INCREASING THE CAPABILITIES OF MULTIROTOR AERIAL ROBOTS WITH THE RASPBERRY PI PLATFORM

Maria ZHEKOVA
University of Luxembourg
Email: maria.zhekova.001@student.uni.lu

Jose Luis SANCHEZ LOPEZ
University of Luxembourg
Email: joseluis.sanchezlopez@uni.lu

Abstract

Over the last few decades, a large variety of robots have been introduced to numerous applications, tasks and markets. This is a main part of the focus in the research field of Human Robot Interaction. In the upcoming field of humanoid and human-friendly robots, the ability of the robot for simple, unconstrained and natural interaction with its users is of central importance. The basis for appropriate action of the robot must be a comprehensive model of the current surrounding and in particular of the humans involved in interaction.

In this paper, we present an extend of the capabilities of a multirotor aerial robots with the Raspberry Pi platform with the use of a given template for the creation of the report [1].

1. Introduction

As robots move into our lives, the importance of enabling users to interact with them in a natural way increases.

Human-Robot Interaction (HRI) is a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with humans. Interaction, by definition, requires communication between robots and humans [2]. That kind of communication may take several forms, but these forms are largely influenced by whether the human and the robot are in close proximity to each other or not. Thus, communication and, therefore, interaction can be separated into two general categories:

- Remote interaction – The human and the robot are not co-located and are separated spatially or even temporally (for example, the Mars Rovers are separated from Earth both in space and time).
- Proximate interactions – The humans and the robots are co-located (for example, service robots may be in the same room as humans).

Within these general categories, it is useful to distinguish between applications that require mobility, physical manipulation, or social interaction. Remote interaction with mobile robots often is referred to as teleoperation or supervisory control, and remote interaction with a physical manipulator is often referred to as telemanipulation. Proximate interaction with mobile robots may take the form of a robot assistant, and proximate interaction may include a physical interaction. Social interaction includes social, emotive, and cognitive aspects of interaction.

2. Project description

The main objective of this Bachelor Semester Project is to increase the capabilities of a multirotor aerial robot with the help of the Raspberry Pi platform in order to create a natural user interface. Finally, the project can be evaluated with a real aerial robot.

2.1. Domains

In this section the scientific and technical domain of this project will be explained.

2.1.1. Scientific

The scientific aspect covered by this Bachelor Semester Project is to extend the capabilities of a multirotor aerial robot such as enhancing its localization and communication or with other words: providing HRI skills.

The natural interaction between humans and machines has become an important topic for the robotic community as it can generalize the use of robots [3]. However, it has become increasingly apparent that social and interactive skills are necessary requirements in many application areas and contexts where multirotor aerial robots need to interact and collaborate with other robots or humans. The research on human–robot interaction poses many challenges regarding the nature of interactivity and 'social behavior' in robots and humans.

2.1.2. Technical

The technological aspects covered by this project are to select the appropriate hardware components, to install and to configure them on a Raspberry Pi platform and to develop the required programs in order to interact with them. Raspberry Pi is the name of a series of single-board computers which are very suitable platforms for this kind of robots due to its reduced cost, small size and weight, and hardware interaction possibilities.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables

The scientific contribution of the project is the development of tools that provides the aerial robot with natural human-robot interaction capabilities. Therefore, the scientific deliverable is the capability of the aerial robot to have natural human-robot interactions.

We needed to choose and design the most convenient natural interfaces for our project suitable to accomplish the natural HRI. We produced them and implemented them as a technical contribution.

Before choosing the most convenient interfaces, we need to understand how the Raspberry Pi operate in order to see which interface will be compatible with it.

After some research we chose to have several Light Emitting Diodes (LEDs), an ultrasonic sensor, a camera and a servo motor (or simply called servo).

2.2.2. Technical deliverables

The targeted technical deliverables are to deliver a Raspberry Pi platform equipped with the chosen natural interfaces.

We connect the Raspberry Pi to a computer so that the user can receive all the information from the distance sensor and the camera and then to give back an order, in our case to rotate the connected servo if it is convenient for the user. In addition, we have installed different colors of LEDs which lights up, depending on the measured distance.

If we provide a multirotor aerial vehicle with the components mentioned in section *2.2.1* we then can obtain a multirotor aerial vehicle able to detect if there are objects near it, transfer a real time video to the user and open or close its door if it is a delivery drone.

The connection between the Raspberry Pi and the user's computer was meant to be via 4G connection. This type of connection needed to be established with a 4G/LTE module, a 3G-4G/LTE base shield, antennas and a sim card. The connection to the internet became impossible and after a lot of research the reason was not found. In this case we needed to change this part and instead of a 4G connection we used a wireless connection.

For the communication between the Raspberry Pi and the computer we used ROS (Robot Operating System) for which will be explained in section *4.1*.

3. Background

3.1. Scientific background

As an important scientific knowledge, it is required to understand the natural human-robot interaction in order to be able to select and explore the most interesting sensors and actuators to achieve a successful interaction. The ability of a social robot to interact with humans, emerges from a coherent exploitation of social skills that explicitly take into account the human presence. Identifying users, understanding their actions, inferring their mental and emotional states, processing non-verbal gestures are just some examples of social skills that are pushing interactive agents towards an efficient and practical collaboration with humans in different scenarios; as partners in industries, as companions for children and elderly, as educational tools, as assistants in public or personal spaces and so on. However, despite progress in the field of HRI, today's state-of-the-art agents still lack skills in accomplishing complex social tasks and are yet unable to function autonomously in real-world social scenarios.

HRI is of critical importance in the entertainment robotics sector. In order to produce a desirable end product that can be enjoyed over extended

periods of time, it is essential that an understanding of not only robotics, but also human psychology be brought to bear.
A robotic being is perceived as an embedded part of human relations with the world, as one of the components of the interactive process but not the separate system. To be relevant to the unpredictable psychological reality of the human, the organization of a robotic creature's behavior needs to be built with the motor and sensory loops running in parallel rather than employ hierarchically structured functioning.
Robots' capacity to adjust to the requirements of the human world have its origins from the concept of artificial intelligence, or a set of programs that control the automatic actions of machines.
The field of HRI is supported by progress in computing, visual recognition, and wireless connectivity, which open the door to a new generation of mobile robotic devices that see, hear, touch, manipulate, and interact with humans.
As a scientific discipline, computer vision is concerned with the theory and technology for building artificial systems that obtain information from images or multi-dimensional data.
A significant part of artificial intelligence deals with planning or deliberation for system which can perform mechanical actions such as moving a robot through some environment.
This type of processing typically needs input data provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot.
Other parts which sometimes are described as belonging to artificial intelligence and which are used in relation to computer vision is pattern recognition and learning techniques.

3.2. Technical background

The technological aspects that are covered by this project consist in the implementation of a communication via ROS between the user's computer and the Raspberry Pi platform and the implementation of the circuit for the interfaces.
Since we will be working with Python, it was important to have knowledge of it.

4. A Scientific Deliverable 1

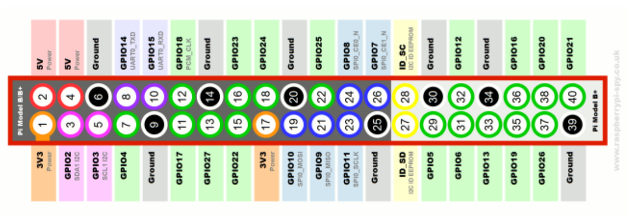
4.1. Requirements

The requirement that satisfies the scientific deliverable is the understanding the functionality of the components and ROS.
In case we have a simple multirotor aerial vehicle and we want to upgrade it into a more functional and more capable to interact with human robot, we then need to add several interfaces.
Let us say, that our wish of a drone is a delivery drone. This drone needs to have a door to open or close in order to deliver. We can do that with the help of a servo.
The drone then needs to be aware of its surroundings. A good way to do that is to add a camera which transfers a life-time video to the user of the drone. Much better will be if it can also detect for his safety if there is an object nearby and then to send back the information to the user. For a better view of the measured distance we can add three LEDs:

