/docs/tutorials/connecting-to-the-internet-via-bluetooth/ (visited on 03/06/2017).

[124] ev3dev. *Connecting to the Internet via USB*. URL: http://www.ev3dev.org/docs/tutorials/connecting-to-the-internet-via-usb/ (visited on 03/06/2017).

[125] ev3dev. *Documentation contributors wanted*. URL: https://github.com/ev3dev/ev3dev/issues/287 (visited on 03/06/2017).

[126] ev3dev. *Downloads - Bootable SD card image files*. URL: http://www.ev3dev.org/download/ (visited on 03/06/2017).

[127] ev3dev. *Help building brickman*. URL: https://github.com/ev3dev/ev3dev/issues/232#issuecomment-69801370 (visited on 03/06/2017).

[128] ev3dev. *Using Bluetooth Tethering*. URL: http://www.ev3dev.org/docs/tutorials/using-bluetooth-tethering/ (visited on 03/06/2017).

[129] ev3dev. *Using USB Tethering*. URL: http://www.ev3dev.org/docs/tutorials/using-usb-tethering/ (visited on 03/06/2017).

[130] ev3dev. *ev3dev docs - HiTechnic NXT Compass Sensor*. URL: http://docs.ev3dev.org/projects/lego-linux-drivers/en/ev3dev-jessie/sensor_data.html#hitechnic-nxt-compass-sensor (visited on 03/16/2017).

[131] ev3dev. *ev3dev docs - LEGO EV3 Color Sensor*. URL: http://docs.ev3dev.org/projects/lego-linux-drivers/en/ev3dev-jessie/sensor_data.html#lego-ev3-color-sensor (visited on 03/16/2017).

[132] ev3dev. *ev3dev docs - Sensors / Input Devices*. URL: http://docs.ev3dev.org/projects/lego-linux-drivers/en/ev3dev-jessie/sensors.html (visited on 03/16/2017).

[133] ev3dev. *ev3dev kernel version 18*. URL: https://github.com/ev3dev/ev3-kernel/releases/tag/v4.4.44-18-ev3dev-ev3_1 (visited on 03/06/2017).

[134] *kernel 18-ev3dev breaks EV3 buttons*. URL: https://github.com/rhempel/ev3dev-lang-python/issues/286 (visited on 03/06/2017).

[135] *pip documentation*. 2017. URL: https://pip.pypa.io/en/stable/# (visited on 03/08/2017).

[136] *pip on wikipedia*. 2017. URL: https://en.wikipedia.org/wiki/Pip_(package_manager) (visited on 03/09/2017).

[137] ev3dev-lang python. *ev3dev-lang-python version 0.8.1*. URL: https://github.com/rhempel/ev3dev-lang-python/releases/tag/0.8.1 (visited on 03/06/2017).

[138] ev3dev team. *Brick Manager for ev3dev*. 2017. URL: https://github.com/ev3dev/brickman (visited on 03/06/2017).