# Contents

# Home

This wiki contains usefull tutorials to set up the Raspberry for the IstiABot project

# Setting up the Raspberry Pi 3 Lubuntu OS

## Lubuntu

For the next steps of this tutorial, we will use a Lubuntu distribution for Rasperry Pi 3. The img file can be downloaded at the following link : https://ubuntu-pi-flavour-maker.org/download/ The version of the OS (Operating System) while writing this document is Lubuntu 16.04.2. After downloading the image file, install it on a SD card using the technics of your choice.

## First start

When running the OS for the first time, a system configuration wizard will show up. Here are the parameters chosen:

```
Welcome > English
Wireless > I do not want to connect to a wi-fi network right now
Where are you? > Paris
Keyboard layout > French > alternative
Who are you? > name > IBOT
    computer's name > ibot-rasp
    username > ibot
    password > tobi
    log in automatically
```

Then the system should start its installation.

Once its done, you should see the Lubuntu desktop : the OS is installed for your raspberry.

## Using AZERTY keyboard

It can happen (all the time?) that even with the correct previous configuration you find yourself with a QWERTY keyboard configuration when the raspberry restarts...

### Open Panel Preferences

To open panel preferences just do a right click on the bottom Lubuntu bar and select “Add/Remove Panel Items” in the contextual menu.

### Add the Keyboard handler

In the “Panel Preferences” menu, choose the “Panel Applets” tab and select the “+Add” button. Then select the “Keyboard Layout Handler” in the “Add plugin to panel“ window. Once done, you should see an American flag on the right bottom of your screen. It means that you keyboard as an us configuration (qwerty). To change it, do a right click on the flag, and select the “Keyboard

Layout Handler Settings" item in the contextual menu. In the "keyboard layout handler" window, unselect the "keep system layouts" item. This should enable the left part of the window. Once enabled, click over the "+Add" button and select French as a new possible keyboard layouts. Now you will be able to change the keyboard configuration. If you do not want the qwerty version (and so the keyboard will be azerty even when rebooting the pi) you can remove the us layout by selecting it and clicking on the "-remove" button. You can now close the window and enjoy the azerty keyboard.

## Connect to eduroam WiFi network

To see all the available wifi network, you can do a left click on the "Indicator applet" (wifi icon on the bottom right). If the eduroam network does not appears in the list, make shure that there is an eduroam network nearby and restart the raspberry.

Once trying to connect to the eduroam network, the "WiFi Network Authentication Required" window should appear. On the "Authentication" item chose "LEAP". The window should redraw and now you should have to text fields : Username and Password. Just enter your university data (as when connecting to your ent) and this should do the trick, now the raspberry should be connected to eduroam.

## Open a terminal

In the later, we will use the terminal to execute most of our commands. To open a LXTerminal window:

```
Lubuntu button > System Tools > LXTerminal
```

You should have a window with the following:

```
ibot@ibot-rasp:~$
```

In the later, to ease the writing, this line will be notified as

```
$
```

### Open a terminal in a directory

When a directory is opened with the File Manager PCManFM (with the graphical user interface), pressing the F4 button will open a Terminal in this directory.

### Adding a terminal to the launch bar

It can be handful to have a terminal shortcut at the left bottom of the screen (with the File Manager, Web Browser shortcuts). To do that, start the Panel Preferences window (right click and add/remove panels items) and select the "Application Launch Bar" in the "Panel Applets/Currently loaded plugins" list. Then click over the "Preferences" button. A new "Application Launch

Bar" window should appear. In the "Installed Applications" select "System Tools/LXTerminal" and click on the "+Add" button. Now the green square icon of the Terminal should appear on the bottom left of your screen.

## Activate the SSH service

To be able to remotely be connected to the Raspberry it is needed to enable the ssh service, which is not done by default. To activate the ssh service when the raspberry start, use this command

```
$ sudo update-rc.d ssh enable && invoke-rc.d ssh start
```

and reboot the raspberry

```
$ sudo reboot
```

To test is the ssh service is started, try to connect to the raspberry from the raspberry (useful just for the test, useless otherwise…) with the command:

```
$ ssh ibot@localhost
```

you should have a message asking if you are sure, say yes, and then enter the password (still tobi), and congratulation, you are connected to the raspberry pi…

The good news is that you can do the same thing with an other computer, by changing "localhost" by the raspberry IP address in the command.

### Issues while trying to connect from an other computer

If you are sure that the ssh service is enabled on the raspberry (previous commands) but you can not remotely connect to it, it may be because the computer and the raspberry are not on the same network.