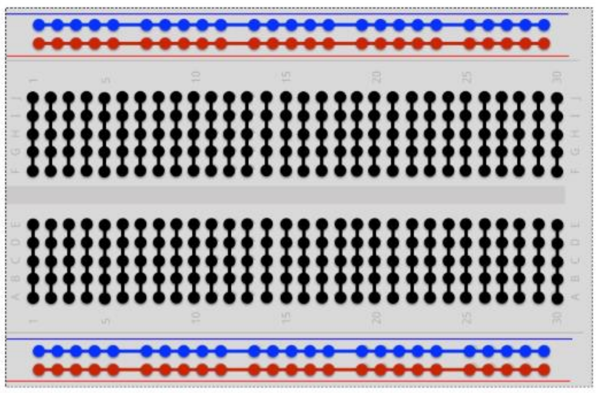
- A red one – which glows when the distance is less than 20cm
- A yellow one – which glows when the distance is between 20cm and 40cm
- A green one – which glows when the distance is more than 40cm

In our project we chose to command the interfaces with Raspberry Pi because it is very good at physical computing, which in this context means programming and interacting with the real world through electronics.
The circuits can be simple or very complex and are made up of electronic components such as LEDs, buzzers, buttons, resistors, capacitors, and even integrated circuit (IC) chips. Some components can be used any way round, such as the resistor or switch. However, others have a specific orientation, such as the LED. Diodes only let electricity flow from positive to negative.
The Raspberry Pi can provide power to a circuit, as well as a negative or ground end through its GPIO pins, GPIO stands for General Purpose Input Output. Some pins are specifically always powered, mostly by 3.3V, and always go to ground. Most of them can be programmed to create or recognize a HIGH or LOW signal, in the case of the Raspberry Pi, a HIGH signal is 3.3V

and a LOW signal is ground or 0V, as shown in the image below. [5]



In order to create the physical circuits, we are using a breadboard. The breadboard allows us to insert components and wires to connect them, without soldering them together. Electrical components are connected by pushing them into the holes in a breadboard. These holes are connected in strips, as shown on the image below. [4]



If we push a wire, or a different component, into one hole in a strip, and another wire into the hole next to it, it's as if we had physically joined (or soldered) the two wires.
Once the components are all hooked up to the breadboard or directly to the Raspberry Pi, we need to be able to control them. In this project we programmed with the Python language.
In our project we used the Raspberry Pi 3 Model B, this model is the third generation Raspberry Pi. It is a powerful credit-card sized single board computer which can be used for many applications. The processor is powerful and additionally adds wireless LAN and Bluetooth connectivity, which are making it the ideal solution for powerful connected designs.

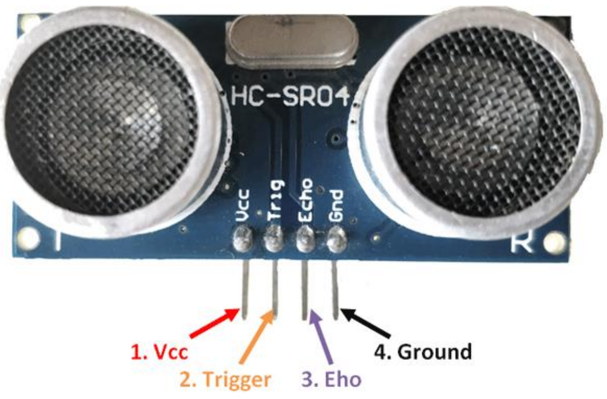
ROS is a flexible framework for writing robot software using the concept of an Operating System (OS). It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. Over the years, ROS has become the essential tool for roboticists. A large community surrounds ROS and there has been extensive input from industrial users in the development of these tools. [6]

4.2. Design

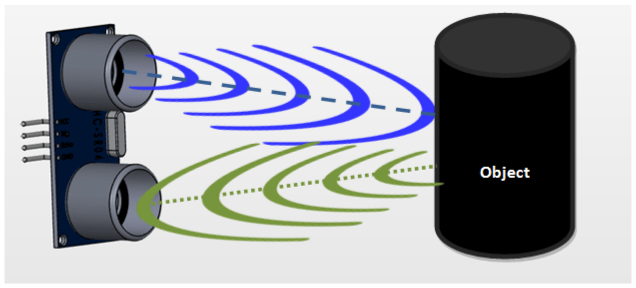
Here we will explain our understandings for the different interfaces and ROS.

The ultrasonic sensor (HC-SR04):

It is a 4-pin module, whose pin names are Vcc, Trigger, Echo and Ground as shown in the image below.



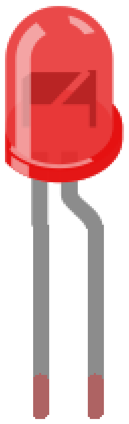
This sensor is a very popular sensor used in many applications where measuring distance or sensing objects are required. The module has two eyes like projects in the front which forms the Ultrasonic transmitter and Receiver. The Ultrasonic transmitter transmits an ultrasonic wave, this wave travels in air and when it gets objected by any material it gets reflected back toward the sensor this reflected wave is observed by the Ultrasonic receiver module as we can see below.



The distance can be calculated with the following formula: $L = ½*T*C$, where L is the distance, T is the time between the emission and reception, and C is the sonic speed. The value is multiplied by *1/2* because *T* is the time for go-and-return distance. [7]

The LED:

It has two legs, as shown on the figure, with one leg shorter than the other.



The longer leg, known as the anode is always connected to the positive supply of the circuit and the shorter leg, known as the cathode is connected to the negative side of the power supply, known as ground.

LEDs only work if power is supplied the correct way around i.e. if the polarity is correct. If they are connected the wrong way around they won't break, they will just not light.

Having a resistor is very important to connect the LED up to the GPIO pins of the Raspberry Pi, because the Raspberry Pi can only supply a small current, about 60mA. If more current is drawn it will burn out the Raspberry Pi, having resistors ensure that only a small current will flow, and the Raspberry Pi will not be damaged. [8]

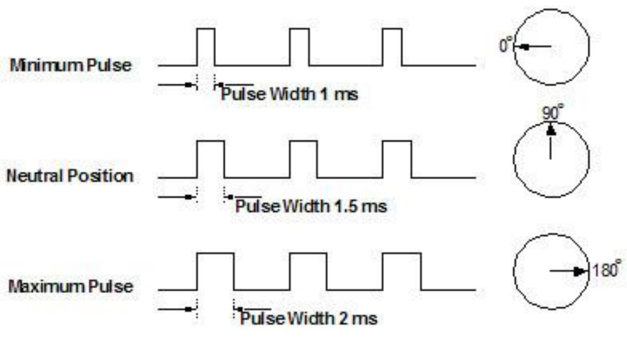
The servo:

Servos are controlled by sending them a pulse of variable width. The control wire is used to send this pulse. The parameters for this pulse are that it has a minimum pulse, a maximum pulse, and a repetition rate. Given the rotation constraints of the servo, neutral is defined to be the position where the servo has exactly the same amount of potential rotation in the clockwise direction as it does in the counter clockwise direction.

It can rotate itself to any angle from 0-180 degrees. Its 90-degree position is generally referred to as 'neutral' position, because it can rotate equally in either direction from that point.

When these servos are commanded to move they will move to the position and hold that position. If an external force pushes against the servo while the servo is holding a position, the servo will resist from moving out of that position. The maximum amount of force the servo can exert is the torque rating of the servo. Servos will not hold their position forever; the position pulse must be repeated to instruct the servo to stay in position.

We can see below the different rotation that we can obtain when sending a particular pulse. [9]



The camera:

The Raspberry Pi Camera v2 is the new official camera board released by the Raspberry Pi Foundation.

The Raspberry Pi Camera Module v2 is a high quality 8-megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi, featuring a fixed focus lens. [10]

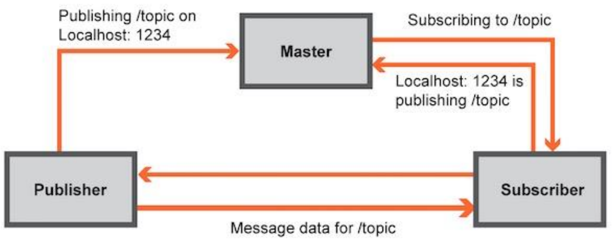
ROS:

In general, ROS consists of code and tools that help the code run and do the required job—including the infrastructure for running it, like messages passing between processes.

ROS is designed to be a loosely coupled system where a process is called a node and every node should be responsible for one task. Nodes communicate with each other using messages passing via logical channels called topics. Each node can send or get data from the other node using the publish/subscribe model.

