



DESIGN OF A PROGRAMMING DEVICE FOR CHILDREN

Josep Oriol Serrano Gavalda

Bachelor's thesis
May 2014
Mechanical and Production
Engineering

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Mechanical and Production Engineering

JOSEP ORIOL SERRANO GAVALDÀ:
Design of a programming device for children

Bachelor's thesis 53 pages, appendices 25 pages
May 2014

The purpose of this thesis is to develop a device to teach children (6-8 years-old) the logic of programming. It also involves the theory related with the design of a user interface for children. The design of the user interface is an important point of the thesis because children of the target ages have some limitations. There is also an event of two days that were searching technological devices for children and the device will be brought to test it. In the event, a survey was filled by the author of the thesis and the results of it would be explained and discussed.

The thesis is divided in five parts. First there is the description of the device developed. Second there is the user interface, where all theory behind is explained and how it has been developed. Third there is a brief description of the hardware used. Fourth there is the description of the device, basically the code used. Finally there is the testing part, with the results of the test.

The results suggest that the device works. Children like it and they are able to follow the logic. But as a first prototype there are some parts that should be improved in future versions.

Keywords: Raspberry Pi, children, user interface, Arduino, python, c#

CONTENTS

1	INTRODUCTION	6
1.1	Possible problems	6
1.1.1	User interface problems	7
1.1.2	Operations problems	7
1.2	What would be studied?	7
2	USER INTERFACE FOR CHILDREN	8
3	DEVELOPMENT	9
3.1	Raspberry PI	10
3.2	DFRduino RoMeo	10
3.3	Description of the robot	11
3.3.1	First trial	12
3.3.2	Second trial	14
3.4	Design of the UI	16
3.4.1	QR reader	16
3.4.2	Computer program	18
3.5	Computer program code	18
3.5.1	Select an image	18
3.5.2	Logic of the blocks	19
3.5.3	Draw picture	20
3.5.4	Send the information	21
3.6	Design of the robot	24
3.7	Main code	25
3.7.1	Check buttons function	26
3.7.2	Output logic function	27
3.7.3	State logic function	32
3.8	Blocks library	33
3.8.1	Movement blocks	33
3.8.2	Condition blocks	34
3.8.3	Action blocks	34
3.8.4	Search function	36
3.8.5	Execute function	36
3.9	LED library	38
3.10	DFRduino Romeo code	38
3.11	Wireless connection	40
3.12	Hardware architecture	41
4	TEST	44

4.1 Results of the survey	45
5 DISCUSSION	47
6 CONCLUSION	49
REFERENCES.....	52
APPENDICES	54
Appendix 1. UI code (PC program)	54
Appendix 2. Main code (Main.py)	62
Appendix 3. Blocks code (Blocks.py).....	69
Appendix 4. LED code (LED.py)	74
Appendix 5. DFRduino Romeo code	75
Appendix 6. Hostpad configuration file	77
Appendix 7. Survey done to children	78

GLOSSARY

TAMK	Tampere University of Applied Sciences
UI	User Interface
QR code	Quick Response Code
RPI	Raspberry Pi
PC	Personal Computer
XBMC	Xbox Media Centre
FSM	Finite State Machine
GPIO	General Purpose Input/Output

1 INTRODUCTION

Nowadays children in school learn a lot of things related with different subjects, but they don't learn the basis of programming. In a society where programming is more important every day, it is good that children understand the logic of programming. Programming is more important every day because almost all the objects used in a day have some electronics with a microprocessor or small computers.

A lot of examples can be found. For example the smartphones: the newest smartphones are better than the first computers, they have a lot of apps and every day more people are trying to make their own apps. Another example can be found in cars: nowadays cars have more electronic components than ever, with a microprocessor controlling them; the most expensive cars have computers inside that helps to make easier living their life. As last example there is the internet: Our life turn around it, it is almost impossible to think of living without internet; it helps us a lot every day and nowadays a lot of jobs are related with it.

As can be seen in the examples before, programming is essential in our lives. So if children start learning how it works, it will help them in the future to understand the world and who knows, maybe in their careers.

The background for the thesis subject is an event for children (9th and 10th of May). The event was focused to children between the ages of 6-8 years-old. The organizers of the event were searching devices to bring at the event and show to children the possibilities of technology and they want to show that technology is funny and also amazing.

TAMK was asked to provide the event with one, such device and that request ignited this thesis. The event would be a great opportunity to test it and to find out if it is possible to teach the programming logic to children.

1.1 Possible problems

As the target ages of the event were children with ages between 6-8 years-old, there were some limitations that would have to be kept in mind. These limitations would be related with the user interface and the kind of operations that they would be able to understand.

1.1.1 User interface problems

Children between these ages (6-8 years old) don't know how to write and read very well. This problem has to be solved using visual combinations or things. Another big trouble: learning programming can be quite boring. In the case of children, this is more important because they like to see the results quickly and play games. So has to be found a way to make it easy and simple to learn the logic of programming. One way to teach the logic of programming is using games. Children always want to play and they spend a lot of time playing, so why don't use this fact?

1.1.2 Operations problems

The logic of programming can be taught in several ways. This make a problem of what kind of operations can be taught and how. An example can be found in a Microsoft game, Microsoft Kodu, this helps children to program. In this game children have to program to design a game. That fact gives them the possibility to make their own game with their rules. Microsoft Kodu is used in some schools, children have the possibility to play with this game in their technology subject and they like it, also some students play with the game at home. This means that the way that it uses to teach is valid, useful and funny.

1.2 What would be studied?

This thesis would be focused in the study of the logic of programming for children. After study how the logic of programming can be applied to children; a device would be built to teach it. The device has to be designed trying to solve all the problems described before. This device will be bring to the event for children, 9th-10th of May, and test it there. In the event a survey would be filled to know if the device works, if it teaches the logic of programming and if children like it.

2 USER INTERFACE FOR CHILDREN

The UI is one of the most important parts of the device. As is discovered in a study of Elizabeth Boling and Sung-Hee Learners' (Peters 2013, 78), ideas about the meaning of instructional images agree with designers' opinion only half the time. So is very important design a good UI.

So, the UI, is very important because it is a device for children. As the target ages of the device are children in the ages between six and eight, limitations that six-year-old children have, have to be kept in mind.

The UI has to follow some points to make it easy, funny, interesting and simple to use. Simple to use means that has to be with basic things, not with a lot of extra functionalities. If the UI has a lot of extra functionalities, it could be difficult to use and ambiguous and that's not what was wanted. Also the software will become more complex and difficult to do. This does not mean that the UI has to be boring or uninteresting; it can be seductive and interesting. Though the items do not have to be loaded noisily.

As the device is for children, it has to be similar to other devices that they use. This point will help them to use it and could give them an approximate idea of how it works. Also is important to know how our brain thinks and works in order to create an easy way to understand UI and to increase the engagement. Our visual perception is the main sense that we use when we are in front of an UI, so is important make a comfortable and attractive UI to our eyes. Is known that the brain has more difficulties to detect colours than contrasts, and the ability to recognise colours is related with the way that colours are used (Johnson 2010, 53). If the visuals are simplified helps to learn and let them focus on the important details, three strategies were followed to simplify the visuals:

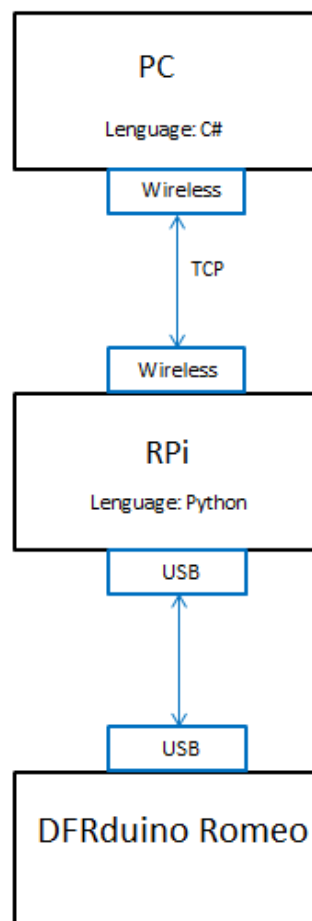
- Use icons
- Remove extra detail
- Clear the background

Colours often are used to differentiate variables. The once that are easier to differentiate are yellow, green, red, blue, black and white. Moreover, the saturation of the colour frequently is related with the quantity. These colours are used as background to help the identification of each kind of block.

3 DEVELOPMENT

The device is based on the Raspberry Pi (RPi). RPi has some limitation when some peripherals like motors, LDR sensors and any kind of analogical sensors are tried to be used, that make it difficult to use. To solve these limitations there are modules that you can add or you can use Arduino. The chosen option was to use a DFRduino Romeo which contains the hardware necessary to control motors and is Arduino compatible. This option was chosen because was an opportunity to learn more how to program using Arduino.

There are two programming languages used. C# is used in the UI and Python in RPi. To exchange the information between both parts, a TCP connection is used.



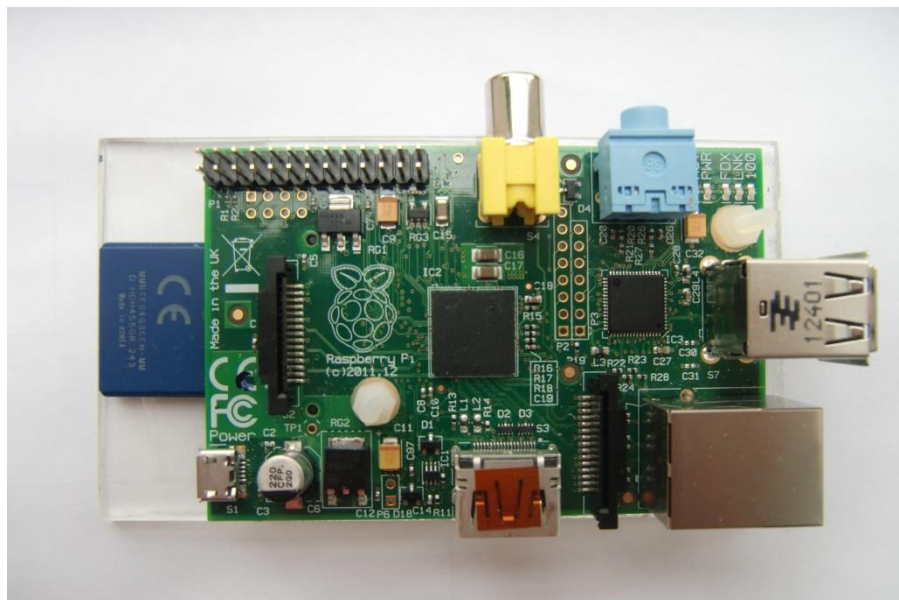
PICTURE 1. Relation between UI, Rpi and DFRduino Romeo

3.1 Raspberry Pi

Raspberry Pi is a low cost (25\$ or 35\$) personal computer. Windows OS is not used in RPi, but there are different OS that can be used. The most used is Raspbian (is based on Debian and is optimized for RPi).

RPi has an Ethernet connector, audio output, RCA video output, HDMI output, two USB sockets and one SD slot for the operating system

RPi can be programmed using almost all programming languages (C, C++, JAVA, Python, HTML....). It was designed to use Python.



PICTURE 2. Raspberry Pi

Nowadays RPi is very used in a lot of projects because it is a cheap computer and gives a lot of possibilities to do very different things.