To test if you are on the same network as the raspberry, try to ping the raspberry with the command

```
$ ping 192.168.0.2
```

with 192.168.0.2 the IP of the raspberry (change with your raspberry IP of course…). This should display several messages like

```
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=0.487 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=0.250 ms
...
```

otherwise, you need to connect the computer and the raspberry over the same network…

### Finding the Raspberry IP address

To know the raspberry IP address, use the command

```
$ ifconfig
```

This command displays the network configuration of your raspberry.

## Connecting the raspberry to an old screen

It can happened that the Raspberry screen will not show up while connecting the Raspberry to an old screen (with VGA-HDMI converter for instance). To avoid that and have your Raspberry desktop displayed over most of the screens, you can change the hdmi settings by editing the /boot/config.txt file. To do that, open the file in editing mode with nano:

```
$ sudo nano /boot/config.txt
```

and at the end of the file, add the following:

```
disable_overscan=0
hdmi_force_hotplug=1
hdmi_group=2
hdmi_drive=2
overscan_left=20
overscan_right=20
overscan_top=20
overscan_bottom=20
```

use Ctrl+X > Y > Enter, to exit and save the modifications.

Note that if you want to modify this file on an other computer (using the sd card), you can find it on the root of the boot partition (and not in the boot folder).

## Using gedit text editor

### Install gedit

When programming using a text editor, I find gedit more friendly than leafdap (the default text editor). To install gedit, enter the command

```
$ sudo apt-get update
$ sudo apt-get install gedit
```

Gedit will use syntactical coloration base on the opened file extension.

### Some useful custom parameters

#### Display the line numbers

To activate the line numbers in gedit go to

```
Edit > Preferences > Display line numbers
```

#### Insert spaces instead of tabulation

I often rather like to add spaces instead of tabulation when programming. To do that with gedit, go to

```
Edit > Preferences > Editor > Tab width : 4
Edit > Preferences > Editor > Insert spaces instead of tabs
```

#### Highlight the current line

To highlight the current line (the line with the cursor), go to

```
Edit > Preferences > Highlight current line
```

#### Display white spacesheader-includes:

When programming I also like to display the white spaces (see tabulations, spaces…). To do that with gedit, you need to install gedit plugins by doing

```
$ sudo apt-get update
$ sudo apt-get install gedit-plugins
```

Once the plugins are installed, in gedit, do:

```
Edit > Preferences > Plugins > Draw Spaces
```

Then you should see a gray point for spaces and arrows for tabulations.

## Using the Raspberry Pi3 as a WiFi Access Point

The following steps are from

https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd/

This can be useful if you want to connect to your raspberry without any WiFi router.

### Packages installation

First you need to install hostpad (package that allows you to use the built in WiFi as an access point), and dnsmasq (a combined DHCP and DNS server that's very easy to configure). To install those packages:

```
$ sudo apt-get update
$ sudo apt-get install dnsmasq hostapd
```

You may need to reboot

```
$ sudo reboot
```

### Configurations

Here are the contents of the configuration files you need to modify in order to make it works. Do not forget to reboot the raspberry pi after that.

### /etc/network/interfaces

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

# The loopback network interface
auto lo
iface lo inet loopback

# for the CAN configuration, not needed for only the wifi hotspot
auto can0
iface can0 inet manual
    pre-up /sbin/ip link set can0 type can bitrate 500000
    up /sbin/ifconfig can0 up
    down /sbin/ifconfig can0 down

# for the wifi hotspot
allow-hotplug wlan0
iface wlan0 inet static
    address 192.168.49.1
    netmask 255.255.0.0
    network 192.168.0.0
    broadcast 192.168.255.255
```

### /etc/hostapd/hostapd.conf

Note that this file may not exist, you may need to create it.

```
# This is the name of the WiFi ibnterface to use
interface=wlan0

# use the nl80211 driver with the brcmfmac driver
driver=nl80211

# This is the name of the network - CHANGE THE VALUE HERE!
ssid=IBOT_BOHORT

# use the 2.4GHz band
hw_mode=g

# Use channel 1
channel=6

# accept all MAC addresses
macaddr_acl=0

# Use wpa authentification
auth_algs=1

# use wpa2
wpa=2

# use a pre-shared key
wpa_key_mgmt=WPA-PSK

# the network passphrase
wpa_passphrase=WIFI_TOBI

# use AES, instead of TKIP
rsn_pairwise=CCMP
wpa_pairwise=CCMP
```

### /etc/init.d/hostapd

change the line

```
DAEMON_CONF=
```

to

```
DAEMON_CONF=/etc/hostapd/hostapd.conf
```

### /etc/default/hostapd

change the line

```
#DAEMON_CONF=""
```

to

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

### /etc/dnsmasq.conf

You may want to save the original file by doing

```
$ sudo cp /etc/dnsmasq.conf /etc/dnsmasq.conf.back
```

Then remove the content of dnsmasq.conf to just have