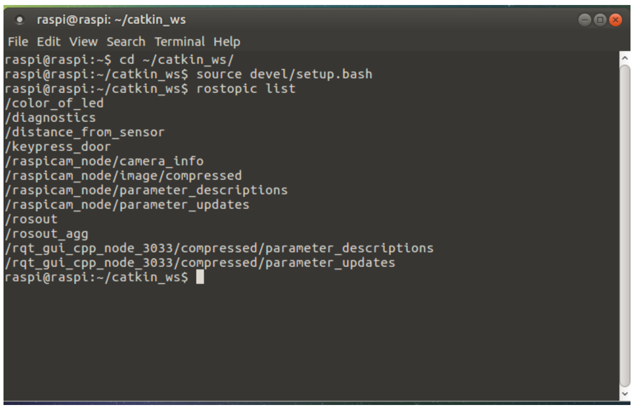
In order to manage the environment, there is a Master in ROS which is responsible for name registration and lookup for the rest of the system. Without the Master, nodes would not be able to find each other or exchange messages. To start the Master, we need to use one of the two following commands:

roscore or *roslaunch*, in order to obtain a connection like the one shown below.


Publishing /topic on Localhost: 1234
Master
Subscribing to /topic
Localhost: 1234 is publishing /topic
Publisher
Subscriber
Message data for /topic

Messages are structures of data filled with pieces of information by nodes. Nodes exchange them using what's called topics (logical connection paths), then nodes either publish topics or subscribe to them. [11]

Respectively, *rostopic list* command-line tool can be used to list topics as we can see on the screenshot below. We used it to see all the working topics for out project.



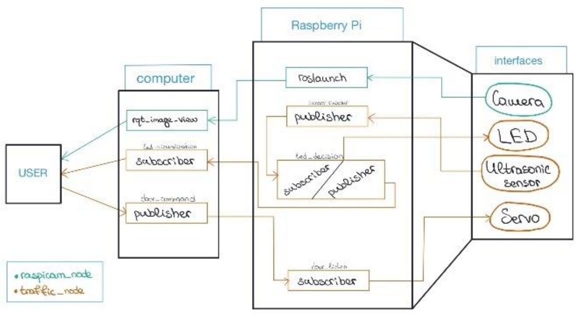
```
raspi@raspi:~$ cd ~/catkin_ws/
raspi@raspi:~/catkin_ws$ source devel/setup.bash
raspi@raspi:~/catkin_ws$ rostopic list
/color_of_led
/diagnostics
/distance_from_sensor
/keypress_door
/raspicam_node/camera_info
/raspicam_node/image/compressed
/raspicam_node/parameter_descriptions
/raspicam_node/parameter_updates
/rosout
/rosout_agg
/rqt_gui_cpp_node_3033/compressed/parameter_descriptions
/rqt_gui_cpp_node_3033/compressed/parameter_updates
raspi@raspi:~/catkin_ws$ 
```

Let us say we want to simulate the data transfer between an ultrasonic sensor and a program that commands the direction of the robot. An ultrasonic sensor such as HC-SR04 returns information as a numeric string. The string of numbers returned by the sensor will be read by the Subscriber node for the calculation and decision-making of the robot navigation.
To simulate such a simple scenario for sending and reading data we create two nodes. The Publisher node will generate the number from the sensor, while the Subscriber node will display the number received from the Publisher node.
ROS is a distributed computing environment. A running ROS system can comprise dozens, even hundreds of nodes, spread across multiple machines. Depending on how the system is configured, any node may need to communicate with any other node, at any time.

As a result, ROS has certain requirements of the network configuration:

- There must be complete, bi-directional connectivity between all pairs of machines, on all ports.
- Each machine must advertise itself by a name that all other machines can resolve.

For this project we work on two machines: the Raspberry Pi and the computer. We need only one *roscore* running and it will be on the Raspberry Pi, since it is the Raspberry Pi which is connected to the interfaces.
Below we can see the designed communication paths for this project.



4.3. Production

The philosophy of ROS is that we can have many individual programs on our machines which communicate over defined API with the help of ROS messages, services etc.

We use two big nodes: one for the camera and the other for the LEDs, the distance sensor and the servo.
The node for the camera: *raspicam_node* was extracted from a source [12]. Therefore, it was already created and ready for use.

The other node: *traffic_node*, was entirely created.
This node contains the connection between the LEDs, the ultrasonic sensor and the servo.
As we can see on the image in section *4.2*, the user can manipulate the rotation of the servo if needed, but it cannot manipulate which LED will be turned on. The LED is illuminated depending of the measured distance by the ultrasonic sensor.
The distance from the sensor is communicated to the user through the Raspberry Pi with the help of two topics for which we are going to describe in detail in section *5*.
The communication is established with the following nodes:

- A publisher: ***sensor_reader.py*** – it publishes the received distance calculated from the distance sensor.
- A subscriber and publisher: ***led_decision.py*** – it receives the distance, uses a code to turn on a LED and publishes a phrase with the color of the illuminated LED and the measured distance.
- A subscriber: ***led_visualization.py*** – receives the given information about the color of the illuminated LED and the distance. This subscriber is started from the user in order to see the information of the interfaces.

The third topic is for the servo with the following nodes:

- A publisher: ***door_command.py*** – this script is running by the user on its computer and is sending a command to the Raspberry Pi to rotate the servo if needed.
- A subscriber: ***door_listen.py*** – it receives the given command and rotates the servo.

4.4. Assessment

Since the requirement for the scientific deliverable was to understand the functionality of the chosen interfaces and ROS. The scientific deliverable of explanation of the functionality satisfies fully the requirement.

5. A Technical Deliverable 1

5.1. Requirements

The main required competences necessary to work will be to have at least a basic knowledge as a starting point of electronics, digital and analog sensors and actuators in order to properly select and mount the required hardware and to be able to connect the sensors and actuators to the Raspberry Pi platform. It is required to have the knowledge of Linux system administration to configure and install the drivers of the sensors and actuators. It is highly recommended to have knowledge of programming in order to develop the programs for the use of components. In this project, we will use Python as a program language.

For our project we need to create a compatible and functional circuit for all the interfaces we are using. In order to obtain a functional circuit, except a good wiring we need to create a script which will control the components.

As we mentioned before, in order to establish a connection through ROS, we need to have an internet connection. This connection was meant to be a 4G connection with the help of two modems mentioned in section *2.2.2*.

In our circuit it is really important to use resistors for the LEDs and the ultrasonic sensor. Resistors are a way of limiting the amount of electricity going through a circuit; specifically, they limit the amount of ‘current’ that is allowed to flow. The measure of resistance is called the Ohm (Ω), and the larger the resistance, the more it limits the current. The value of a resistor is marked with colored bands along the length of the resistor body.

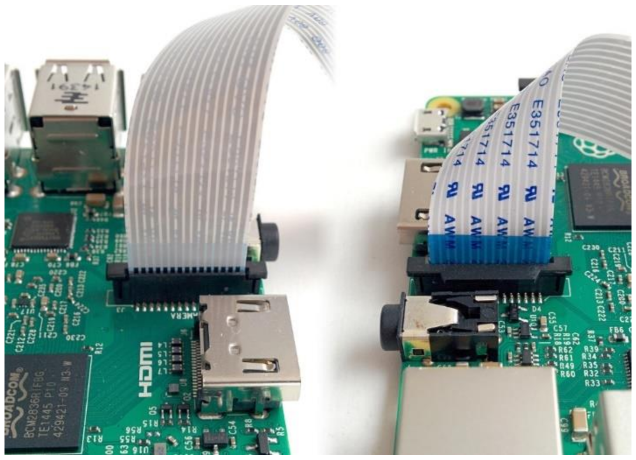
We are be also using jumper wires to connect the elements and to ‘jump’ form one connection to another.

5.2. Design

Before starting with the script, we needed to design the circuit for the project.

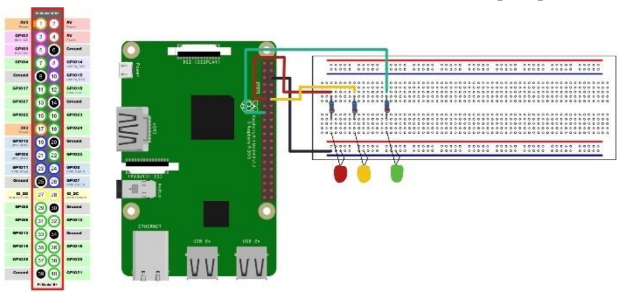
Here, will be explained the connection of the Raspberry Pi with each interface separately, because it is easy to view, due to many mixed wires.