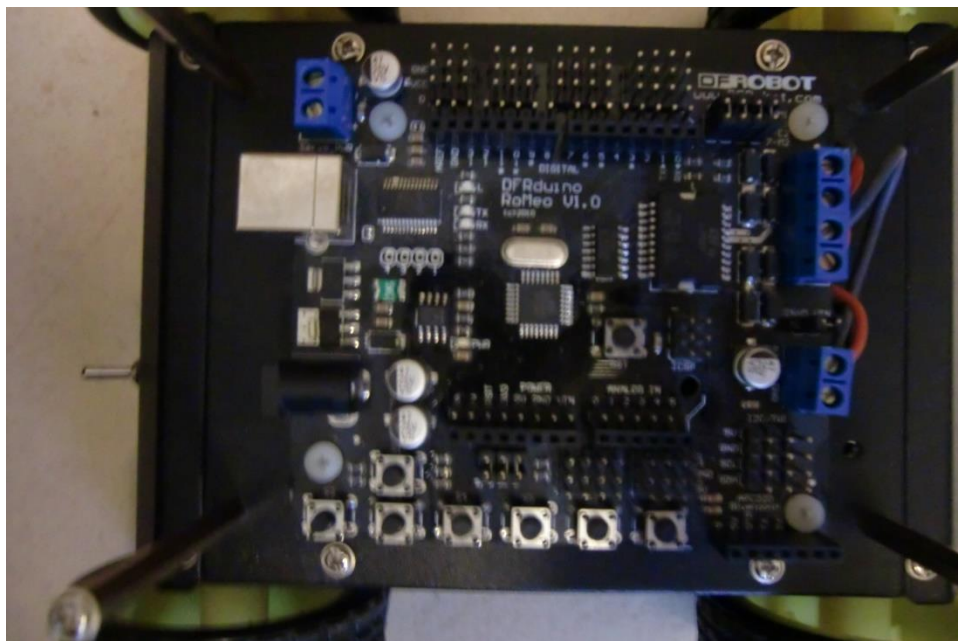
3.2 DFRduino RoMeo

DFRduino RoMeo is an Arduino compatible microcontroller designed for robotic applications. DFRduino Romeo can be expanded using the arduino shields. It is very similar to an Arduino Duemilanove 328, so you can program it using the Arduino IDE. A 2 way DC motor driver is integrated.

As can be seen in PICTURE 3, has a lot of components.

The specifications are (http://www.dfrobot.com/wiki/index.php/DFRduino_Romeo-All_in_one_Controller_%28SKU:DFR0004%29):

- Atmega 168/328
- 14 Channels Digital I/O
- 6 PWM Channels (Pin11,Pin10,Pin9,Pin6,Pin5,Pin3)
- 8 Channels 10-bit Analog I/O
- USB interface
- Auto sensing/switching power input
- ICSP header for direct program download
- Serial Interface TTL Level
- Support AREF
- Support Male and Female Pin Header
- Integrated sockets for APC220 RF Module and DF-Bluetooth Module
- Five I2C Interface Pin Sets
- Two way Motor Drive with 2A maximum current
- 7 key inputs
- DC Supply : USB Powered or External 7V~12V DC.
- DC Output : 5V /3.3V DC and External Power Output
- Dimension : 90x80mm



PICTURE 3. DFRduino Romeo

3.3 Description of the robot

Two different trials were intended to make the user interface (UI): Using blocks as UI and using the PC as UI.

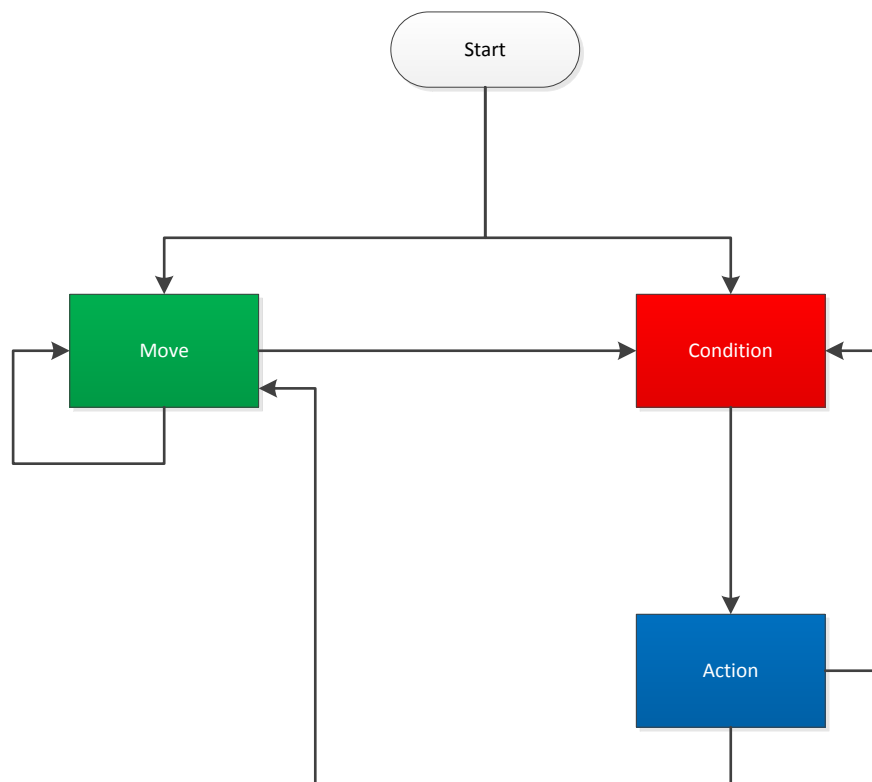
The first trial was the option chosen at the beginning. But there were some problems with the camera and was modified to the second option. Both trials have the same principle, only differ by the way that the UI is done.

3.3.1 First trial

The idea was to control the robot with blocks. To place the blocks we have to follow four rules (explained below). The blocks are selected using a program for a PC and then it sends the information to the robot.

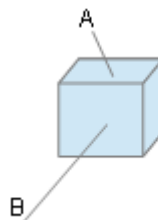
The rules that we have to follow when we select the blocks are related with the three kinds of blocks: green blocks are for movement, red blocks are for the conditions (when) and blue blocks define the actions (do). The rules are explained below and in a flow chart (PICTURE 4). Rules:

1. You have to start with a green or red block (movement or condition)
2. After a red block (condition) you only can select a blue block (action)
3. After a blue block (action) you have to select a green or red block (movement or condition).
4. After a green block (movement) you can select a green or red block (movement or condition)



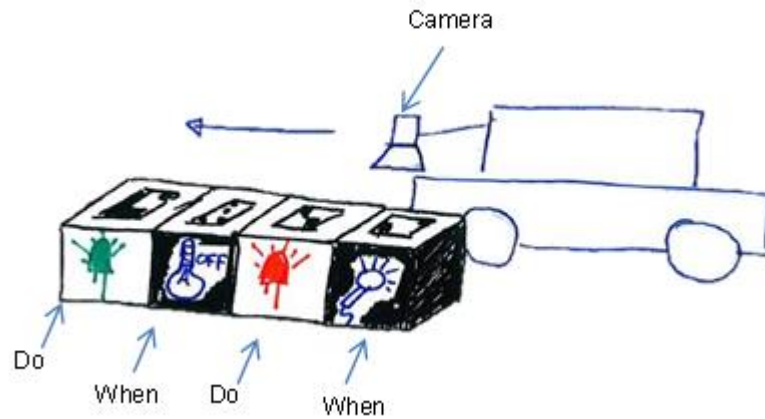
PICTURE 4. Flow chart of the blocks

The UI is the way that children interact with the device. In the first trial blocks were used to program the device (PICTURE 5).



PICTURE 5. Block design

As you can see in the image above, there are two sides of the block that are used (side A and B). These sides are used to hold the information of each block. Side A is used to hold a QR code that the device will read to identify each block. Side B has an image for the users to be able to recognize the block; this image has to follow the rules described before to make it easy to understand. The device will do like on the PICTURE 6 to read the blocks once they are placed.



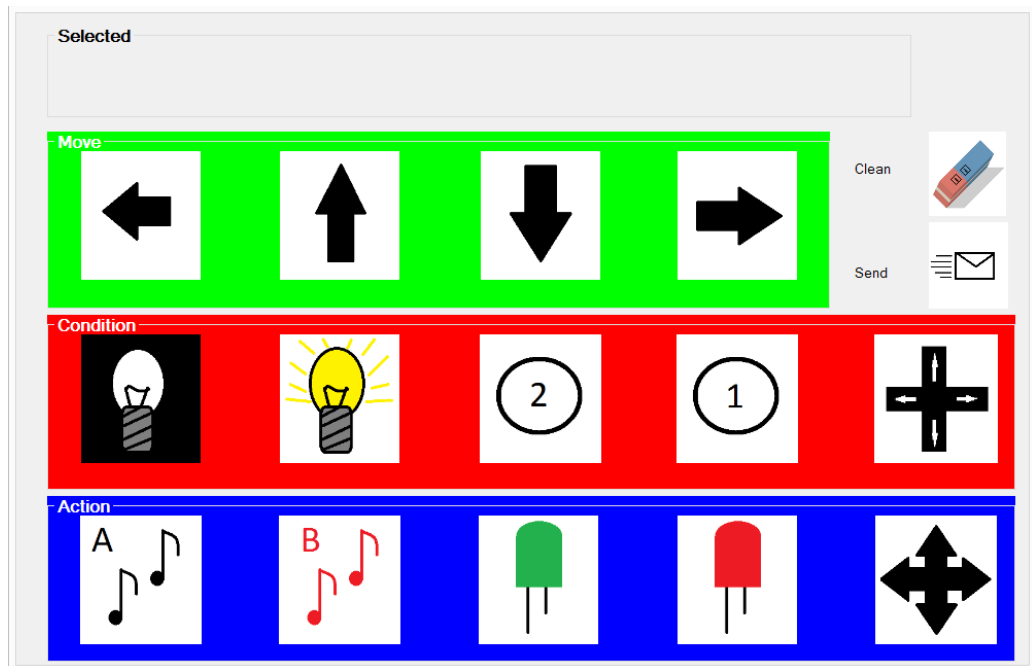
PICTURE 6. Blocks and Robot

This idea of the UI was discarded because there were some problems with the QR code reader. There were problems with the drivers needed to use the webcams and using the RPi camera there were problems focusing the image.

As the way that the children have to place the blocks is important, the new UI has to be quite similar and has to follow the same “philosophy”.

3.3.2 Second trial

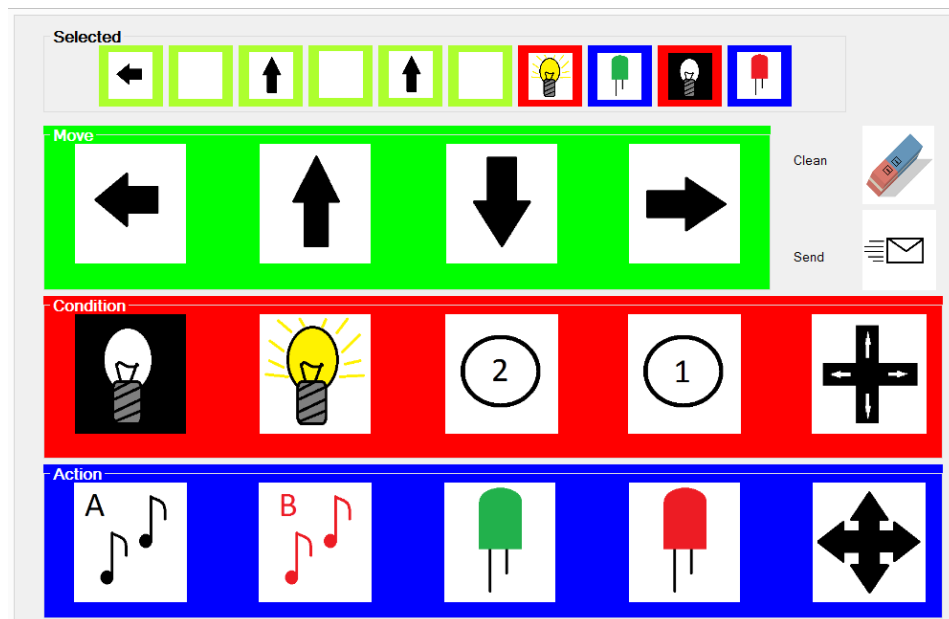
This second trial uses the same principle but is a PC program developed in C# (PICTURE 7).



PICTURE 7. UI without blocks selected

On this new UI children have to click on the images and then send the images selected to the robot. Now the images have the same function that side B of the blocks on the first approach, and the function of the side A is done with a tag that each image has and is sent using TCP to the robot.

The UI can be split on four sections: Selected, Move, Condition, and Action. The first section is “Selected”, here the images that the user has selected are placed smaller to know what you have selected (PICTURE 8). The next three sections (Move, Condition and Action) have the same functionality, hold the images. These three sections have different colours, these colours are the RGB that are primary colours and are easy to differentiate.



PICTURE 8. UI with blocks selected

3.4 Design of the UI

The first explanation is about design of the UI using QR codes. But there were some problems with the camera and finally it was designed using a computer program, this will be explained in the point 3.4.2 Computer program.

3.4.1 QR reader

The first idea of the UI was use a camera to read QR codes. The device was supposed to do that. As the code used in the device is written in python and the OS is based in Debian there are several ways to do that.