```
interface=wlan0
dhcp-range=192.168.0.2,192.168.0.10,255.255.0.0,12h
```

Then reboot the pi by doing

```
$ sudo reboot
```

## Some usefull tools

### To control the raspberry memory and computing health

You can install htop, wich is a programm that display the used RAM memory, processors and list the running processus:

```
$ sudo apt-get install htop
```

To use htop just do

```
$ htop
```

# Setting up the CAN Bus module

All modules of the IstiaBot communicate using a CAN network. Here are the steps to configure the CAN network of the Raspberry.

## Modify the /etc/interfaces file

First you need to configure the interfaces file. To do that, we will use the nano terminal text editor

```
$ sudo nano /etc/network/interfaces
```

the password is “tobi”, and you should have this screen:

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

# The loopback network interface
auto lo
iface lo inet loopback
```

modify this file to have

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

# The loopback network interface
auto lo
iface lo inet loopback

auto can0
iface can0 inet manual
    pre-up /sbin/ip link set can0 type can bitrate 500000
    up /sbin/ifconfig can0 up
    down /sbin/ifconfig can0 down
```

to exit and save the modification use: the modification, use

```
Ctrl + X > Y > enter
```

You can use the command

```
$ cat /etc/network/interfaces
```

to check if the modification has been done (the file content should be displayed).

## Modify the /boot/config.txt file

To edit the config.txt file, use the command:

```
$ sudo nano /boot/config.txt
```

I should display a “big” file containing the settings of your raspberry. Go to the end of the file and add:

```
# Additional overlays and parameters are documented /boot/overlays/README

dtparam=spi=on
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
dtoverlay=spi1-1cs
```

Ctrl+X > Y > Enter, to exit and save the modification.

## Check the CAN interface

By doing the previous modification, the Raspberry CAN interface should be OK.

Once this is done, reboot the raspberry so the new configuration can be loaded:

```
$ sudo reboot
```

To check if the CAN interface is enable enter the command

```
$ ifconfig
```

and you should have a line can0 in the result, as

```
$ ifconfig
can0    Link encap:UNSPEC HWaddr ...
-00
    UP RUNNING NAORP ...
    RX packets: ...
    TX packets: ...…
```

if you do not have the can0 interface, check - the interfaces file - the config.txt file - your CAN bus network

## Reboot can0 interface

It can happen that the can interface fails. To reset the interface without rebooting the Pi, use the command

```
$ sudo ip link set can0 down && sudo ip link set can0 up
```

## Install CAN tools

To test CAN communication you can install the can-utils software by doing:

```
$ sudo apt-get update
$ sudo apt-get install can-utils
```

This should install the software, otherwise, check your internet connection.

## Use can utils

Can utils software provide at least two useful commands: cansend and candump. You can use them to check if the CAN network is OK. Note: be sure to have a CAN network, you have to connect at least 2 CAN components together to have a CAN network. If you only have the raspberry and its CAN shield, it will not work!

### Candump

First you can use candump command to see what is happening over the CAN bus. Start a new Terminal and use the commands

```
$ candump can0
```

This should display nothing more. Do not close this terminal for the next command!

### Cansend

Open a new Terminal and use cansend to send a CAN message by doing

```
$ cansend can0 123#45
```

This does nothing to this terminal but if you check the terminal with the candump command (hopping that you did not closed it…) you should have:

```
$ candump can0
    can0    123 [1] 45
```

Which means that over the can0 interface (your CAN network), a message occurred with the identifier 123 (hexa value) and 1 byte of data with the value 45 (hexa).

To use cansend, here is the syntax: #{R|data}. Note that each value correspond to an hexa value. If I want to send a data message with the identifier 0xC0 and the data 0x785, I use the command:

```
$ cansend can0 0C0#0785
```

Note that the identifier should be three digits (for standard frame) or 8 digits for extended format message. The data should be an even number of digit between 0 and 16 digits. To send a remote request, you have to add the R flag after the #. For instance a request with no data and 0xAB for identifier can be send with the command

```
$ cansend can0 0AB#R
```

The candump command should then display

```
$ candump
can0    0AB [0] remote request
```

meaning that a CAN remote request message occurred with the identifier 0AB (hexa) and 0 data.

# Programming ATMEGA components with the Raspberry

## Installing avrdude

We want to write a program on the Raspberry, compile it and load it in the ATMEGA micro controller. Doing that is called cross-compilation: we want to compile a program that will not be executed in the host (computer where we do the compilation) but on a target device. To be able to cross compile for ATMEGA micro controller, we need to install the avrdude tool. Do it with the command

```
$ sudo apt-get update
$ sudo apt-get install avrdude avr-libc
```

This should install avrdude and the tools that will be used to compile and load the program.

## Using a makefile

It is possible to do the compilation, loading steps by hand each time you want to update a program on the micro controller, writing commands like