The camera module can be easily connected to the Raspberry Pi’s camera port as shown on the photo below. [12]



On each image we can see next to the Raspberry Pi a visualisation its pins.

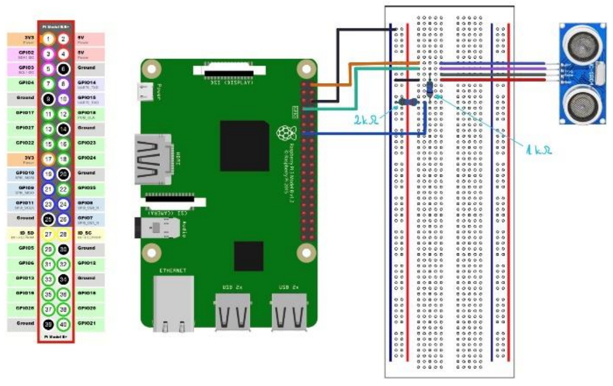
➢ First, we started with the LEDs [13]:



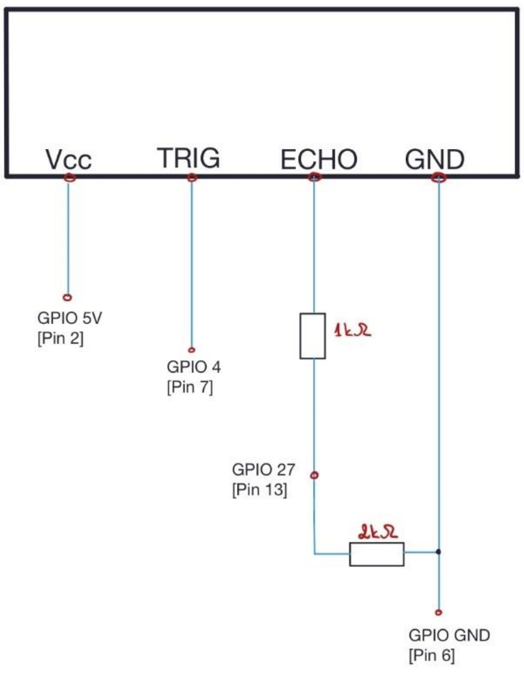
We use a jumper wire to connect a ground pin (GND) to the rail, marked with blue.
The positive side of the LED is connected to the resistance and the negative to the rail of the GND. The resistors we are using are of 330kΩ.
Then they are connected to GPIO pins:

- The one for the red LED is to GPIO 17
- The one for the yellow LED is to GPIO 18
- The one for the green LED is to GPIO 22

➢ Then we added the ultrasonic sensor [14]:

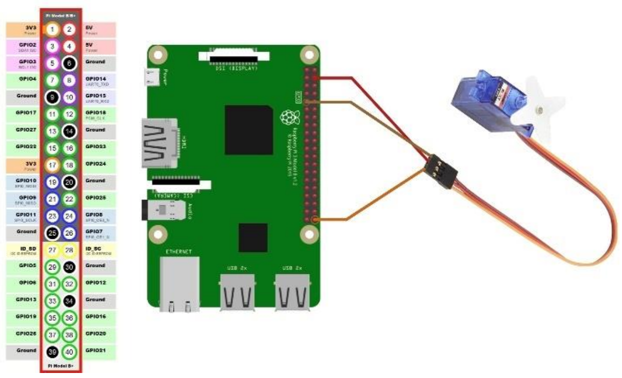


As we can see on the image we worked with two different resistors: 1kΩ and 2kΩ.
The GND is still connected to the breadboard and the GPIO pins mentioned before are in use of the LEDs.
We can see clearly on the following schema how it is connected:



A voltage divider consists of two resistors in series connected to an input voltage (ECHO), which needs to be reduced to our output voltage from 5V to 3.3V.

➢ Finally, we need to add the servo to the circuit [15]:



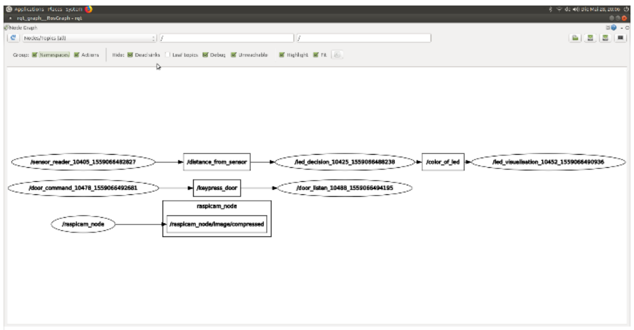
To connect the servo to the Raspberry Pi we do not need resistors, we can simply connect it to the GPIO pins. It has three wires: power, ground and signal. In our case the power wire is the red one and should be connected to the 5V pin [Pin 4] on the Raspberry Pi. The ground wire is brown and should be connected to a ground pin [Pin 9]. The signal pin is orange and should be connected to a digital pin, in our case to GPIO21 [Pin 40].

After creating the circuit, we need to design the ROS workspace:

```
|-- catkin_ws (folder)
|--|-- build (folder) [created by catkin_make]
|--|--|-- ...
|--|-- devel (folder) [created by catkin_make]
|--|--|-- ...
|--|-- src (folder)
|--|-- |-- zhekova_bsp2 (folder)
|--|-- |-- |-- doc (folder)
|--|-- |-- |-- |-- info.txt
|--|-- |-- |-- stack (folder)
|--|-- |-- |-- |-- raspicam_node (folder) [took from
GitHub]
|--|-- |-- |-- |-- |-- ...
|--|-- |-- |-- |-- traffic_node (folder)
|--|-- |-- |-- |-- |-- scripts (folder)
|--|-- |-- |-- |-- |-- |-- sensor_reader.py
|--|-- |-- |-- |-- |-- |-- distance.py
|--|-- |-- |-- |-- |-- |-- led_decision.py
|--|-- |-- |-- |-- |-- |-- led.py
|--|-- |-- |-- |-- |-- |-- led_visualisation.py
|--|-- |-- |-- |-- |-- |-- door_command.py
|--|-- |-- |-- |-- |-- |-- door_listen.py
|--|-- |-- |-- |-- |-- |-- servo.py
|--|-- |-- |-- |-- |-- |-- openDoor.py
|--|-- |-- |-- |-- |-- |-- closeDoor.py
|--|-- |-- |-- |-- |-- CMakeLists.txt
|--|-- |-- |-- |-- |-- package.xml
|--|-- |-- |-- LICENSE
|--|-- |-- |-- README.md
|--|-- |-- CMakeLists.txt
```

The folder *build* and *devel* are created due to the command *catkin_make*, for which will be explained in section *5.3*.

We can see clearly on the following screenshot how the communication between the nodes in ROS should be established through topics.

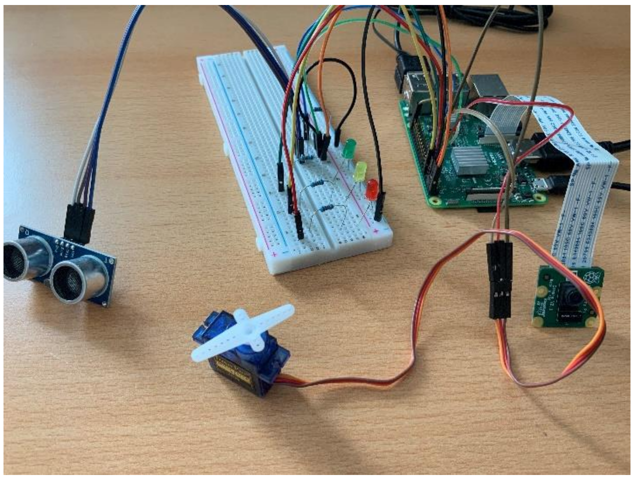

Nodes/topics (all)
Group: Namespaces Actions
Hide: Dead sinks Leaf topics Debug Unreachable
Highlight Fit
/sensor_reader_10405_1559066482827
/distance_from_sensor
/led_decision_10425_1559066488238
/color_of_led
/led_visualisation_10452_1559066490936
/door_command_10478_1559066492681
/keypress_door
/door_listen_10488_1559066494195
raspicam_node
/raspicam_node
/raspicam_node/image/compressed