The first approach was using zbarcam. Zbarcam is a library that is used to decode bar codes and QR codes from a video device. It can be used in the command line with the next code:

```
zbarcam -raw -nodisplay /dev/video1
```

It uses the device /dev/video1 to obtain the images and print the output in the command line. As it has to be used in the Main program, it has to be called from the python file:

```
file = open('qr2.txt','w+')
```

```
p = subprocess.Popen(["zbarcam", "--raw", "--nodisplay", "/dev/video1"],
stdout=file)
```

The `subprocess.Popen()` function is used, the result is stored in the `qr2.txt` file, and later it is read.

The `subprocess.Popen()` function executes a child program in a new process. The filed `stdout` specifies the standard output of the executed program. This output value is redirected using pipe (output of a process feeds the input of another process) to the variable *file* (<https://docs.python.org/2/library/subprocess.html>).

First the code was tested in a laptop running Ubuntu and it worked correctly, but when the code was run in the RPi, there were some problems with the cameras used.

The error was:

```
libv4l2: error turning on stream: Broken pipe
ERROR: zbar processor in v4l2_start():
system error: starting video stream (VIDIOC_STREAMON): Broken pipe (32)
```

The problems with the webcams were related with the drivers, there are no drivers for the webcams that were used that allow using this library in the RPi. Also it was tested using the RPi camera but the problems with this camera were related with the focus. The RPi comes without a lens so is impossible to focus an object when it is near the lens, a solution is put a lens but it is difficult because the camera has to be disassembled.

The next approach was to use a Zbarimg (is another library) and the result was the same than the one using Zbarcam.

And the last approach using the device to read the QR codes was using QR tools (python library for code and decode QR codes). QR tools works with python so it was no need to use a pipe or a file to store the information of the codes. But the result was the same than the one with the other two libraries.

As conclusion was learned that it is not possible to use some webcams with these libraries, so you need to find a camera that work correctly or modify the source code of the drivers (but this option is difficult and is far away of the objective of the thesis). And the RPi camera has the problem with the focus that has to be solved using lens.

Finally, this way to read the blocks was discarded for all the problems described above.

3.4.2 Computer program

After thinking in different ways to send the information of the blocks to the device, two possible solutions were found: Create a webpage in the RPi using php to select the blocks or a program for a PC and then send the information using TCP. Of these two options the used was the second one (PC program). The computer program will be explained in the next point because is quite long.

3.5 Computer program code

The program is written using C# and the information is send to the device using TCP via a wireless connection. The next four sections will explain the main functionalities used in the program. These main functionalities are: Select an image, logic of the blocks, draw the image selected and send the information.

The last section, 3.5.4 Send the information, has more than one method because the information sent has to be correct and has to follow a specific format. This format will be explained also in the last section.

3.5.1 Select an image

The code related with the selected image action is located in the method *Select_Picture* (*Image image, Object tag, Boolean moveBlock*). It determines if is possible to select the image or not. This method is run when an image is clicked. Each image represents a block. The images used in the program are loaded from a local archive.

```
private void Select_Picture(Image image, Object tag, Boolean moveBlock)
{
    if (LogicBlocks(tag.ToString()))
    {
        Draw_picture(image, tag, moveBlock);
    }
}
```

3.5.2 Logic of the blocks

The *LogicBlocks(String tag)* method determines if the image that you have clicked follows the four rules explained in PICTURE 4. **Error! Reference source not found.** The method returns true if the image selected follows the rules and returns false if it doesn't follow the rules.

If the image selected follows the rules, it goes to the method *Draw_picture(Image image, Object tag, Boolean moveBlock)*. In this method the program draws the image in the correct place of the "Selected" section of the UI.

Here is the code of the method with the logic (*LogicBlocks(String tag)* method):

```
public bool LogicBlocks(String tag)
{
    /*
     * i=0 => Move block & initial value
     * i=1 => Condition block
     * i=2 => Action block
     */

    int i = -1;
    try
    {
        i=Convert.ToInt16(tag)/10000;
    }
    catch (Exception)
    {
        return false;
    }

    if (i == -1)
        return false;
    switch (_previousModeBlock)
    {
        case 0:
            if (i == 0)
            {
                _previousModeBlock = i;
                return true;
            }
            if (i == 1)
            {
                _previousModeBlock = i;
                return true;
            }
            return false;
        case 1:
            if (i == 2)
            {
                _previousModeBlock = i;
                return true;
            }
            return false;
    }
}
```

```

        case 2:
            if (i != 2)
            {
                _previousModeBlock = i;
                return true;
            }
            return false;
        default:
            return false;
    }
}

```

This method is simple to understand because follows the rules explained in the PICTURE 4. To be able to know which kind of block has been selected before, the method uses the global variable *_previousModeBlock*. This variable is an integer and assigns a number to each kind of block:

- 0: Move block and initial value
- 1: Condition block
- 2: Action block

3.5.3 Draw picture

An image is drawn in the “Selected” section of the UI by the *Draw_pictures (Image image, Object tag, Boolean moveBlock)* method.

One important point that this method does is know if there is still space in the “Selected” section to draw an image. To know if there is still space, it checks if the variable *Selected_images* (is a list that contains all the PictureBoxes of the “Selected” section) has any element without an image. Then it assigns the image and the tag to the element. If the image selected is of the type move block, draws a white image in the next element (image selected).

It also runs the method *panelColor(int i, int tag)* that paints the panel that is behind the image. This method is useful because help to know what kind of block you have selected.

Here is the code of the method *Draw_picture (Image image, Object tag, Boolean moveBlock)*:

```

private void Draw_picture(Image image, Object tag, Boolean moveBlock)
{
    for (int i = 0; i <10; i++)
    {

```

```

        if (Selected_images[i].Image == null)
        {
            Selected_images[i].Image = image;
            Selected_images[i].Tag = tag;
            Selected_images[i].SizeMode = PictureBoxSizeMode.Zoom;//To
resize the image
            try // Paints the panel that is down the image, so you
know the kind of image
            {
                panelColor(i, Convert.ToInt32(tag.ToString()));
            }
            catch (Exception)
            {
                Console.WriteLine("Error");
                throw;
            }
            if (moveBlock)//Next image is a white image
            {
                Draw_picture(white, white_tag, false);
            }
            break;
        }
    }
}

```

3.5.4 Send the information

The last step of the UI is to send the information to the device. To be able to do that, it uses a TCP connection where the RPi is the server and the program is the client. The program sends the information when the button send is clicked.

```

private void button1_Click(object sender, System.EventArgs e)
{
    if (CheckSelection())
    {
        Client();
    }
}

```

When you click the button, the method *CheckSelection()* checks that you have selected a correct number of images. What it basically does is check if you have ended the selection with a movement or action block and also counts the number of conditions that you want to do with the device (number of condition and movement blocks).

Code of the *CheckSelection()* method

```

public bool CheckSelection()
{
    int i;
    for (i = 1; i < 10; i++)
    {
        if (Selected_images[i].Image == null)
        {
            if (i%2 != 0)//Means that you have ended the selction with
a condition
                return false;
            else //You have ended the selection with an action or a
movement block
            {
                break;
            }
        }
    }
    _conditionNumber = i / 2;
    return true;
}

```

If all is okay it returns true and then it sends the information to the RPi using the TCP connection. To do that it uses the method *Client()* where it creates a TCP connection and sends the information. Here is the code of the *Client()* method:

```

public void Client()
{
    byte[] data;
    string input, stringData;
    TcpClient server;
    string selected;

    try
    {
        server = new TcpClient(IP, 6112);
    }
    catch (SocketException)
    {
        Console.WriteLine("Unable to connect to server");
        return;
    }
    NetworkStream ns = server.GetStream();
    selected = send_info();
    data = new byte[Encoding.ASCII.GetBytes(selected).Length];
    data = Encoding.ASCII.GetBytes(selected);
    ns.Write(data, 0, data.Length);
    Console.WriteLine("Disconnecting from server...");
    ns.Close();
    server.Close();
}

```

This method creates a socket to the server and uses the port 6112. The number of the port selected is not important if it is free (there is no other program or service that is using it). The address of the server is selected using a global variable IP (is a string with the address).

When the connection is done, the method needs to generate the string with all the information that will send. To do that it uses the method *send_info()*:

```
private string send_info()
{
    var str = _conditionNumber.ToString();
    for (int i = 0; i < _conditionNumber; i++)
    {
        str = str + "-" + Selected_images[(2*i)].Tag + "-" + Selected_images[(2*i)+1].Tag;
    }
    return str;
}
```

This method generates a string with all the information needed for the device. The string has the next format:

1. Number of conditions that you want to do with the device (number of condition and movement blocks).
2. Delimiter character: “-“
3. Tag of the first image
4. Delimiter character: “-“
5. Tag of the second image

And repeats the points 2 to 5 until have written all the conditions. Finally it returns the string to the method *Client()*.

The method *Client()* send this string to the server (RPi).

To clean the selected images there is the Clean icon. This icon makes equal to null all the images of the *Selected_images* list and change the colour of the panels to transparent. Also reloads the value of the *_previousModeBlocks* to 0, so the program will be able to run the *logicBlocks(String tag)* method without errors.

```
private void CleanImage_Click(object sender, EventArgs e)
{
    _previousModeBlock = 0;
    for (int i = 0; i < 10; i++)
    {
        Selected_images[i].Image = null;
        _selectedPanels[i].BackColor = Color.Transparent;
    }
}
```

3.6 Design of the robot

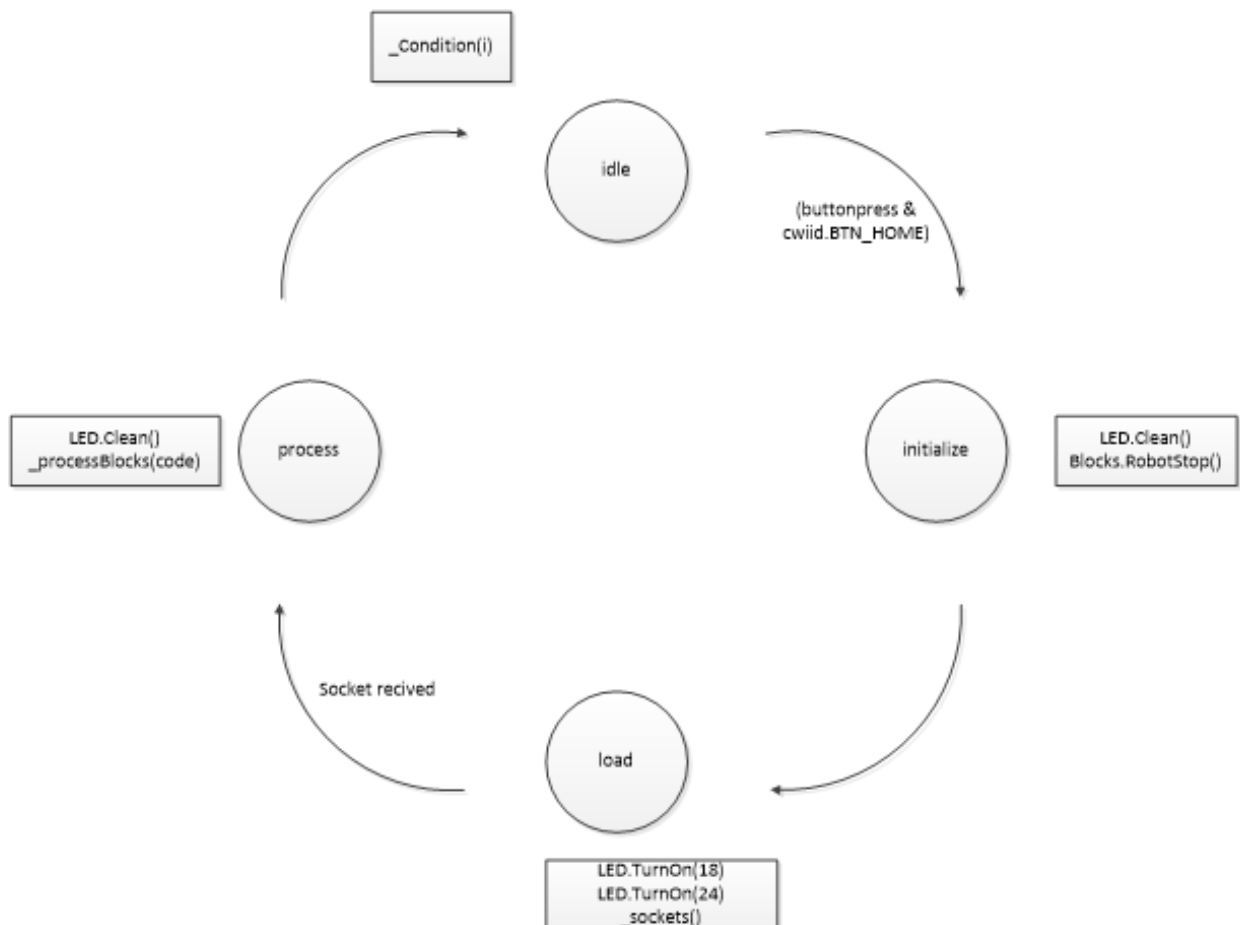
The code used in the robot follows the FSM programming style. The robot follows a state diagram that can be seen in the **PICTURE 9Error! Reference source not found..** There are four different states. The first state is idle. Here the program executes the code received from the PC program. The program stays in this state until the “Home” button of the Wii mote (remote control of Nintendo Wii console) is pressed, and then it switches to the clean state.