```
$ avrdude -c avrispmkii -p m32m1 -P usb -U flash::w:output
$ avr-g++ -c -W -Wall -Werror -mmcu=atmega32m1 -Os -I src/ -DF_CPU=16000000UL
    -std=c++1 src/pin.cpp -o src/pin.o
...
```

… But it is quite painful, and it is easier (and possible) to automate most of it. To do that, we will do a makefile. A makefile is a file named makefile that contains instruction for the Terminal and can be executed with the command make.

### My first makefile

Lets make a really simple makefile just to illustrate. Create a new folder named test1 on the Desktop, and inside this folder create a new file named makefile. Edit this file using gedit and write the following:

```
all:clean
	ifconfig > ifconfig.txt
clean:
	rm -f ifconfig.txt
```

Warning: before ifconfig and rm you need to have tabulations, not spaces!!!

Lets try this makefile by opening a Terminal in the test1 folder (F4) and doing the command make :

```
$ make
rm -f ifconfig.txt
ifconfig > ifconfig.txt
```

If everything went well you should now have a ifconfig.txt file next to the makefile file with the network configuration written inside. What happened??

By doing make, we said that we wanted to "execute" the "all" instructions. But before doing the "all" instruction, we need to do the "clean" instruction (rm… ). This command line remove the ifconfig.txt file if it exists. Once the "clean" instruction is done, we can do the "all" instruction (ifconfig > ifconfig.txt) that write the result of the ifconfig command into a text file named ifconfig.txt. It creates the file if it does not exist.

#### A generic makefile for the ATMEGA programming

The makefile we want to use here will be more complicated than the previous as the instructions to automate are more complicated.

## Organizing the files

For the following makefile to do the job the program files must be organized as follow:

```
PROJECT_FOLDER/                  (folder)
code/                            (folder)
    main.cpp                     (file)
    include/                     (folder)
    *.h                          (file)
    *.cpp                        (file)
    Makefile                     (file)
documentation/                   (folder)
    doxygen_configuration.txt    (file)
    logo.png                     (file)
    documentation.html           (file)
```

Basically, all the source code should be in a "code" named folder, and the Makefile file should also be in that folder. The main file of the project must be named main.cpp or main.c and the rest of the source code should be in a "include" folder next to it.

The documentation folder is needed for the documentation generation using oxygen software, detailed later.

## The Makefile file

The Makefile source code to compile the code can be find in some repositories of the Istia Mecatronique Club organization gitHub https://github.com/IstiaMecatroniqueClub . Note that this is for programming an ATMEGA32M1 micro controller using a AVR-ISP-MK2 programmer. If you do not have the same configuration you may have to change few things. Note also that for a Makefile, tabulations must be tabulations and not spaces!

```
# Parameters for compiling

TARGET = output

FOLDER_NAME = include

```

```
F_CPU = 16000000UL

SRC = $(wildcard $(FOLDER_NAME)/*.cpp *.cpp)
INC = -I $(FOLDER_NAME)/
OBJ = $(SRC:.cpp=.o)

# Default compiler and programmer
CC = avr-g++
AVRDUDE = avrdude
OBJCOPY=avr-objcopy
# Device name (compiler and programmer)
MCU = atmega32m1
AVRDUDE_MCU = m32m1
AVRDUDE_PROG = avrispmkii

# for the documentation
DOCDIR = ../documentation
DOCFILE = doxygen_configuration.txt

AVRDUDEFLAGS = -P usb
# Default compiler and linker flags
CFLAGS = -c -W -Wall -Werror -mmcu=$(MCU) -Os $(INC) -DF_CPU=$(F_CPU) -std=c++11
LDFLAGS =

all: hex upload clean

documentation: $(DOCDIR)/$(DOCFILE)
	rm -rf $(DOCDIR)/html
	doxygen $(DOCDIR)/doxygen_configuration.txt

hex: $(TARGET).hex

# Create object files
%.o : %.cpp
	$(CC) $(CFLAGS) $^ -o $@

$(TARGET).hex: $(OBJ)
	$(CC) $(LDFLAGS) -mmcu=$(MCU) $^ -o $@

clean:
	rm $(OBJ)
	rm $(TARGET).hex

# Upload hex file in the target
upload:
	$(AVRDUDE) -c $(AVRDUDE_PROG) -p $(AVRDUDE_MCU) $(AVRDUDEFLAGS) -U
	    flash:w:$(TARGET).hex
```

```

flash:
	$(AVRDUDE) -c $(AVRDUDE_PROG) -p $(AVRDUDE_MCU) $(AVRDUDEFLAGS) -U lfuse:w:0xEE:m