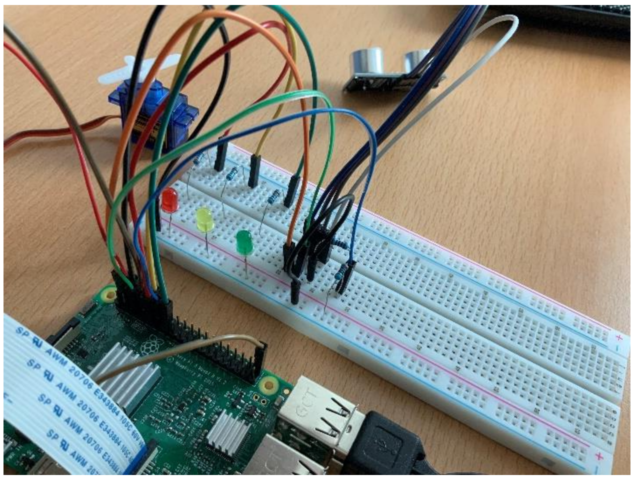
5.3. Production

At the beginning of the project we needed to choose the best Raspberry Pi model, a way to have a 4G connection and the most interesting components to add on the platform. After finding and ordering all the chosen components we trained with some LEDs and an ultrasonic sensor till all the elements arrive. The elements for the 4G connection arrived, but the connection could not be established due to unknown problem. Then we decided to have a wireless connection.

Before we can start the configurations and the construction of the circuits we needed to install an operation system which supported the Raspberry Pi. The chosen operation system was Ubuntu MATE, its image fit on a minimum of 4GB micro SD card. Its image was downloaded from the website of Raspberry Pi, then the image was written on the card with the application balenaEtcher. After that, we needed to configure system locales and make an update of the system.

We connected the components as explained in section *5.2* in order to obtain a well-constructed circuit:





For using ROS, we need to install it by following the tutorial from *wiki.ros.org*

Before creating the nodes for our project, we needed to create a workspace for ROS. Our workspace is a created folder: *catkin_ws/src/*. After creating the workspace, we need to run `catkin_make`, which will create a `CMakeLists.txt` link to the *src* folder. Additionally, it will create a *build* and *devel* folder. The *devel* space contains only artefacts generated by a build step.

In every terminal we open to use ROS we need to setup the network by typing the following commands:

On the Raspberry Pi:

- When in the ROS workspace: `$ source devel/setup.bash`
- `$ export ROS_HOSTNAME= raspberryIP`
- `$ export ROS_MASTER_URI=http://raspberryIP:11311/`
- `$ export ROS_IP=raspberryIP`

Where the raspberryIP is the IP of the Raspberry Pi.

On the user's computer:

- `$ export ROS_HOSTNAME= pcIP`
- `$ export ROS_MASTER_URI=http://pcIP:11311/`
- `$ export ROS_IP=pcIP`

Where the pcIP is the IP of the computer.

Now we will provide an explanation for the production of the scripts. The entire scripts can be seen in section *7*.

➢ For the **sensor_distance.py**:

Firstly, we created a script with a function which controls the ultrasonic sensor: *distance.py*. While measuring the time, the script is sending the pulse signal and sets the GPIO pin, which is connected to the ECHO, to high for the amount of time it takes for the pulse to go and come back. Then with the measured time we are able to calculate the distance between the sensor and an object in front of it. This function returns the measured distance, which we use for our first publisher node: *sensor_reader*.

In every Python node we have a declaration at the top:

```
#!/usr/bin/env python
```

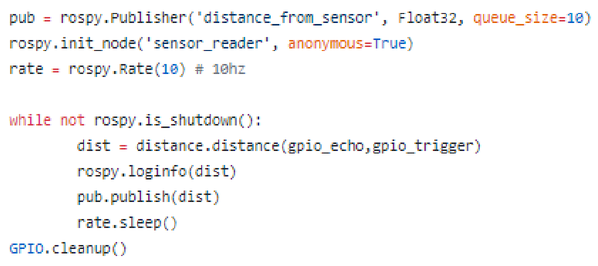
This line makes sure the script is executed as a Python script.

Since we are writing a ROS node we need to import *rospy* and the type of the message we are publishing all together with the libraries and scripts we are using:

```
import rospy
import distance
from std_msgs.msg import Float32
import Rpi.GPIO as GPIO
```

Then we initialize some variables and properties used in the *distance.py* script regarding the ultrasonic sensor.

To declare the topic on which the node is publishing we use:



```
pub = rospy.Publisher('distance_from_sensor', Float32, queue_size=10)
rospy.init_node('sensor_reader', anonymous=True)
rate = rospy.Rate(10) # 10hz

while not rospy.is_shutdown():
        dist = distance.distance(gpio_echo,gpio_trigger)
        rospy.loginfo(dist)
        pub.publish(dist)
        rate.sleep()
GPIO.cleanup()
```

The initialization of `pub` is declaring that the node is publishing to the *distance_from_sensor* topic using the message of type *Float32* and with the help of `rospy.init_node` we tell `rospy` the name of the node in order to communicate with ROS Master.

The initialization of `rate` helps to have a convenient way for looping at the desired rate.

Then we enter the while loop only if `rospy` is working, we use the script of the sensor to put the measured distance in the variable `dist` and then we publish the number to the *distance_from_sensor* topic with `pub.publish` command. We also use `rospy.loginfo` to print the message to the screen, to write it on the Node's log file and to *rosout*. The *rosout* is used for debugging, we can pull up messages using `rqt_console` instead of having to find the console window with the Node's output.

To maintain the desired rate, we write `rospy.sleep()` in the end of the loop.

When the `rospy` is shut down it is important to clean all the ports we used with `GPIO.cleanup()`, it resets any ports we have used in this script back to its input mode.
Finally, to execute the code we need:

```
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        rospy.loginfo("Measurement stopped by user.")
```

➢ For the **led_decision.py**:

The second node we created was a node which acts like a subscriber and a publisher. Therefore, this node receives an information, connects with the LED interface and publishes a result to another node.
This multifunctional node is called *led_decision.py* and it uses a script to command the LED which is called *led.py*.
The *led.py* script contains a function which has as parameters the distance received from *sensor_reader.py* and the number of the GPIO pin the LEDs are connected. The function contains three if statements, depending of the received distance, it illuminates the appropriate LED and in the end, it returns a String message with the status of the calculation and the color of the illuminated LED.
This time the subscribing node has two functions:

- **callback(msg)** – initialize the new publishing message received from the *led.py*, publishes to the *color_of_led* topic and prints to the screen the received information from the node *sensor_reader.py*

```
def callback(msg):
        pub = rospy.Publisher('color_of_led', String, queue_size=10)
        rospy.loginfo(rospy.get_caller_id() + " I heard: "+ str(msg.data))
        resp=led.traffic(msg.data,ledRed,ledYellow,ledGreen)
        pub.publish(resp)
```

- **listener_talker()** – sets some properties to the GPIO pins, with `rospy.Subscriber` it subscribes to the *distance_from_sensor* topic and when new messages are received the *callback(msg)* is invoked with the message as the first argument. The final addition `rospy.spin()` is keeping the node from exiting until it has been shut down.

```
def listener_talker():
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)

        GPIO.setup(ledRed, GPIO.OUT)
        GPIO.setup(ledYellow, GPIO.OUT)
        GPIO.setup(ledGreen, GPIO.OUT)

        rospy.init_node('led_decision', anonymous=True)
        rospy.Subscriber('distance_from_sensor', Float32, callback)
    # spin() simply keeps python from exiting until this node is stopped
        rospy.spin()
```

In order to execute the code, we need to add also an if statement:

```
if __name__ == '__main__':
        try:
                listener_talker()
                GPIO.cleanup()

        #when pressing CTR+C
        except KeyboardInterrupt :
                pass
        print("Measurement stopped by user.")
```

➢ For the **led_visualisation.py**:

This is the subscriber which is used by the user. The script has the same functionality as the *led_visualisation.py*, just except the publishing part. When the node is running, we receive the message published by the node *led_decision.py*. Hence, the user receives the status of the distance:

- if it is less than 20cm we have the message:`"--STOP! The distance is "+str(dist)+ " It is red. "`
- if it is between 20cm and 40cm we have the message: `"--ATTENTION! The distance is "+str(dist)+ " It is yellow. "`
- if it is more than 40cm we have the message:`"--You may proceed! The distance is "+str(dist)+" It is green. "`

➢ For the **door_command.py**:

This is another publisher node which is controlled by the user. It uses a keyboard input and sends it to its subscriber through the *keypress_door* topic.
Before the user needs to press a key, the node prints a message with instructions which key is for what: `"Press 1 to open or 0 for close and then enter. "`. The input is initialized with `n = int(raw_input())`, then printed to the

screen and published to the topic. These commands are also in a while loop:

```
while not rospy.is_shutdown():
          print("Press 1 to open or 0 for close and then enter. ")
          n = int(raw_input())
          rospy.loginfo(str(n))
          pub.publish(n)
          rate.sleep()
```

- For the **door_listen.py**:

This node is running on the Raspberry Pi, because it controls the rotation of the servo.