The initialize state is used to turn off the LEDs, stop the device in case that it moves and set to default values some variables.

After do this, it goes to the load state where it waits the socket connection. Also all the LEDs are turn on. This is done to know that is waiting a socket.

When the socket has arrived, it goes to the process state, on this state it processes the information received and makes it able to use; and turns off the LEDs.

Finally it goes to the idle state and the cycle starts again.

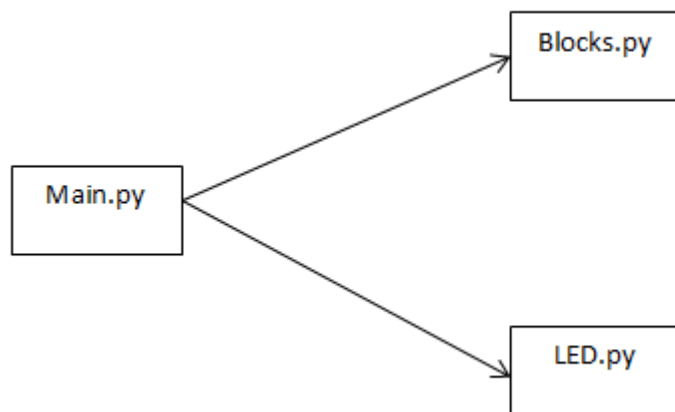


PICTURE 9. State Diagram

As there are a lot of code lines, is necessary to split the code in libraries. Splitting the code in libraries makes the code easy to understand, clear and more modular (is possible to reuse the code in other projects). The code is divided in three files:

1. Main.py (Main code)
2. Blocks.py (Blocks library)
3. LED.py (LED library)

The objective and the main functions of these files are explained in the next three points. The relation between the files can be seen in PICTURE 10.



PICTURE 10. Relation between files

3.7 Main code

This file can be found in the Appendix 2. It contains the core of the program, all the states and has the other two files as libraries. To import the other two files as libraries, these lines have to be added at the beginning of the code:

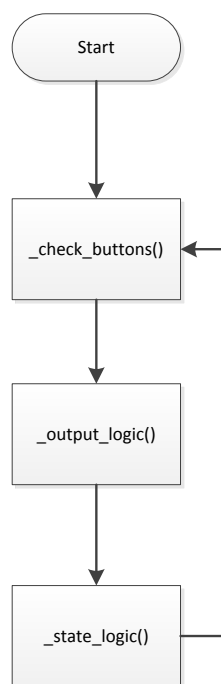
```
import Blocks #Libraries with the code  
import LED
```

Now the Main.py is able to run all the code that is inside these files.

The Main.py is the core, so has to have all the necessary code to follow the FSM programming style. The code that uses to follow this style is the one bellow.

```
if __name__ == "__main__":
    #FSM
    while(1):
        _check_buttons()
        _output_logic()
        _state_logic()
```

In the PICTURE 11 can be seen the flow chart of it.



PICTURE 11. Flow chart Main.py

3.7.1 Check buttons function

The first thing that the program does is go to the function *_check_buttons()*.

```
def _check_buttons():
    global wii,buttonpress
    buttons = wii.state['buttons']
    buttonpress=buttons
```

This function checks which Wii mote button are you pressing and stores the result in the global variable *buttonpress*.

3.7.2 Output logic function

After do this function the program goes to the function *output_logic()*. Here is located all the code related with the output of the device. It does all the processors that are associated to each state of the state diagram (PICTURE 9).

```
#Output logic of the FSM

def _output_logic():
    global state, _blocks, _ConditionFlags

    if state==idle:
        print "-----numberCondition-----"
        "+str(_numberCondition)

        #Run all the conditions
        if _numberCondition>0:
            for x in xrange(1,6):
                _Condition(x)
            if x==int(_numberCondition):
                #When has runned all the conditions stop
                print "----"
                BREAK!!!----"
                x=10
                break

        return

    if state==initialize:
        #Turn OFF the LEDs and Stop the robot
        LED.Clean()

        Blocks.RobotStop()

        #Set to the default value (-1) this variable, -1 is
        associated to error. So if it does not change means that there is an error
        _blocks = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]

        #Set to the default value the Condition Flag
        cond = [1,2,3,4,5]

        for num in cond:
            _ConditionFlags[num]=True

        return
```

```

        if state==load:

            LED.TurnOn(18)

            LED.TurnOn(24)

            _sockets() #Wait the socket

            return

    if state==process:

        LED.Clean()

        _processBlocks(code) #Process all the information re-
ceived

        return

```

If the state is idle, it runs all the conditions. To be able to run all the conditions, the program runs the function `_Condition(x)` that is located inside a *for* loop. The code can be seen bellow.

```

#Function that executes the conditions
def _Condition(i):
    global _ConditionFlags
    j=(i*2)-1 #Operation need to get the index if the block from the
number of condition
    #Ex. condition 1 = blocks 1,2 ; condition 2 = blocks 3,4...
    print "condition-----"+str(i)+"----blockID---"
"+str(_blocksId[j])+"-----positionBlock-----"+str(j)
    if(_blocks[j]==2): #If the block is a condition block
        print "Condition-----"+str(_blocksId[j])
        a=Blocks.ExWhenBlocks(_blocksId[j], buttonpress) #if
the condition is achieve return 1
        if ((a==1) | (_blocksId[j+1]== 20004) | (a== -1)): #if the
condition is achieve do this or if you want to control the car
        #If you want to control the car when you don't press
it has to stop

        e=Blocks.ExDoBlocks(_blocksId[j+1],buttonpress,a)

    elif(_blocks[j]==1): #if the block is a movement block
        if _ConditionFlags[i]: #check is the flag of the con-
dition related with this block is True
            print "Move-----"
            a=Blocks.ExMovementBlocks(_blocksId[j])

#Execute the code

    if a ==1:
        print "DONE"

```

```

                                _ConditionFlags[i]=False    #Set
the condition flag related with this block to False
                                print _ConditionFlags[i]
elif(_blocks[j]==-1):
    print "-----NO CODE-----"

```

The first thing that does is check if the block is a condition block. If is a condition block it executes the function *Blocks.ExWhenBlocks(_blocksId[j], buttonpress)*. This function is located inside the blocks library and returns 1, 0 or -1.

If the condition is done, returns 1. If the condition is not done, returns 0. Finally returns -1 only in the case that is wanted to know if there is or not light, and there is the opposite case. For example: if the block of light has been selected and there is no light it returns -1 and if there is light returns 1. These two blocks (there is or not light) are different to the others. Because if a LED block has been selected as the action block for this condition if there is no light it has to turn off the LED.

After the idle state it goes to the initialize state. There, the first think that it does is turn off the LEDs and stops the robot. To do these actions it uses the library of Blocks and LED. Once these are done it writes the default value to the variable *_blocks* and sets the default value to the *_conditionFlag*. Now all the variables have the default value, so it can start working without troubles.

The next state is the load state. The objective of this state is waiting for the information that the PC program sends using a TCP socket. But first it turns on the LEDs to know that you are waiting for the socket with the information.

The code related with the socket connection is inside the function *_sockets()*. The code of this function is the server code for the function *Client()* located in the PC program.

```

#Socket connection
def _sockets():
    global code
    print "INSIDE SOCKETS"
    Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while(1):
        try:
            Socket.bind((IP, PORT))
            break
        except socket.error:

```

```

        print "-----ADRESSS IN USE-----"
        time.sleep(0.2)

    Socket.listen(1)
    while 1: #Run this until it recives data
        connection, address = Socket.accept()
        code = connection.recv(BUFF_SIZE)
        print code

        if not code: print "no data"
    print "received data:", code
        break
    time.sleep(0.2)
    connection.close()
    Socket.close()

```

The function listens in the port *PORT* (variable defined, is the same number that the one defined in the client). And when it receives a connection reads it and stores the information in the variable *code*. In the next state this information will be processed.

Finally there is the process state. The information received in the previous state is processed, it makes this information useful. But first it turns off the LEDs using the function *LED.Clean()* (located in the LED library).

To process the blocks it uses the function *_processBlocks(code)*.

```

#Process the information received.
#Writes in the variable _blocks what kind of block is received
#and also counts the number of conditions
def _processBlocks(str):

    global _blocks,_numberCondition

    _movement = False
    _when = False

    str=str.split('-')
    num = len(str)
    i=0
    print str

    for code in str:
        code = str[i]

        print code
        if i==0:

            _numberCondition=code

    elif _movement:

```

```

if Blocks.SearchMovementBlocks(int(code)):
    print "OK"

    _blocks[i]=1
    _movement= True
elif int(code)==55555:
    print "OK"
    _blocks[i]=1
    _movement= True

elif Blocks.SearchWhenBlocks(int(code)):
    print "OK"

    _blocks[i]=2
    _movement = False
    _when = True
else:
    print "Error"

    _blocks[i]=-1

    _Movement = False
    break

elif _when:
    if Blocks.SearchDoBlocks(int(code)):
        print "OK"

        _blocks[i]=3
        _when= False
    else:
        print "Error"

        _blocks[i]=-1
        _when = False
        break
else:
    if (Blocks.SearchMovementBlocks(int(code))==True):
        print "OK"

        _blocks[i]=1
        _movement = True
    elif Blocks.SearchWhenBlocks(int(code)):
        print "OK"

        _blocks[i]=2
        _when = True

```

```

elif Blocks.SearchDoBlocks(int(code)):
    print "Error"

    _blocks[i]=-1
    break

    _blocksId[i]=int(code)
    i=i+1

```

This code is very similar to the *logicBlock(String tag)* function, used in the PC program. The main difference is that now it stores the information received. Stores the ID of the block in the variable *_blocksId[]*, the kind of block in the variable *_blocks[]* and the number of conditions in the variable *_numberCondition*. The number of conditions is the first information that is stored in the string. So when the information arrives, is the first thing that stores and after there are the ID of the blocks.

3.7.3 State logic function

The state diagram is followed by this function. Is the responsible to change the states.

```

#State logic of the FSM
def _state_logic():
    global state
    if state==idle:
        print buttonpress
        if (buttonpress & cwiid.BTN_HOME): #If the user press
the button home in the Wii Mote
            state = initialize
        return
    if state==initialize:
        state=load
        return
    if state ==load:
        state = process
        return

    if state==process:
        print _blocks
        state=idle
        return

```

It can be seen that the code above follows the state diagram (PICTURE 9). When the program is in one state it changes the value of the variable *state* to go to the next state.

3.8 Blocks library

This file can be found in the Appendix 3. All the code related with the blocks is located in this file. Each block is declared as a function. Dictionaries are used to associate each block with an ID, where the ID is the key word of the function. Can be seen in the code bellow

```
_MovementBlocks = {00000: Forward,
                    00001: Back,
                    00002: Left,
                    00003: Right,
                    }
```

It is done with the three kinds of blocks (Movement, Condition and Action). So there are three dictionaries: *_MovementBlocks* (Movement), *_WhenBlocks* (Condition) and *_DoBlocks* (Action).

3.8.1 Movement blocks

All the movement blocks have similar code. The only difference is the letter that they send. But first has to be define to where the information has to be send.

In all the code is used the word arduino instead of DFRduino Romeo because is shorter. And as is said in the Hardware Used section is very similar to Arduino Duemilanove.

```
#define arduino
try:
    arduino = serial.Serial('/dev/ttyUSB1', 19200) #19200 => Bit rate
of the communication
except serial.serialutil.SerialException: #If is plugged to the other USB
    arduino = serial.Serial('/dev/ttyUSB0', 19200)
#Movement blocks
def Forward():
    arduino.write('o')
    return 1
```

A letter is sent and the arduino processes the information received and executes the code. The code related with it will be explained in the section DFRduino Romeo code.