# .PHONY => force the update
.PHONY: clean all upload documentation
```

## Using the makefile

For the following, you should have downloaded a valid makefile in the gitHub repository. By using the command :

```
$ make
```

you do three things: you compile the program and generate an output.hex file for the micro controller, you load the output.hex file into the micro controller, and you remove all the files generated during the compilation, including the output.hex file.

If do not want to load the file but just generate it, you can do:

```
$ make hex
```

If the ATMEGA32M1 was never programmed before, you may need to flash it. This can be done with the command

```
$ make flash
```

Finally, if doxygen is installed you can use it to generate a documentation of your program with the commands

```
$ make documentation
```

All those commands do the job if the Makefile file is the one detailed above and if the files are well organized as presented before!

Not working with IstiaBot boards

If the programming method does not work well with the IstiaBot boards. First check if the compilation is OK with "make hex". If not, your program needs to be checked. If the output.hex file is well generated but can not be loaded: * Try to flash the fuse with "make flash" and do the loading step again * Check the wiring of the programmer. Be careful where is the ground located on the board! The board must be powered while being programmed (with the CAN connector for instance)

## Using doxygen to generate documentation

Doxygen is a software that can generate an html documentation based on the comments in your source codes. To install doxygen, use the commands

```
$ sudo apt-get update
$ sudo apt-get install doxygen
```

and also install graphviz to generate graph for your documentation

```
$ sudo apt-get install graphviz
```

To use doxygen with the previous Makefile, you need * to have the doxygen_configuration.txt file detailed in the appendix. In this file, you can change the PROJECT_NAME entry to match your project name * to have a logo.png image

Note that this will generate all the html document inside a html/ file. To ease the use of the documentation, you can create a documentation.html file with only the following content:

```
<meta http-equiv="REFRESH" content="0;URL=html/index.html">
```

To access the document you thus just have to open the documentation.html file with your favorite web explorer.

Be careful to follow the file organization presented above.

# Using Qt5 with the Raspberry

It possible to install and use Qt with the Raspberry. Which ease using TCP/IP socket in C++ programs for instance. Note that in the later I only use Qt and gedit, I do not install Qt creator as development environment.

## Installing Qt5

Actually we will install two softwares: qt5 and qmake. As we are not using qtcreator to develop, we will use directly a makefile to compile the code. But the makefile for a Qt project is quite complicated to write and maintain. That is where qmake is useful: based on your files, it will generate a *.pro file (the project file of you Qt program) and a makefile. To install those tools:

```
$ sudo apt-get update
$ sudo apt-get install qt5-default qt5-qmake
```

This can take few minutes.

## My first Qt program on the Raspberry

Assuming that Qt and Qmake are correctly installed. Create new folder on you desktop named testQt. In this folder create a new file named main.cpp with the following content:

```
#include <QDebug>

int main(void){
```

```
    qDebug() << "Hello Qt from the "Rasp;
    return 0;
}
```

This simple program should just display the message “Hello Qt from the Rasp” using the Qt qDebug() stream.

### Create the project file (*.pro)

Now that we have the program written, we want to generate a project file for our Qt project. We could write it by hand but as we have installed qmake, we are going to use it. Enter the following (the Terminal should be opened in the testQt folder!).

```
$ qmake -project
```

This command should create a testQt.pro file with the following content:

```
######################################################################
# Automatically generated by qmake (3.0) ven. mai. 19 11:16:01 2017
######################################################################

TEMPLATE = app
TARGET =  testQt
INCLUDEPATH += .

# Input
SOURCES += main.cpp
```

### Create the Makefile file

Now that we have the project file, we can use it to generate a Makefile file using qmake:

```
$ qmake testQt.pro
```

This should create a Makefile file with a lot of stuff inside that I will not re-write here. Now you can see why qmake is nice...

### Compiling the code

Now that we have the Makefile file, we can compile the source code, finally, by doing

```
$ make
```

This will generate a testQt binary file.

### Executing the program

The binary file can be executed by doing

```
$ ./testQt
Hello Qt from the Rasp
```

The expected message should be displayed. Congratulation, you have made a Qt based programm using the raspberry pi.

### Do you need all those steps every time?

Ok this could be quite annoying to do that all the time, but the good news is: once you have created all the files for your project and that you are only modifying the content, you do not need to generate the project file and the Makefile file every time. Just once, and then modify your source code and compile with the make command.

But be careful, if you add new files to your project, you need to generate new project and Makefile file…

## PiCam and Qt

To use the PiCam in a Qt program, two things need to be done: installing the PiCam library, and configuring the Qt project to handle the library.

### Installing the Picam library

I use the raspicam library available at https://sourceforge.net/projects/raspicam/files

The current version while writing this is the raspicam-0.1.6. Download the zip and extract it.

To generate the makefile, the library needs cmake to be installed. That is, install cmake with