It uses two external scripts: *openDoor* and *closeDoor*. These scripts are very small, and they are just giving a command to the servo without doing any calculation for its rotation.

openDoor.py:

```
import servo

def openDoor(pwm):
        servo.SetAngle(90,pwm)
        print("The door is open")
```

closeDoor.py:

```
import servo

def closeDoor(pwm):
        servo.SetAngle(0,pwm)
        print("The door is closed.")
```

These small scripts are using another script which contains of a function: *SetAngle(..)* which is the one making the calculations and rotating the servo to the desired angle.

The subscriber, itself is deciding with *if* statements whether to turn the servo to 0 degrees or to 90 degrees. The statements are relaying to the received number from the user through the *keypress_door* topic. It is also printing the received message to the screen.

```
def callback(msg):
        pwm = GPIO.PWM(21, 50)
        rospy.loginfo(rospy.get_caller_id() +" "+ str(msg.data))
        if(msg.data==0):
                openDoor.openDoor(pwm)
        elif(msg.data==1):
                closeDoor.closeDoor(pwm)
```

After creating the scripts, we added another node for the camera. The scripts for this node are taken from the internet.

When the

5.4. Assessment

The only element which does not satisfy the requirements mentioned above was the type of internet connection due to administration problems. We adopted quick and found the solution of a wireless connection and ROS.

Acknowledgment

We would like to thank the University of Luxembourg for allowing us to spend a semester working on the project for increasing the capabilities of a multirotor aerial robot.

A special gratitude goes to the project academic tutor, Jose Luis Sanchez Lopez, whose contribution and encouragement, helped to coordinate the project, especially for the communication between the multiple interfaces, and for the help where it was needed more knowledge to gather more information to explain each part of the report.

6. Conclusion

In this project we learned a lot about configuration, since we encountered a lot of different problems, we needed to get used to reinstall, recover a backup and search for the right command, program or version in order to resolve the problem and all this was to be established from the terminal. We learned how we can connect multiple elements to the Raspberry Pi through a breadboard.

Nonetheless, we needed to change the initial idea of the project regarding the internet connection, the project was a success because we could deliver in time functional interfaces connected to the Raspberry Pi.

7. Appendix

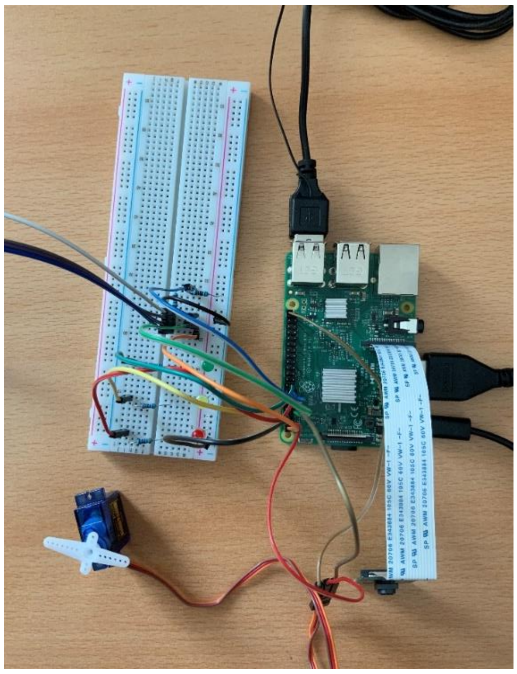
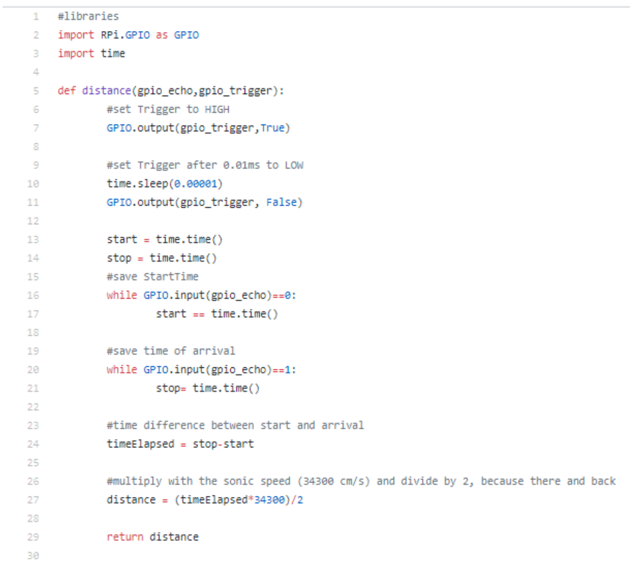


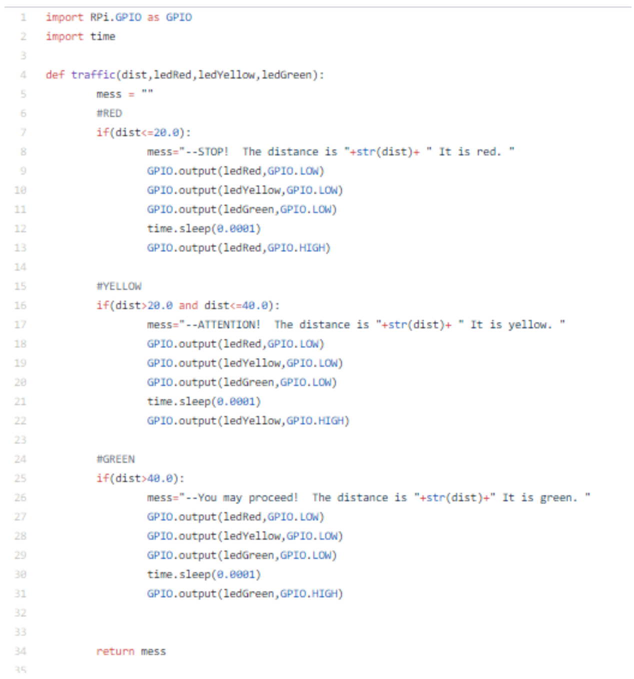
Fig1 – photo of the circuit from above



```python
#libraries
import RPi.GPIO as GPIO
import time

def distance(gpio_echo,gpio_trigger):
        #set Trigger to HIGH
        GPIO.output(gpio_trigger,True)

        #set Trigger after 0.01ms to LOW
        time.sleep(0.00001)
        GPIO.output(gpio_trigger, False)

        start = time.time()
        stop = time.time()
        #save StartTime
        while GPIO.input(gpio_echo)==0:
                start == time.time()

        #save time of arrival
        while GPIO.input(gpio_echo)==1:
                stop= time.time()

        #time difference between start and arrival
        timeElapsed = stop-start

        #multiply with the sonic speed (34300 cm/s) and divide by 2, because there and back
        distance = (timeElapsed*34300)/2

        return distance
```