3.8.2 Condition blocks

The condition blocks can be splitted in two kinds: Condition of light (two blocks) and condition using the Wii Move (three blocks).

The first type are used to detect if there is or no light. To be able to do that they use an LDR connected to arduino, the information is sent using the serial port.

```
#Condition blocks (When blocks)
def NoLight():
    arduino.write('b') #Measure light
    sensor=arduino.read()
    if sensor == "f": #There is no light
        return 1
    else: #there is light
        return -1
```

This function returns 1 if the condition is achieved and -1 if not.

The code for the block that check if there is light is more or less the same, the only difference is in the *if*. It has:

```
if sensor == "t": #There is light
```

The other kind of blocks have ,as an input, the button press. The only difference in the function of these blocks is the *if*. Each block has the *if* related with the buttons that you have to press.

```
def Button1(buttonpress):
    if (buttonpress & cwiid.BTN_1): #Press button 1
        print 'Button 1 pressed'
        time.sleep(0.1)
        return 1
```

3.8.3 Action blocks

The action blocks can be separated in three types: Song, LED and move. All of them return 1.

The first type contains the two blocks related with the songs. To reproduce a song it uses *aplay* (command-line audio file player) that reproduces the song. The only difference between the two song blocks is the file that it reproduces.

```
#Action blocks (Do blocks)
def SongA(a):
    if a==1:
        print "Song A"
        p = subprocess.Popen(["aplay", "a.wav"]) #Reproduce
a.wav
        return 1
```

The next type contains also two blocks. These two blocks are the ones associated to each LED color. The only difference between the code of them is the PIN selected (Green LED PIN 24 and Red LED PIN 18).

```
def GreenLED(a):
    if a==1:
        #GREEN LED is on PIN24
        LED.TurnOn(24)
        return 1
    elif a==-1:
        LED.TurnOff(24)
        return 1
```

Finally, the move block. This block is a bit different because has to know which button are you pressing.

```
def Move(buttonpress):
    print "Move"
    if (buttonpress & cwiid.BTN_UP):
        arduino.write('w') #Move Forward
    if (buttonpress & cwiid.BTN_DOWN):
        arduino.write('s') #Move Back
    if (buttonpress & cwiid.BTN_LEFT):
        arduino.write('a') #Turn left
    if (buttonpress & cwiid.BTN_RIGHT):
        arduino.write('d') #Turn right
    if buttonpress == 0:
        arduino.write('x') #Stop
    return 1
```

This function sends a letter to arduino and it executes it.

3.8.4 Search function

There is a search function for each kind of block. The objective of this function is know if the block exists or not. The code bellow is the one used to know if the block is a movement block, the code is similar for the other kinds of blocks (only change the dictionary used).

```
#Search blocks
def SearchMovementBlocks(ID):
    if ID in _MovementBlocks:
        return True
    else:
        return False
```

The function search if there is a key word with the value ID in the dictionary *_MovementBlocks*. In case that exists returns True, if not it returns False.

3.8.5 Execute function

The code related with the action that the device has to do to execute each block is inside this function. The execute function is different for each kind of block. Each function has an *except KeyError*, just in case that the ID cannot be found in the dictionary. Also all of them return an integer.

```
#Execute blocks
def ExMovementBlocks(ID): #Execute code for Movement Blocks
    try:
        _MovementBlocks[ID]()
        return 1
    except KeyError:
        print "Key Error"
```

The code related with the Movement blocks only has to do the code of the block. In this case there is no need to get a feedback of the function because is just a predetermined movement.

```
def ExDoBlocks(ID, buttonpress,a): #Execute code for Action Blocks
    try:
        if ID==20004: #free move block
            e=_DoBlocks[ID](buttonpress)
        else:
```

```

                                e=_DoBlocks[ID] (a)
                                return 1
except KeyError:
    print "Key Error"

```

The code related with the action blocks has some differences. The first difference is that has an *if*. It is used to differentiate the free move block (you can move the robot using the joystick) from the others blocks. This block is special because has to know which button are you pressing.

The others blocks just need to know the variable *a*. Actually this variable is only needed for the blocks related with the LEDs, but to make the code simple is passed to the LEDs and song blocks. It is used by the LEDs blocks only in the case that you have selected a light block as condition, so it turns off the LEDs when the condition is false.

```

def ExWhenBlocks(ID, buttonpress): #Execute code for Condition Blocks
    a=0
    try:
        if (ID == 10002) | (ID == 10003) | (ID == 10004):
            a=_WhenBlocks[ID] (buttonpress)
        else:
            a=_WhenBlocks[ID] ()

        return a
    except KeyError:
        print "Key Error"

```

Finally the code related with the condition blocks. This code has also an *if*. The *if* is used for the condition blocks, related with the Wii mote, where a button has to be press. And the other two blocks (there is or not light) do not need to pass any value.

In this function the value returned is not 1. The value returned is the variable *a* that is 1, 0 or -1. If the condition is achieved 1 is given back. When the condition related with the Wii mote is not accomplished 0 is returned .And when a light block is selected to indicate that the condition is not achieved -1 is given back.

3.9 LED library

This file can be found in the Appendix 4. In this file is located the code related with the GPIO PINs used of RPi. A library containing all the code necessary code has to be imported.

```
import RPi.GPIO as GPIO, time
```

There are three functions: *TurnOn(PIN)*, *TurnOff(PIN)* and *Clean()*.

The first function is used to turn on a LED. The location of the LED is given by the variable *PIN*.

```
def TurnOn(PIN):
    #Green = PIN24
    #Red = PIN18
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, True) #Turn on
```

The second function is used to turn off a LED. The code used is very similar that the one used in the function *TurnOn(PIN)*.

```
def TurnOff(PIN):
    #Green = PIN24
    #Red = PIN18
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False) #Turn off
```

The third function is used to clean all the PINs selected. First the function defines all the PINs that can be used as output and turn it off and after clean them.

```
def Clean(): #Clean all the outputs
    GPIO.setup(18, GPIO.OUT)
    GPIO.output(18, False) #Turn off
    GPIO.setup(24, GPIO.OUT)
    GPIO.output(24, False) #Turn off
    GPIO.cleanup() #Clean all the pins assigned, so they can be re-
used
```

3.10 DFRduino Romeo code

The DFRduino Romeo has to process and execute all the code received of the RPi. The code is received using a serial connection. In the code of the RPi it is declared as a variable *arduino*. All the code can be seen in the Appendix 5.

The core of the code is located inside the *void loop(void)*.

```
void loop(void)
{
    if(Serial.available()){
        char val = Serial.read();
        if(val != -1)
        {
            switch(val)
            {
                case 'w':
                    forward (255,255);    //move forward in max speed
                    break;
                case 's':
                    backward (255,255);    //move back in max speed
                    break;
                case 'a':
                    left (255,255); //turn left in max speed
                    break;
                case 'd':
                    right (255,255); //turn right in max speed
                    break;
                case 'x':
                    stop();
                    break;
                case 'o':
                    forward(125,125);
                    delay(2000); //move forward 2s
                    stop();
                    break;
                case 'i':
                    left(255,255);
                    delay(1500); //move left 1.5s
                    stop();
                    break;
                case 'p':
                    right(255,255);
                    delay(1500); //move right 1.5s
                    stop();
                    break;
                case 'l':
                    backward(125,125);
                    delay(2000); //move backward 1.5s
                    stop();
                    break;
                case 'b':
                    sensorValue = analogRead(LDR);    //read the sensor
```

```

        if(sensorValue>89) //There is light
        {
            Serial.write("t");
        }
        else
        {
            Serial.write("f");
        }
    }
}
else stop();
}
}

```

As can be seen in the code above, there is a switch and each code received is a case. Using a switch the code becomes simpler and easy to understand. Before do the *loop(void)* it has to initialize all the variables and define the pins used. All the code related with the movement is located in functions. This functions are called in the *void loop(void)* function. The function code is quite long and can be seen in the appendix. The code related with the movement is taken from the example located in this webpage: http://www.dfrobot.com/wiki/index.php/Romeo_V2-All_in_one_Controller_%28R3%29_%28SKU:DFR0225%29

3.11 Wireless connection

The connection between the RPi and the PC is made using a wireless connection. To do this connection an ad-hoc network is created. This network is created by the RPi using Hostapd. Hostapd is a daemon for wireless access point; it runs in the background and allows the creation of a wireless network.

First, Hostapd has to be installed in the RPi. After install the software, a configuration file has to be created to configure Hostapd. This file is created inside the folder `/etc/hostapd/`. The file has to contain all the information necessary to create the ad-hoc network. This code can be found in the Appendix 6.

After configure the file is important define a fixed IP address for the RPi. This is done modifying the file `/etc/network/interfaces` and adding these lines:

```

iface wlan0 inet static:
    address 192.168.2.3
    netmask 255.255.255.0

```

The first line defines the interface that has been defined. The next two lines define the configuration values of the interface.

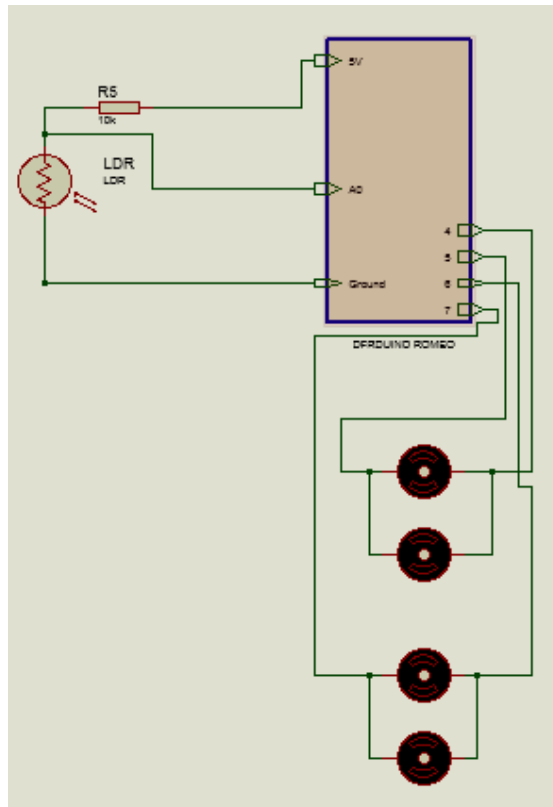
Now the RPi is able to create an ad-hoc network that allows the PC to connect. As there is no DHCP server running in the RPi, the PC will have a static IP (for example: 192.168.2.1).

3.12 Hardware architecture

The last step of the robot design has been the connection of all the hardware components to RPi and DFRduino Romeo. The connection has been done in a breadboard, soldering the components.

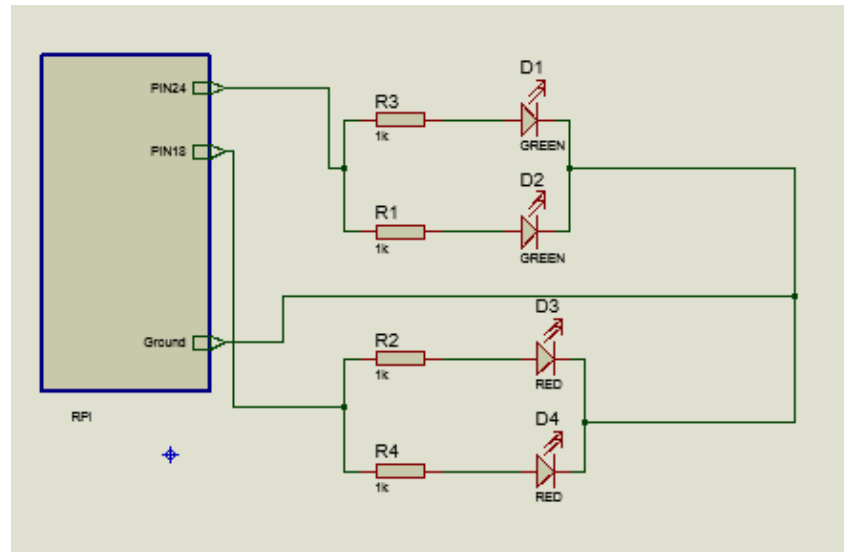
In PICTURE 12 can be seen all the connections done in the DFRduino Romeo.

It only controls the LDR and the motors of the robot.