```
$ sudo apt-get update
$ sudo apt-get install cmake
```

Once cmake is installed, use it to generate the Makefile

```
$ cmake CMakeList.txt
```

Note that this command has to be done in the raspicam folder. If this worked, you should now have a Makefile generated.

Then, to install the library do

```
$ make
```

and

```
$ sudo make install
```

If everything went well you should have a raspicam folder in /usr/local/include and raspicam libraries in /usr/local/lib.

It appens that sometimes the camera libraries are not detected while executing a program, if it is the case, the following command should solve the problem:

```
$ sudo ldconfig
```

### Enable the camera

Once the library is installed, before using the camera it is needed to enable the camera module. To do that, open the /boot/config.txt file, by doing for instance

```
$ sudo gedit /boot/config.txt
```

and find the camera settings to change the start_x=0 line as start_x=1

```
#################################################################
##  Camera Settings
#################################################################

## start_x
##     Set to "1" to enable the camera module.
##
##     Enabling the camera requires gpu_mem option to be specified with a value
##     of at least 128.
##
##     Default 0
##
start_x=1
```

### Test the camera

To quickly test the camera setting, you can do the command:

```
$ raspistill -o test.jpg
```

This will take a picture named test.jpg with the PiCam.

### Quick program to test the PiCam

Create a new folder named testPiCam. Inside this folder create a new file named main.cpp with the content:

```
#include <fstream> // for ofstream
#include <raspicam/raspicam.h>
#include <unistd.h> // for usleep
#include <QDebug>
```

```

int main ( void) {
    raspicam::RaspiCam Camera; //Camera object
    //Open camera
    qDebug()<<"OpeningPiCam and Qt";

    if ( !Camera.open()) {
        qDebug()<<"Error opening camera";
        return -1;
    }
    //wait a while until camera stabilizes
    qDebug()<<"Sleeping for 3 secs";
    usleep(3000000);
    //capture
    Camera.grab();
    //allocate memory
    unsigned char*data= new unsigned char[Camera.getImageTypeSize (
       raspicam::RASPICAM_FORMAT_RGB )];
    //extract the image in rgb format
    Camera.retrieve ( data,raspicam::RASPICAM_FORMAT_RGB );//get camera image
    //save
    std::ofstream outFile ( "raspicam_image.ppm",std::ios::binary );
    outFile<<"P6\n"<<Camera.getWidth() << " " <<Camera.getHeight() <<" 255\n";
    outFile.write ( ( char* ) data, Camera.getImageTypeSize (
       raspicam::RASPICAM_FORMAT_RGB ) );
    qDebug()<<"Image saved at raspicam_image.ppm";
    //free resrources
    delete data;
    return 0;
}
```

This code should generate a program that save a ppm image using the camera.

Then use qmake to generate the project file

```
$ qmake -project
```

Note that the generated project file is not linked to the raspicam library. If you use it as generated by qmake it will not work. You need to modify the qmake project file to have

```
TEMPLATE = app
TARGET = testPiCam
INCLUDEPATH += /usr/local/include/raspicam
LIBS += -L /usr/local/lib -lraspicam

# Input
SOURCES += main.cpp
```

Then, use qmake once again to make the Makefile

```
$ qmake  testPiCam.pro
```

Finally make the project with

```
$ make
```

and execute the program

```
$ ./testPiCam
```

This should display the following results

```
Opening …Camera
Sleeping for 3 secs
Image saved at raspicam_image.ppm
```

And should have created an image name raspicam_image.ppm.

# Using gitHub with the Raspberry

## Installing Git

Git can be easily installed with the command

```
$ sudo apt-get update
$ sudo apt-get install git
```

## Initializing the git folder

Create a directory

```
$ mkdir ~/myGitHub
```

Initialize the git repository by doing

```
$ cd ~/myGitHub
$ git init
```

Voilà, the git repository is initialized.

## Getting a repository from gitHub

To recover a repository from gitHub, go to your myGitHub folder

```
$ cd ~/myGitHub
```

and close the gitHub repository by doing

```
$ git clone https://github.com/IstiaMecatroniqueClub/US_Board.git
```

Note that this is to clone the US_Board repository, change the url with the repository you want!

Now you should have a US_Board folder into your myGitHub directory.

## Some useful commands

Note that the following commands have to be executed in the repository folder (in US_Board folder for instance) To add all the files to the repository

```
$ git add *
```

To get the current status of the repository (what has been commit, the last modifications)