distance.py

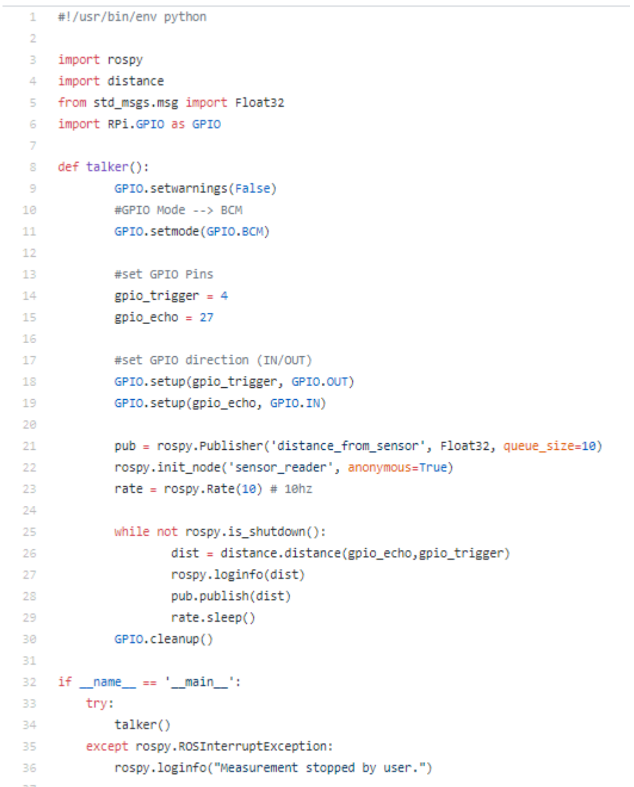
– explanation for the script in section 5.3



```python
import RPi.GPIO as GPIO
import time

def traffic(dist,ledRed,ledYellow,ledGreen):
        mess = ""
        #RED
        if(dist<=20.0):
                mess="--STOP!  The distance is "+str(dist)+ " It is red. "
                GPIO.output(ledRed,GPIO.LOW)
                GPIO.output(ledYellow,GPIO.LOW)
                GPIO.output(ledGreen,GPIO.LOW)
                time.sleep(0.0001)
                GPIO.output(ledRed,GPIO.HIGH)

        #YELLOW
        if(dist>20.0 and dist<=40.0):
                mess="--ATTENTION!  The distance is "+str(dist)+ " It is yellow. "
                GPIO.output(ledRed,GPIO.LOW)
                GPIO.output(ledYellow,GPIO.LOW)
                GPIO.output(ledGreen,GPIO.LOW)
                time.sleep(0.0001)
                GPIO.output(ledYellow,GPIO.HIGH)

        #GREEN
        if(dist>40.0):
                mess="--You may proceed!  The distance is "+str(dist)+" It is green. "
                GPIO.output(ledRed,GPIO.LOW)
                GPIO.output(ledYellow,GPIO.LOW)
                GPIO.output(ledGreen,GPIO.LOW)
                time.sleep(0.0001)
                GPIO.output(ledGreen,GPIO.HIGH)


        return mess
```

led.py

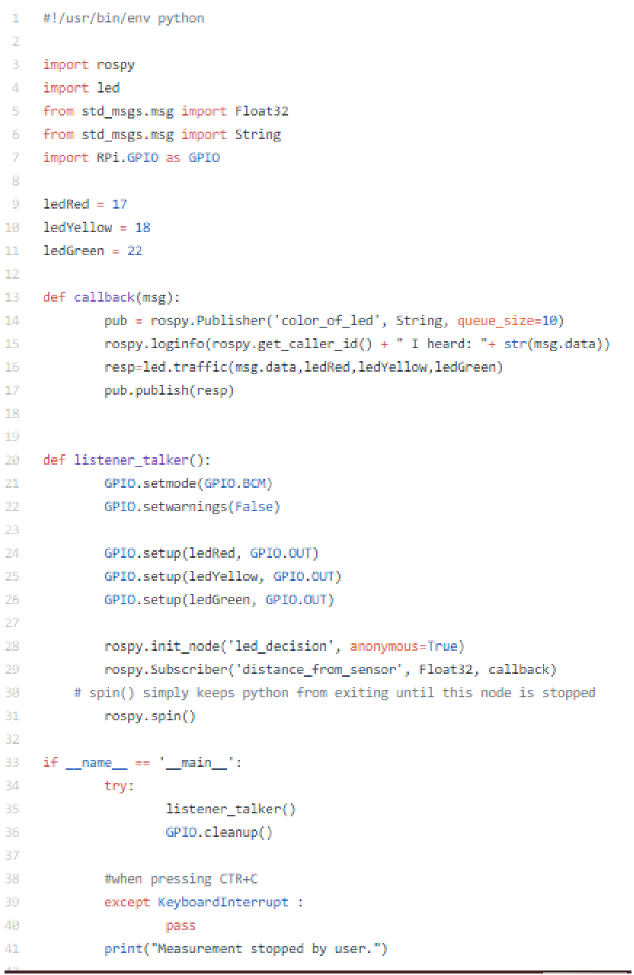
- explanation for the script in section 5.3



```python
#!/usr/bin/env python

import rospy
import distance
from std_msgs.msg import Float32
import RPi.GPIO as GPIO

def talker():
        GPIO.setwarnings(False)
        #GPIO Mode --> BCM
        GPIO.setmode(GPIO.BCM)

        #set GPIO Pins
        gpio_trigger = 4
        gpio_echo = 27

        #set GPIO direction (IN/OUT)
        GPIO.setup(gpio_trigger, GPIO.OUT)
        GPIO.setup(gpio_echo, GPIO.IN)

        pub = rospy.Publisher('distance_from_sensor', Float32, queue_size=10)
        rospy.init_node('sensor_reader', anonymous=True)
        rate = rospy.Rate(10) # 10hz

        while not rospy.is_shutdown():
                dist = distance.distance(gpio_echo,gpio_trigger)
                rospy.loginfo(dist)
                pub.publish(dist)
                rate.sleep()
        GPIO.cleanup()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        rospy.loginfo("Measurement stopped by user.")
```

sensor_reader.py

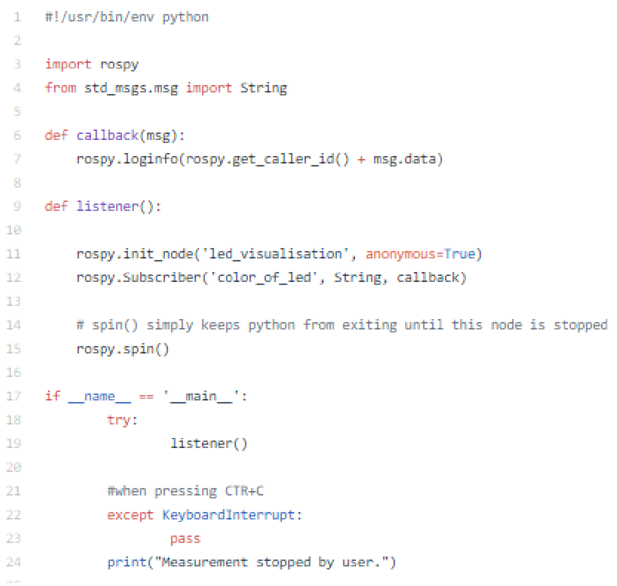
- explanation for the script in section 5.3



```python
#!/usr/bin/env python

import rospy
import led
from std_msgs.msg import Float32
from std_msgs.msg import String
import RPi.GPIO as GPIO

ledRed = 17
ledYellow = 18
ledGreen = 22

def callback(msg):
        pub = rospy.Publisher('color_of_led', String, queue_size=10)
        rospy.loginfo(rospy.get_caller_id() + " I heard: "+ str(msg.data))
        resp=led.traffic(msg.data,ledRed,ledYellow,ledGreen)
        pub.publish(resp)


def listener_talker():
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)

        GPIO.setup(ledRed, GPIO.OUT)
        GPIO.setup(ledYellow, GPIO.OUT)
        GPIO.setup(ledGreen, GPIO.OUT)

        rospy.init_node('led_decision', anonymous=True)
        rospy.Subscriber('distance_from_sensor', Float32, callback)
    # spin() simply keeps python from exiting until this node is stopped
        rospy.spin()

if __name__ == '__main__':
        try:
                listener_talker()
                GPIO.cleanup()

        #when pressing CTR+C
        except KeyboardInterrupt :
                pass
        print("Measurement stopped by user.")
```