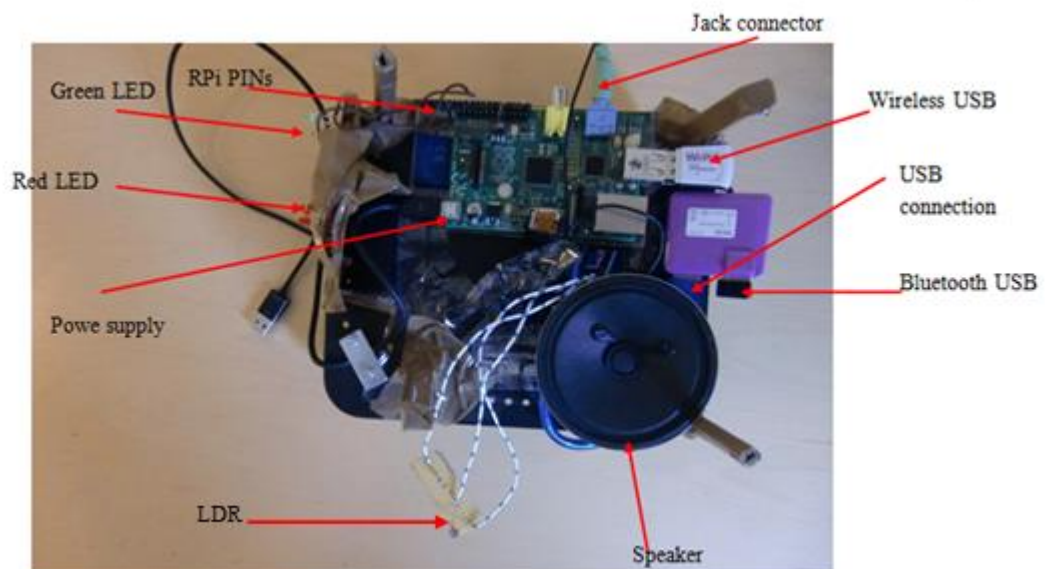
PICTURE 12. DFRduino Romeo schematic

The RPi controls only the LEDs using their PINs, the other hardware peripherals (speakers, Bluetooth USB, wireless USB) are controlled with the corresponding connectors. The schematic of the PIN connection in the RPi can be seen in PICTURE 13.



PICTURE 13. RPi schematic

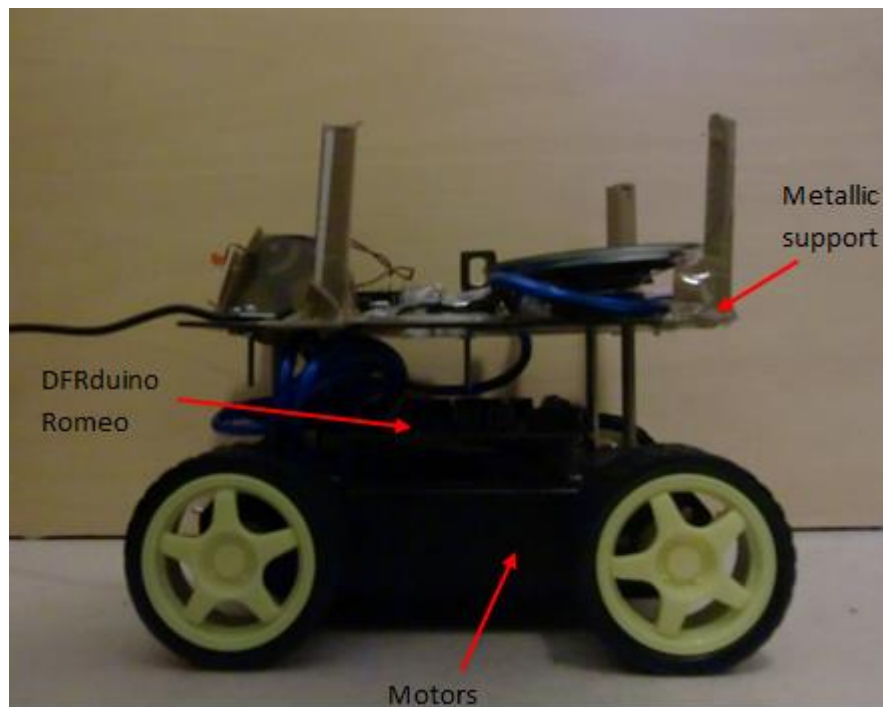
After define the architecture of the robot the last step has been assembling all together. The result can be seen in PICTURE 14.



PICTURE 14. Up picture of the robot

Almost all the components are located on a metallic support. The DFRduino Romeo is located down the support. It is connected with an USB to the RPi.

The motors are located inside the metallic structure near the wheels with 5 AA batteries. These batteries are used to feed the motors with the necessary power. PICTURE 15 helps to understand better where is the DFRduino Romeo placed.



PICTURE 15. Side picture of the robot

4 TEST

The device has been tested in the event of 9th-10th of May. In both days a survey has been filled looking the reaction of the children by the writer of this thesis. The survey can be found in the Appendix 7. The survey was filled after children stopped playing with the device. In case that they were in group it was filled looking the average reaction of children.



PICTURE 16. Children playing with the robot

The robot has been located in a room with other technological devices and children had the opportunity to test it by their own.

The first day, 9th of May, the robot was located in a sports hall with all the technological devices brought to the event. There were seven devices located near the wall of the sports hall. The children were in groups of ten and they were able to choose which device they wanted to play. There was a limitation of 15min in each device so children were able to visit all the devices.

The second day, 10th of May, the robot was located in a hall inside the school. This time the robot was in the same hall with other two devices. Also children did not have a limitation of time and they were able to play as long as they wanted.

Both days the robot was located in the floor and the PC on a table. The author of the thesis helps them to start with a demo. The help was limited because there were some troubles with the language, some children don't speak English and the author of the thesis does not speak Finnish.



PICTURE 17. Children playing with the robot

4.1 Results of the survey

More than a half (61%) does not understand how the device works without reading the rules. Also almost the same number of children (66%) understands the rules and was able to play alone.

Practically all of them play without troubles (88%) and the device works correctly.

Three quarters of children (77%) like the device.

When children were playing with the device they made some interesting comments:

One girl said: "I want to have it as a pet".

Some children said that they find cute the device because it looks like a ladybird.

A group of girls said that is a device for boys, because you can control it as a remote control car.

The first day the device worked without troubles, it only stopped working two times. The first time was because a child unplugs the power supply. The second time was because a child presses the synchronization button on the Wii Mote.

The second day the device worked fine the first two hours, but then a child crashes the car many times with the wall and it stops working. The problem was that the USB Bluetooth of the RPi broke and also some wires were touching and making a short circuit. After the event the problem was solved and now it works fine.

Both days there were two patterns that the children followed. One was that almost all of them were able to play and understand it easier after a demo. The second was that the most attractive option was program the device as a remote control car.

5 DISCUSSION

The results of the survey are really good.

The author is really surprised by the fact that children programmed and enjoyed playing with the robot very enthusiastically. But there were the next factors that could affect the results:

- The survey was filled by the author of the Thesis. Although he tried to be impartial the results can be biased in some degree. This fact will be solved if the children will be able to fill themselves the survey. This can be done creating a computer program with images, so children could understand it, and let the children fill it. Or also can be solved asking to the parents to fill the survey in the case of the second day.
- The first day children were in groups of ten and the survey was filled looking the average reaction. As a consequence of this, the results don't express all the reaction of children. The best solution for this problem is give a survey to every children.

After seen these results, there are some improvements of the survey that would good to do in future versions. There are two questions: Do you play without troubles? And is there any problem with the device? Both are very similar. The results of these two questions are the same, which means that are not really useful. So will be a good improvement delete one of these questions.

There are two questions that would be interesting to add: Gender and what is your favourite combination?

The first question can look not really important but during the event, there were different reactions between boys and girls. So this question would help to understand better if they like the device and if there understand the rules. And can help to know if there is any relation with the gender and the choices.

The second question will help to know if they just like the device because you can use it as an RC car. Or really they like because is flexible in terms of multiple combinations of actions (you can combine any red block with any blue block).

6 CONCLUSION

The device was tested in an event of two days with children and the results were good. The first day the device worked without troubles (only two). But the second day there was a big trouble that made it impossible to work. As it was the first test with children and the device was a prototype, the results can be considered satisfactory. More than the half of the time, the device was working without troubles and these were good results for a first prototype. The problem, in the second day, was related with the cover because it was made of cardboard. This material is not suitable to protect the hardware.

One important point is that they liked a lot playing with the device like a remote controlled car. They liked it because this option is the closest one to their known games. Also it is similar to the toys that they possibly have at home.



PICTURE 18. Children programming the robot

Children didn't have any problem to use successfully all the movement blocks. Because they could look quickly the results and saw what happened. Although these blocks were no related with the logic of programming that the device wanted to teach.

Other point was that children were not able to understand how the device works without reading the rules but some of them didn't read fluently. A possibly solution could be making the device simpler.

After knowing the rules more than a half of the children were able to understand the rules and play alone. One important point to make it easier to understand is making a demo of how it works or showing a video with an example.



PICTURE 19. Children playing with the robot

There are some hardware aspects that would be improved in future prototypes: Speakers and LEDs. The volume of speakers is not enough high and sometimes was difficult listen to the music. There are too less LEDs and is not easy to see all of them all the time. Another aspect related with the robot that would be improved in future versions is the cover of it. In this version the cover is done with cardboard and is no enough strong. For that reasons will be good to design a cover using plastic or aluminium.

In plastic because you can make a lot of shapes and is stronger than cardboard, one option to do it will be using a 3D printer, there are a lot of different shapes can be done using a 3D printer.

In aluminium because is stronger than plastic and cardboard, and is more or less easy do the cover.

The UI can also be improved because half of children had some difficulties following the rules. To solve this, the UI should be clearer in that aspect. One possible solution

will be place a diagram with the rules in the UI, so then will be easier to follow the rules. Another will be showing in a panel the colour of the next block that has to be selected. Doing this option will be more intuitive follow the rules.

REFERENCES

Lutz, M and Ascher, D. 2003. Learning Python. O'Reilly 97-98, 103-111, 115-116, 189-225, 257-275, 307-316, 393.

G. Kochan, S and Wood, P. 2003. UNIX Shell Programming. SAMS 19-26, 30-34, 83, 115-120, 390-391

Johnson, J. 2010. Designing with the mind in mind. Academic Press 8, 53, 97, 120-123.

Peters, D. 2013. Interface Design for Learning: Design Strategies for Learning Experiences. New Riders 76-79, 81-82, 86,89, 97

Tutorialspoint. Tutorial .Read 6.3.2014.
http://www.tutorialspoint.com/python/python_for_loop.htm.

Ubuntu. Manual. Read 17.3.2014
<http://manpages.ubuntu.com/manpages/precise/man1/zbarcam.1.html>

Python. Tutorial. Read 19.3.2014. <http://docs.python.org/2/tutorial/classes.html>.

Daniweb. Web page. Read 1.4.2014. <http://www.daniweb.com/software-development/python/threads/145792/variable-referenced-before-assignment>.

Bytebaker. Web page. Read 3.4.2014. <http://bytebaker.com/2008/11/03/switch-case-statement-in-python/>.

StackOverflow. Forum. Read 3.4.2014.
<http://stackoverflow.com/questions/1602934/check-if-a-given-key-already-exists-in-a-dictionary>.

Python. Documentation. Read 7.4.2014.
<https://docs.python.org/2/library/subprocess.html>

StackOverflow. Forum. Read 15.4.2014.
<http://stackoverflow.com/questions/14388706/socket-options-so-reuseaddr-and-so-reuseport-how-do-they-differ-do-they-mean-t>.

Python. Tutorial. Read 15.4.2014.
<https://docs.python.org/2/library/socket.html?highlight=socket%20tcp>.

Arduino. Manual. Read 18.4.2014. <http://playground.arduino.cc/Main/DCMotorControl>

Make. Tutorial. Read 23.4.2014. <http://makezine.com/projects/tutorial-raspberry-pi-gpio-pins-and-python/>.

Adafruit. Tutorial. Read 23.4.2014. <https://learn.adafruit.com/raspberry-pi-e-mail-notifier-using-leds/python-script>.

RasPi.TV. WebPage. Read 23.4.2014. <http://raspi.tv/2013/rpi-gpio-basics-3-how-to-exit-gpio-programs-cleanly-avoid-warnings-and-protect-your-pi>.

Python. Tutorial. Read 23.4.2014.
<https://docs.python.org/2/library/stdtypes.html#str.split>.