```
$ git status
```

Commit the repository

```
$ git commit -a -m "Text for the "commit
```

To push the commit to gitHub

```
$ git push
```

To update your repository with the one on git hub

```
$ git pull
```

# Using ROS with the Raspberry

## Installing ROS

Before installing ROS, it is needed to explicit to apt-get where it can download the ROS programs. To do that, create and edit a new file named “ros-latest.list” in the “/etc/apt/sources.list.d/” directory (root privileges are needed):

```
$ sudo nano /etc/apt/sources.list.d/ros-latest.list
```

In this file, just write the following line:

```
deb http://packages.ros.org/ros/ubuntu xenial main
```

And close while saving with Ctrl+X - Y - Enter.

Note that while writing this tutorial the current version of Lubuntu is “xenial”. If you have an other version replace “xenial” by your name version. To identify your Ubuntu version:

```
$ lsb_release -sc
```

Note that all the previous steps can be done with only one command:

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

This is not finished yet, to be able to download from this new address you need to be authenticated. To set up your keys, use the following command:

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

The key should be imported correctly.

The you can update apt-get with this new source:

```
$ sudo apt-get update
```

Finally, you can download and install ROS. Here we install the desktop kinetic ROS version:

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

This can take a while, go grab a coffee... Note: do not choose the ros-kinetic-desktop-full version for the raspberry! We do not need it and it takes more time to install!

## Initialize rosdep

The first time you will run ROS (and only once), you should initialize the dependencies first. To do that, you can run the following commands:

```
$ sudo rosdep init
$ rosdep update
```

Now the dependencies are initialized, you can continue to the next step.

## Setting up the catkin workspace

Usually the ROS projects are in a catkin_ws folder (ws stands for workspace). To create it, do

```
$ mkdir ~/catkin_ws
```

in this "catkin_ws" directory you should have a "src" folder that will contain all your source code. Thus do

```
$ cd ~/catkin_ws
$ mkdir src
```

Now everything is set, you can use the catkin_make command to initialize your workspace. To use the catkin_make command, you have to specify where is this command. To do so, use the following command

```
$ source /opt/ros/kinetic/setup.bash
```

This should do nothing (on the terminal I mean), but now you can do the following

```
$ catkin_make -j2
```

This command will create files and folders. Now you should have a "build" and a "devel" folders next to your "src". Note that the source command should be done every time you open a new terminal. Note that to use the programs of your catkin workspace you should source your workspace and not the general ROS setup, by doing

```
$ cd ~/catkin_ws
$ source devel/setup.bash
```

#### Avoid the source command each time

The source command to add setup.bash needs to be done each time you open a new terminal... In order to automate this source command, you can edit the .bashrc file:

```
$ nano ~/.bashrc
```

by adding at the end the folowing lines:

```
# source for ROS
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

Once this is done, you will not have to do the source command ever again.

### RPLiDAR in ROS

#### Allowing ROS to access the LiDAR

Before installing or using the RPLiDAR ROS node, you should allow it to access the sensor when plugged into the raspberry. Most of the time the LiDAR is mapped to /dev/ttyUSB0. The idea it to allow ROS to access the /dev/ttyUSB0. To do that you can do

```
$ sudo chmod 666 /dev/ttyUSB0
```

But this solution is not really nice, since you will have to do it each time you will reboot the raspberry or replug the LiDAR. To make it more permanently, you can add a dev rule (device rule). To do so, create and edit a new file, named "49-rplidar.rules" in the folder "/etc/udev/rules.d/"

```
$ sudo nano /etc/udev/rules.d/49-rplidar.rules
```

In this file write the line

```
KERNEL"==ttyUSB"*, MODE""==0666
```

And save and close the file with Ctrl+X - Y - Enter. Then reboot the raspberry

```
$ sudo reboot
```

### Installing and using the RPLiDAR node

To use the RPLiDAR ROS node you first need to download the source from github (https://github.com/robopeak/rplidar_ros). Extract the downloaded “rplidar_ros-master.zip” archive and put the extracted folder into your “catkin_ws/src” folder. Rename the “rplidar_ros-master” folder by removing the “-master” that is not useful. You should now have a “rplidar_ros” folder into the “~/catkin_ws/src” folder.

You can then make the project by doing

```
$ catkin_make -j2
```

in the “catkin_ws” directory. The -j2 option is to avoid that the raspberry freeze because it uses all the four cores at their maximum.

Note that if you are using a new terminal, you have to do

```
$ source devel/setup.bash
```

first! Otherwise the catkin_make command will not be recognize.