led_decision.py

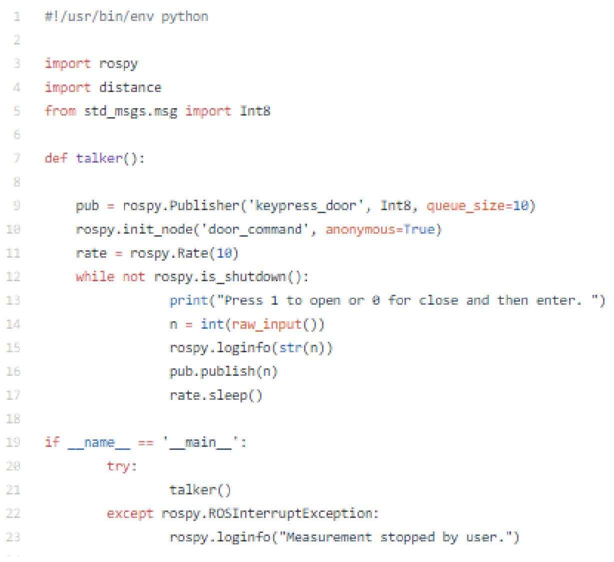
- explanation for the script in section 5.3



```python
#!/usr/bin/env python

import rospy
from std_msgs.msg import String

def callback(msg):
    rospy.loginfo(rospy.get_caller_id() + msg.data)

def listener():

    rospy.init_node('led_visualisation', anonymous=True)
    rospy.Subscriber('color_of_led', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
        try:
                listener()

        #when pressing CTR+C
        except KeyboardInterrupt:
                pass
        print("Measurement stopped by user.")
```

led_visualisation.py

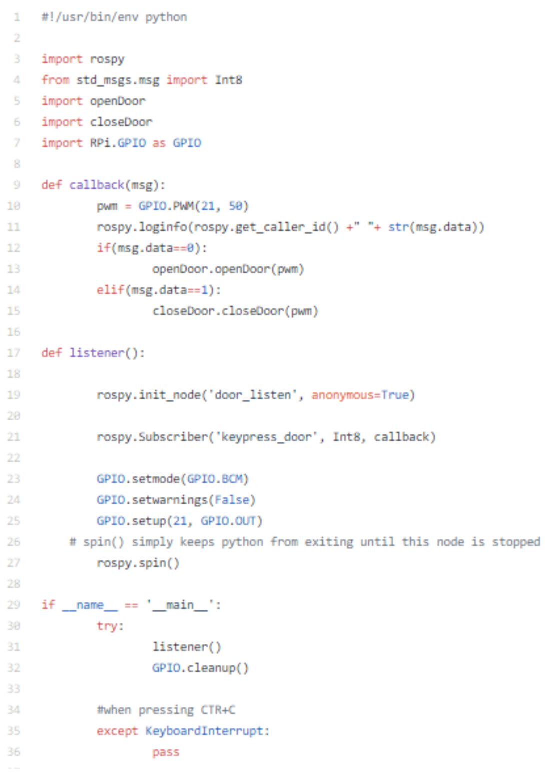
- *explanation for the script in section 5.3*



```python
#!/usr/bin/env python

import rospy
import distance
from std_msgs.msg import Int8

def talker():

    pub = rospy.Publisher('keypress_door', Int8, queue_size=10)
    rospy.init_node('door_command', anonymous=True)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
                print("Press 1 to open or 0 for close and then enter. ")
                n = int(raw_input())
                rospy.loginfo(str(n))
                pub.publish(n)
                rate.sleep()

if __name__ == '__main__':
        try:
                talker()
        except rospy.ROSInterruptException:
                rospy.loginfo("Measurement stopped by user.")
```

door_sommand.py

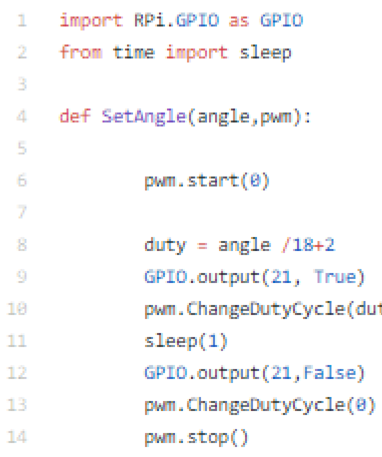
- *explanation for the script in section 5.3*



```python
#!/usr/bin/env python

import rospy
from std_msgs.msg import Int8
import openDoor
import closeDoor
import RPi.GPIO as GPIO

def callback(msg):
        pwm = GPIO.PWM(21, 50)
        rospy.loginfo(rospy.get_caller_id() +" "+ str(msg.data))
        if(msg.data==0):
                openDoor.openDoor(pwm)
        elif(msg.data==1):
                closeDoor.closeDoor(pwm)

def listener():

        rospy.init_node('door_listen', anonymous=True)

        rospy.Subscriber('keypress_door', Int8, callback)

        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(21, GPIO.OUT)
    # spin() simply keeps python from exiting until this node is stopped
        rospy.spin()

if __name__ == '__main__':
        try:
                listener()
                GPIO.cleanup()

        #when pressing CTR+C
        except KeyboardInterrupt:
                pass
```

door_listen.py

- *explanation for the script in section 5.3*



```python
import RPi.GPIO as GPIO
from time import sleep

def SetAngle(angle,pwm):

        pwm.start(0)

        duty = angle /18+2
        GPIO.output(21, True)
        pwm.ChangeDutyCycle(duty)
        sleep(1)
        GPIO.output(21,False)
        pwm.ChangeDutyCycle(0)
        pwm.stop()
```

servo.py

- *explanation for the script in section 5.3*

References

[1] N. Guelfi, "BiCS Bachelor Semester Project Report Template,"2019. [Online]. Available: https://github.com/nicolasguelfi/lu.uni.course.bics.global/blob/master/lu.uni.course.bics.all.projects.template.reports/main.pdf [Accessed 31st May 2019]

[2] D.Feil-Seifer and M.J Mataric, "Human-Robot Interaction" [Online]. Available: http://robotics.usc.edu/publications/media/uploads/pubs/585.pdf [Accessed 1st June 2019]

[3] M. Chu and B. Begole,”17.3.1 Natural Interaction”2010. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/natural-interaction [Accessed 1st June 2019]
[4] The MagPi magazine, “Breadboard Tutorial ”2017. [Online]. Available: https://www.raspberrypi.org/magpi/breadboard-tutorial/ [Accessed 1st June 2019]
[5] Raspberry Pi Fondation, “GPIO Pins ”. [Online]. Available: https://projects.raspberrypi.org/en/projects/physical-computing/3 [Accessed 1st June 2019]
[6] By Ros.org, “About ROS” [Online]. Available: https://www.ros.org/about-ros/ [Accessed 1st June 2019]
[7] By ModMyPi LTD, “HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi”[Online]. Available: https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi [Accessed 1st June 2019]
[8] The Pi Hut. Ecommerce Software by Shopify,” Turning on an LED with your Raspberry Pi’s GPIO Pins” 2019[Online]. Available: https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins [Accessed 1st June 2019]
[9] RPi Labs, “Controlling a Servo from the Raspberry Pi”. [Online]. Available: https://rpi.science.uoit.ca/lab/servo/ [Accessed 1st June 2019]
[10] The Pi Hut, “Raspberry Pi Camera Module V2” [Online]. Available: https://thepihut.com/products/raspberry-pi-camera-module [Accessed 1st June 2019]
[11] Y. Tawil, “An Introduction to Robot Operating System (ROS)”, 2017. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/ [Accessed 2nd June 2019]
[12] Ubiquity Robotics, “ROS node for camera module of Raspberry Pi”, 2018.[Online]. Available: https://github.com/UbiquityRobotics/raspicam_node/tree/195694afee514370aaf28712e1e09c48bdaf2af7 [Accessed 4th June 2019]
[13] By Raspberry Pi Foundation, “Connect the Camera Module”. [Online]. Available: https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/4 [Accessed 1st June 2019]
[14] The Pi Hut, “Turning LED with Raspberry Pi’s GPIO Pins”, 2019. [Online]. Available: https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins [Accessed 2nd June 2019]
[15] Gus, “Raspberry Pi Distance Sensor: How to setup the HC-SR04”,2019. [Online]. Available: https://pimylifeup.com/raspberry-pi-distance-sensor/ [Accessed 2nd June 2019]
[16] lanc1999, “Servo Motor Control With Raspberry Pi”. [Online]. Available: https://www.instructables.com/id/Servo-Motor-Control-With-Raspberry-Pi/ [Accessed 2nd June 2019]