Brianhensley. Blog. Read 23.4.2014. <http://www.brianhensley.net/2012/08/wii-controller-raspberry-pi-python.html>.

Python. Tutorial. Read 24.4.2014.
<https://docs.python.org/2/library/queue.html?highlight=queue#Queue>.

Brainy Betty. Tutorial. Read 24.4.2014.
<http://www.brainybetty.com/soundsforpowerpoint2.htm>.

Arduino. Tutorial. Read 24.4.2014. <http://arduino.cc/en/Serial/write>.

Make. Tutorial. Read 24.4.2014. <http://makezine.com/projects/tutorial-raspberry-pi-gpio-pins-and-python/>.

Novitiate. Tutorial. Read 25.4.2013. <http://www.novitiate.co.uk/?p=183>.

Adafruit. Tutorial. Read 28.4.2014. <https://learn.adafruit.com/downloads/pdf/setting-up-a-raspberry-pi-as-a-wifi-access-point.pdf>.

Instructables. Tutorial. Read 28.4.2014. <http://www.instructables.com/id/How-to-make-a-WiFi-Access-Point-out-of-a-Raspberry/>.

Andremiller. Diagram. Read 29.4.2014. <http://www.andremiller.net/content/raspberry-pi-pinout-diagram>.

Raspberry Pi. Description. Read 2.5.2013. <http://www.raspberrypi.org>.

Openclipart. Image. Read 8.5.2014.
http://openclipart.org/image/300px/svg_to_png/188214/1383334832.png.

DFRduino Romeo. Manual. Read 13.5.2014.
http://www.dfrobot.com/wiki/index.php/DFRduino_Romeo-All_in_one_Controller_%28SKU:DFR0004%29

APPENDICES

Appendix 1. UI code (PC program)

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Program
{
    public partial class Form1 : Form
    {
        /* _previous block values
        * i=0 => Move block & initial value
        * i=1 => Condition block
        * i=2 => Action block
        */
        private int _previousModeBlock;
        private string currentdirectory;
        private Image white;
        private Object white_tag;
        private string IP = "192.168.2.3"; //Wireless IP
        // private string IP = "192.168.137.2"; //Etherne IP (just for test)
        List<PictureBox> Selected_images = new List<PictureBox>();
        List<Panel> _selectedPanels = new List<Panel>();
        private int _conditionNumber = 0;

        public Form1()
        {
            InitializeComponent();
            pictureBox1.Tag = "00002";
            pictureBox2.Tag = "00000";
            pictureBox3.Tag = "00001";
            pictureBox4.Tag = "00003";
            pictureBox5.Tag = "10000";
            pictureBox6.Tag = "10001";
            pictureBox7.Tag = "10002";
        }
    }
}

```

```

        pictureBox8.Tag = "10003";
        pictureBox9.Tag = "10004";
        pictureBox10.Tag = "20000";
        pictureBox11.Tag = "20001";
        pictureBox12.Tag = "20002";
        pictureBox13.Tag = "20003";
        pictureBox14.Tag = "20004";
        _previousModeBlock = 0; //Initial value
        currentdirectory = Directory.GetCurrentDirectory();
        white = new Bitmap(Image.FromFile(currentdirectory +
        "\\images\\55555.png"));
        white_tag = "55555";
        initializepictures();

    }

    private void initializepictures()
    {
        //Initialize pictureBox
        Selected_images.Add(Selected_Picture1);
        Selected_images.Add(Selected_Picture2);
        Selected_images.Add(Selected_Picture3);
        Selected_images.Add(Selected_Picture4);
        Selected_images.Add(Selected_Picture5);
        Selected_images.Add(Selected_Picture6);
        Selected_images.Add(Selected_Picture7);
        Selected_images.Add(Selected_Picture8);
        Selected_images.Add(Selected_Picture9);
        Selected_images.Add(Selected_Picture10);

        //Initialize panel
        _selectedPanels.Add(panel1);
        _selectedPanels.Add(panel2);
        _selectedPanels.Add(panel3);
        _selectedPanels.Add(panel4);
        _selectedPanels.Add(panel5);
        _selectedPanels.Add(panel6);
        _selectedPanels.Add(panel7);
        _selectedPanels.Add(panel8);
        _selectedPanels.Add(panel9);
        _selectedPanels.Add(panel10);
    }

```

```

public void Client()
{
    byte[] data;
    string input, stringData;
    TcpClient server;
    string selected;

    try
    {
        server = new TcpClient(IP, 6112);
    }
    catch (SocketException)
    {
        Console.WriteLine("Unable to connect to server");
        return;
    }

    NetworkStream ns = server.GetStream();
    selected = send_info();
    data = new byte[Encoding.ASCII.GetBytes(selected).Length];
    data = Encoding.ASCII.GetBytes(selected);
    ns.Write(data, 0, data.Length);
    // Console.WriteLine("Disconnecting from server...");
    ns.Close();
    server.Close();
}

private string send_info()
{
    var str = _conditionNumber.ToString();
    for (int i = 0; i < _conditionNumber; i++)
    {
        str = str + "-" + Selected_images[(2*i)].Tag + "-" + Selected_images[(2*i)+1].Tag;
    }
    return str;
}

public bool LogicBlocks(String tag)
{
    //Image a = new Image();

    /*
    * i=0 => Move block & initial value
    * i=1 => Condition block
    * i=2 => Action block
    */
}

```

```

int i = -1;
try
{
    i=Convert.ToInt16(tag)/10000;

}
catch (Exception)
{

    return false;
}

if (i == -1)
    return false;
switch (_previousModeBlock)
{
    case 0:
        if (i == 0)
        {
            _previousModeBlock = i;
            return true;
        }
        if (i == 1)
        {
            _previousModeBlock = i;
            return true;
        }
        return false;
    case 1:
        if (i == 2)
        {
            _previousModeBlock = i;
            return true;
        }
        return false;
    case 2:
        if (i != 2)
        {
            _previousModeBlock = i;
            return true;
        }
        return false;
    default:
        return false;
}
}

```

```

        private void Select_Picture(Image image, Object tag, Boolean
moveBlock)
        {
            if (LogicBlocks(tag.ToString()))
            {
                Draw_picture(image, tag, moveBlock);
            }
        }

        private void Draw_picture(Image image, Object tag, Boolean moveBlock)
        {
            for (int i = 0; i <10; i++)
            {
                if (Selected_images[i].Image == null)
                {
                    Selected_images[i].Image = image;
                    Selected_images[i].Tag = tag;
                    Selected_images[i].SizeMode = PictureBoxSizeMode.Zoom;//To
resize the image
                    try // Paints the panel that is down the image, so you
know the kind of image
                    {
                        panelColor(i, Convert.ToInt32(tag.ToString()));
                    }
                    catch (Exception)
                    {
                        Console.WriteLine("Error");
                        throw;
                    }
                    if (moveBlock)//Next image is a white image
                    {
                        Draw_picture(white, white_tag, false);
                    }
                    break;
                }
            }
        }

        private void panelColor(int i, int tag)
        {
            switch (tag/10000)
            {
                case 0:
                    _selectedPanels[i].BackColor = Color.GreenYellow;
                    break;
                case 1:

```

```

        _selectedPanels[i].BackColor = Color.Red;
        break;
    case 2:
        _selectedPanels[i].BackColor = Color.Blue;
        break;
    case 5:
        _selectedPanels[i].BackColor = Color.GreenYellow;
        break;
    default:
        break;
    }
}

public bool CheckSelection()
{
    int i;
    for (i = 1; i < 10; i++)
    {
        if (Selected_images[i].Image == null)
        {
            if (i%2 != 0)//Means that you have ended the selction with
a condition

                return false;
            else //You have ended the selection with an action or a
movement block

                {
                    break;
                }
        }
    }
    _conditionNumber = i / 2;
    return true;
}

private void pictureBox1_Click(object sender, System.EventArgs e)
{

    Select_Picture(pictureBox1.Image, pictureBox1.Tag,true);

}

private void pictureBox2_Click(object sender, System.EventArgs e)

```

```

{
    Select_Picture(pictureBox2.Image, pictureBox2.Tag,true);
}

private void pictureBox3_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox3.Image, pictureBox3.Tag,true);
}

private void pictureBox4_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox4.Image, pictureBox4.Tag,true);
}

private void pictureBox5_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox5.Image, pictureBox5.Tag,false);
}

private void pictureBox6_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox6.Image, pictureBox6.Tag,false);
}

private void pictureBox7_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox7.Image, pictureBox7.Tag,false);
}

private void pictureBox8_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox8.Image, pictureBox8.Tag,false);
}

private void pictureBox9_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox9.Image, pictureBox9.Tag,false);
}

private void pictureBox10_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox10.Image, pictureBox10.Tag,false);
}

private void pictureBox11_Click(object sender, System.EventArgs e)
{
    Select_Picture(pictureBox11.Image, pictureBox11.Tag,false);
}

```

```

    }

    private void pictureBox12_Click(object sender, System.EventArgs e)
    {
        Select_Picture(pictureBox12.Image, pictureBox12.Tag,false);
    }

    private void pictureBox13_Click(object sender, System.EventArgs e)
    {
        Select_Picture(pictureBox13.Image, pictureBox13.Tag,false);
    }

    private void pictureBox14_Click(object sender, EventArgs e)
    {
        Select_Picture(pictureBox14.Image, pictureBox14.Tag,false);
    }

    private void CleanImage_Click(object sender, EventArgs e)
    {
        _previousModeBlock = 0;
        for (int i = 0; i < 10; i++)
        {
            Selected_images[i].Image = null;
            _selectedPanels[i].BackColor = Color.Transparent;
        }
    }

    private void SendImage_Click(object sender, EventArgs e)
    {
        if (CheckSelection())
        {
            Client();
        }
    }
}
}

```

Appendix 2. Main code (Main.py)

```
#!/usr/bin/python
import socket
import string
import Blocks #Libraries with the code
import LED
import cwiid #wii remote
import time

#Global variables
idle = 'A' #Execute the conditions
initialize = 'B'
load = 'C' #Capture the QR code
process = 'D' #Check that the codes are correct

#Codes recived
code="aar"

#For the socket
PORT = 6112
BUFF_SIZE = 1024
IP = "192.168.2.3"
#IP="192.168.137.2"
numberCondition = 0 #Has the number of conditions that are runed
state=idle

error = False
_blocks=[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1] #This variable is used to know what
kind of block is each block
#-1 = error, 1=movement, 2=when, 3=do
_blocksId=[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]

_numberCondition=0 #Stores the number of conditions that are selected
#All these flags are used only if you have selected a move block, to don't run
the code more than once
_conditionFlag1=True
_conditionFlag2=True
_conditionFlag3=True
_conditionFlag4=True
_conditionFlag5=True

buttonpress = 0 #variable to store the button press on the WiiControl
```

```

#_ConditionFlags is a dictionary that is used to have acces to the
_conditionFlag...
_conditionFlags = { 1: _conditionFlag1,
                    2:
_conditionFlag2,
                    3:
_conditionFlag3,
                    4:
_conditionFlag4,
                    5:_conditionFlag5,}

#Initialize Wiimote
def _initializeWiimote():

    try:
        wii=cwiid.Wiimote()
        print "Wii correct"

    except RuntimeError:
        print "Error opening wiimote connection"
        quit()
    wii.rpt_mode = cwiid.RPT_BTN
    return wii

#Socket connection
def _sockets():
    global code
    print "INSIDE SOCKETS"
    Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while(1):
        try:
            Socket.bind((IP, PORT))
            break
        except socket.error:
            print "-----ADRESSS IN USE-----"
            time.sleep(0.2)
    Socket.listen(1)
    while 1: #Run this until it recives data
        connection, address = Socket.accept()
        code = connection.recv(BUFF_SIZE)
        print code

        if not code: print "no data"
    print "received data:", code
    break
    time.sleep(0.2)
    connection.close()

```

```

Socket.close()

#Process the information received.
#Writes in the variable _blocks what kind of block is received
#and also counts the number of conditions
def _processBlocks(str):
    global _blocks,_numberCondition
    _movement = False
    _when = False
    str=str.split('-')
    num = len(str)
    i=0
    print str
    for code in str:
        code = str[i]
        print code
        if i==0:
            _numberCondition=code

    elif _movement:
        if Blocks.SearchMovementBlocks(int(code)):
            print "OK"
            _blocks[i]=1
            _movement= True
        elif int(code)==55555:
            print "OK"
            _blocks[i]=1
            _movement= True

        elif Blocks.SearchWhenBlocks(int(code)):
            print "OK"
            _blocks[i]=2
            _movement = False
            _when = True
        else:
            print "Error"
            _blocks[i]=-1
            _Movement = False
            break

    elif _when:
        if Blocks.SearchDoBlocks(int(code)):
            print "OK"
            _blocks[i]=3
            _when= False

```