Once the compilation is done, you can install the rplidar node with

```
$ catkin_make install
```

Congratulation, the node is installed, you can start using it. To do so, open a new terminal, and lets named it terminal 1 (1), *andanotherterminal*(2). On the first terminal, we will start the roscore node:

```
1$ cd catkin_ws
1$ source devel/setup.bash
1$ roscore
```

If everything went well you sould have a message like

```
started code service [/rosout]
```

Now on the second terminal, start the rplidar_ros node (without closing the terminal 1)

```
2$ cd catkin_ws
2$ source devel/setup.bash
2$ roslaunch rplidar_ros view_rplidar.launch
```

If rviz opens but the LiDAR is not displayed, make sure that the LiDAR is plugged correctly to the raspberry and that you gave the rights to ROS to access the ttyUSB0 device.

## Using Rviz in a remote computer

It can be useful to run rviz in a remote computer in order to have a visual representation of the robot’s LiDAR measurements, map…

To do so, you first need to have ROS installed on both systems. Lets name them IBOT for the robot and UI for the remote computer that will only use rviz.

Assuming that IBOT has 192.168.49.1 as its IP address, and UI has 192.168.0.2.

The idea is to run roscore only on IBOT and specify to UI the IP of the running roscore.

To do that, on IBOT use the commands

```
IBOT$ export ROS_MASTER_URI=http://192.168.49.1:11311
IBOT$ export ROS_IP=192.168.49.1
```

then run roscode

```
IBOT$ roscore
```

On UI, use the commands

```
UI$ export ROS_MASTER_URI=http://192.168.49.1:11311
UI$ export ROS_IP=192.168.0.2
```

and run rviz

```
UI$ rosrun rviz rviz
```

Note that each time you will open a new terminal you will have to do the export command! To check the state of the environment variables you can do

```
$ printenv
```

or

```
$ printenv | grep ROS
```

if you only want the ROS variables.

If you want the export to be permanent, you can edit the bashrc file that initialize the terminals.

```
$ sudo gedit ~/.bashrc
```

by adding the two exports at the end. For IBOT for instance:

```
export ROS_MASTER_URI=http://192.168.0.1:11311
export ROS_IP=192.168.0.2
```

## Kinect1, Raspberry Pi and ROS

### install freenect

In order to install the kinect1 drivers, use the commands

```
$ sudo apt-get update
$ sudo apt-get install freenect
```

Then you can plug the kinect to the RPI, and test it with the command

```
$ freenect-glview
```

This should display the color and depth images on the screen.

### install ros-freenect

To interface the kinect with ROS, you can install a freenect node by doing

```
$ sudo apt-get install ros-kinetic-freenect-stack
```

To launch the ROS node

```
$ roslaunch freenect_launch freenect.launch
```

# Demo setup for the IstiABot

## Using the previous part of this tutorial

You need to install and setup Lubuntu, the CAN bus, github, ROS, the wifi as hotspot (after downloading everything…). This has to be done according to the previous part of this tutorial.

## Specific downloads

### RosBridge

You need to install rosbridge for the distance communication between the robot and a PC or smartphone. To do that:

```
$ sudo apt-get install ros-kinetic-rosbridge-server
```

### gitHub Repositories

In the catkin_ws folder (ROS installation), you need to clone the following repositories from github: - robotpeak/rplidar_ros - IstiaMecatroniqueClub/ibot_can_interface - IstiaMecatroniqueClub/I-botSlam - IstiaMecatroniqueClub/ibot_launch

Then do not forget to compile and install the nodes:

```
catkin_make -j2
catkin_make install
```

### The Web interface

You need to install apache2 for the distance control and visualization over a webbrowser. To do that:

```
$ sudo apt-get install apache2
```

Then clone the repository IstiaMecatroniqueClub/ibot_html, and replace the /vat/www/html folder of the rapsberry by the one in the repository. Note that you can change the name of the robot in the index.hml file.

### ease the demo start with an executable file

At the home of the raspberry pi, create a start_demo.bash file :

```
$ cd
$ nano start_demo.bash
```

and add the following content :

```
#!/bin/bash

roslaunch ibot_launch IstiABot_demo.launch
```

To make the file executable, do

```
$ chmod +x start_demo.bash
```

## Starting the demo

Once everything is ready in the raspberry pi, you can start the robot.

A wifi network should appears named with IBOT_NAME. Connect yourself to the wifi (WIFI_TOBI as password), and use ssh to be connected to the robot (IP: 192.168.49.1).

Once connected with ssh, you can start the demo with the command:

```
$ ./start_demo.bash
```

This should launch ROS and all the needed node.

### Control the robot

Once ROS is started, open a webbrowser and go to the url : http://192.168.49.1

You should have a web page with the name of the robot, and a message saying that you are connected. If there is a message saying that you are not connected, this is because rosbridge is not started in the robot.

If you are connected, you should be able to move the robot using the keyboard arrows or the letters 'Z','S','Q','D'.

### Visualizing the map

To visualize the map, you can use a webbrowser and go to the url : http://192.168.49.1/map.html

You can also use rviz remotly (previously explained in this tutorial).