```

else:
    print "Error"
    _blocks[i]=-1
    _when = False
    break
else:
    if (Blocks.SearchMovementBlocks(int(code))==True):
        print "OK"
        _blocks[i]=1
        _movement = True
    elif Blocks.SearchWhenBlocks(int(code)):
        print "OK"
        _blocks[i]=2
        _when = True
    elif Blocks.SearchDoBlocks(int(code)):
        print "Error"
        _blocks[i]=-1
        break
    _blocksId[i]=int(code)
    i=i+1

#Function that executes the conditions
def _Condition(i):
    global _ConditionFlags
    j=(i*2)-1 #Operation need to get the index if the block from the
number of condition
    #Ex. condition 1 = blocks 1,2 ; condition 2 = blocks 3,4...
    print "condition-----"+str(i)+"----blockID---"
    "+str(_blocksId[j])+"-----positionBlock-----"+str(j)
    if(_blocks[j]==2): #If the block is a condition block
        print "Condition-----"+str(_blocksId[j])
        a=Blocks.ExWhenBlocks(_blocksId[j], buttonpress) #if
the condition is achieve return 1
        if ((a==1) | (_blocksId[j+1]== 20004)|(a==-1)):#if the
condition is achieve do this or if you want to control the car
        #If you want to control the car when you don't press
it has to stop

        e=Blocks.ExDoBlocks(_blocksId[j+1],buttonpress,a)

    elif(_blocks[j]==1):#if the block is a movement block
        if _ConditionFlags[i]: #check is the flag of the con-
dition related with this block is True
            print "Move-----"
            a=Blocks.ExMovementBlocks(_blocksId[j])

#Execute the code

    if a ==1:

```

```

print "DONE"
_ConditionFlags[i]=False #Set
the condition flag related with this block to False
print _ConditionFlags[i]

elif(_blocks[j]==-1):
    print "-----NO CODE-----"

#Output logic of the FSM
def _output_logic():
    global state, _blocks, _ConditionFlags

    if state==idle:
        print "-----numberCondition-----"
        "+str(_numberCondition)
        #Run all the conditions
        if _numberCondition>0:
            for x in xrange(1,6):
                _Condition(x)
                if x==int(_numberCondition):
#When has runned all the conditions stop
                print "----"
                BREAK!!!----"
                x=10
                break

        return

    if state==initialize:
        #Turn OFF the LEDs and Stop the robot
        LED.Clean()
        Blocks.RobotStop()
        #Set to the default value (-1) this variable, -1 is
associated to error. So if it does not change means that there is an error
        _blocks = [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
        #Set to the default value the Condition Flag
        cond = [1,2,3,4,5]
        for num in cond:
            _ConditionFlags[num]=True
        return

    if state==load:
        LED.TurnOn(18)
        LED.TurnOn(24)
        _sockets() #Wait the socket
        return

    if state==process:
        LED.Clean()

```

```

        _processBlocks(code) #Process all the information re-
ceived

        return

#State logic of the FSM
def _state_logic():
    global state
    if state==idle:
        print buttonpress
        if (buttonpress & cwiid.BTN_HOME): #If the user press
the button home in the Wii Mote
            state = initialize
        return
    if state==initialize:
        state=load
        return
    if state == load:
        state = process
        return

    if state==process:
        print _blocks
        state=idle
        return

#Check which button is press
def _check_buttons():
    global wii,buttonpress
    buttons = wii.state['buttons']
    buttonpress=buttons

#initialize wii remote
wii=_initializeWiimote()

if __name__ == "__main__":
    #FSM
    while(1):

```

```
_check_buttons()  
_output_logic()  
_state_logic()
```

Appendix 3. Blocks code (Blocks.py)

```
#!/usr/bin/python
import LED
import cwiid
import time
import serial
import subprocess

#define arduino
try:
    arduino = serial.Serial('/dev/ttyUSB1', 19200)#19200 => Bit rate
of the communication
except serial.SerialException: #If is plugged to the other USB
    arduino = serial.Serial('/dev/ttyUSB0', 19200)

#Movement blocks
def Forward():
    arduino.write('o')
    return 1

def Back():
    arduino.write('l')
    return 1

def Left():
    arduino.write('i')
    return 1

def Right():
    arduino.write('p')
    return 1

#Condition blocks (When blocks)
def NoLight():
    arduino.write('b') #Measure light
    sensor=arduino.read()
    if sensor == "f": #there is no light
        return 1
    else:#there is light
        return -1

def Light():
    print "Light"
    arduino.write('b') #Measure light
```

```

        sensor=arduino.read()
        print sensor
        if sensor == "t":#there is light
            return 1
        else: #there is no light
            return -1

def ButtonMove(buttonpress):
    if (buttonpress & cwiid.BTN_UP): #Press up
        return 1
    if (buttonpress & cwiid.BTN_DOWN): #Press down
        return 1
    if (buttonpress & cwiid.BTN_LEFT): #Press left
        return 1
    if (buttonpress & cwiid.BTN_RIGHT): #Press right
        return 1

def Button1(buttonpress):
    if (buttonpress & cwiid.BTN_1): #Press button 1
        print 'Button 1 pressed'
        time.sleep(0.1)
        return 1

def Button2(buttonpress):
    if (buttonpress & cwiid.BTN_2): #Press button 2
        print 'Button 2 pressed'
        time.sleep(0.1)
        return 1

#Action blocks (Do blocks)
def SongA(a):
    if a==1:
        print "Song A"
        p = subprocess.Popen(["aplay", "a.wav"]) #Reproduce
a.wav
        return 1

def SongB(a):
    if a==1:
        print "Song B"
        p = subprocess.Popen(["aplay", "b.wav"]) #Reproduce
b.wav
        return 1

def GreenLED(a):
    if a==1:
        #GREEN LED is on PIN24

```

```

        LED.TurnOn(24)
        return 1
    elif a==1:
        LED.TurnOff(24)
        return 1

def RedLED(a):
    if a==1:
        #RED LED is on PIN18
        LED.TurnOn(18)
        return 1
    elif a==1:
        LED.TurnOff(18)
        return 1

def Move(buttonpress):
    print "Move"
    if (buttonpress & cwiid.BTN_UP):
        arduino.write('w') #Move Forward
    if (buttonpress & cwiid.BTN_DOWN):
        arduino.write('s') #Move Back
    if (buttonpress & cwiid.BTN_LEFT):
        arduino.write('a') #Turn left
    if (buttonpress & cwiid.BTN_RIGHT):
        arduino.write('d') #Turn right
    if buttonpress == 0:
        arduino.write('x') #Stop
    return 1

def RobotStop():
    arduino.write('x') #Stop

#Create the dictionaries
_MovementBlocks = {00000: Forward,
                    00001: Back,
                    00002: Left,
                    00003: Right,
                    }

_WhenBlocks = {10000: NoLight,
               10001: Light,
               10002: ButtonMove,
               10003: Button1,
               10004: Button2,
               }

```

```

_DoBlocks = {20000: SongA,
              20001: SongB,
              20002: GreenLED,
              20003: RedLED,
              20004: Move,}

#Search blocks
def SearchMovementBlocks(ID):
    if ID in _MovementBlocks:
        return True
    else:
        return False

def SearchDoBlocks(ID):
    if ID in _DoBlocks:
        return True
    else:
        return False

def SearchWhenBlocks(ID):
    if ID in _WhenBlocks:
        return True
    else:
        return False

#Execute blocks
def ExMovementBlocks(ID): #Execute code for Movement Blocks
    try:
        _MovementBlocks[ID]()
        return 1
    except KeyError:
        print "Key Error"

def ExDoBlocks(ID, buttonpress,a): #Execute code for Action Blocks
    try:
        if ID==20004:
            e=_DoBlocks[ID](buttonpress)
        else:
            e=_DoBlocks[ID](a)
        return e
    except KeyError:
        print "Key Error"

def ExWhenBlocks(ID, buttonpress): #Execute code for Condition Blocks
    a=0
    try:

```

```
        if (ID == 10002) | (ID == 10003) | (ID == 10004):  
            a=_WhenBlocks[ID] (buttonpress)  
        else:  
            a=_WhenBlocks[ID] ()  
  
        return a  
except KeyError:  
    print "Key Error"
```

Appendix 4. LED code (LED.py)

```
import RPi.GPIO as GPIO, time

GPIO.setmode(GPIO.BCM) #Defines the order of the PINs that you use

def TurnOn(PIN):
    #Green = PIN24
    #Red = PIN18
        GPIO.setup(PIN, GPIO.OUT)
        GPIO.output(PIN,True) #Turn on

def TurnOff(PIN):
    #Green = PIN24
    #Red = PIN18
        GPIO.setup(PIN, GPIO.OUT)
        GPIO.output(PIN,False)#Turn off

def Clean(): #Clean all the outputs
    GPIO.setup(18, GPIO.OUT)
    GPIO.output(18,False)#Turn off
    GPIO.setup(24, GPIO.OUT)
    GPIO.output(24,False)#Turn off
    GPIO.cleanup() #Clean all the pins assigned, so they can be re-
used
```

Appendix 5. DFRduino Romeo code

```
//Standard PWM DC control
int E1 = 5;      //M1 Speed Control
int E2 = 6;      //M2 Speed Control
int M1 = 4;      //M1 Direction Control
int M2 = 7;      //M1 Direction Control

//For the LDR
int LDR = A0;
int sensorValue = 0;

void stop(void)                                //Stop
{
    digitalWrite(E1,LOW);
    digitalWrite(E2,LOW);
}
void forward(char a,char b)                    //Move forward
{
    analogWrite (E1,a);
    digitalWrite(M1,LOW);
    analogWrite (E2,b);
    digitalWrite(M2,LOW);
}
void backward (char a,char b)                  //Move backward
{
    analogWrite (E1,a);
    digitalWrite(M1,HIGH);
    analogWrite (E2,b);
    digitalWrite(M2,HIGH);
}
void left (char a,char b)                      //Turn Left
{
    analogWrite (E1,a);
    digitalWrite(M1,LOW);
    analogWrite (E2,b);
    digitalWrite(M2,HIGH);
}
void right (char a,char b)                     //Turn Right
{
    analogWrite (E1,a);
    digitalWrite(M1,HIGH);
    analogWrite (E2,b);
    digitalWrite(M2,LOW);
}
void setup(void)
{
    int i;
    for(i=4;i<=7;i++)
        pinMode(i, OUTPUT);
    Serial.begin(19200);      //Baud Rate of 19200
    pinMode(LDR, INPUT); // LDR as input
}
void loop(void)
{
    if(Serial.available()){
        char val = Serial.read();
        if(val != -1)
        {
            switch(val)
            {
                case 'w':
                    forward (255,255);    //move forward in max speed
                    break;
                case 's':
                    backward (255,255);    //move back in max speed
                    break;
                case 'a':
```

```

        left (255,255); //turn left in max speed
        break;
    case 'd':
        right (255,255); //turn right in max speed
        break;
    case 'x':
        stop();
        break;
    case 'o':
        forward(125,125);
        delay(2000); //move forward 2s
        stop();
        break;
    case 'i':
        left(255,255);
        delay(1500); //move left 1.5s
        stop();
        break;
    case 'p':
        right(255,255);
        delay(1500); //move right 1.5s
        stop();
        break;
    case 'l':
        backward(125,125);
        delay(2000); //move backward 1.5s
        stop();
        break;
    case 'b':
        sensorValue = analogRead(LDR); //read the sensor
        if(sensorValue>89) //There is light
        {
            Serial.write("t");
        }
        else
        {
            Serial.write("f");
        }
    }
}
else stop();
}
}

```

Appendix 6. Hostpad configuration file

```
ctrl_interface=/var/run/hostapd  
macaddr_acl=0 auth_algs=1  
driver=nl80211  
interface=wlan0  
ssid=RPi  
hw_mode=g  
ieee80211n=1  
channel=1  
auth_algs=1  
wmm_enabled=0
```

Appendix 7. Survey done to children

1. Understand how it works without reading the rules?☐ YES☐ NO**2. Understand the rules?**☐ YES☐ NO**3. Play without troubles?**☐ YES☐ NO**4. Like the device?**☐ YES☐ NO**5. Is there any problem with the device?**☐ YES☐ NO