TECHNISCHE UNIVERSITÄT
ILMENAU

RASPBERRY PI PROJECT DESCRIPTION, THE ROD TRACKER

JUN.-PROF. DR. MATTHIAS HIRTH
EDWIN GAMBOA M.ENG.
JOSE ALEJANDRO LIBREROS M.ENG.

Version: October 18, 2021

CONTENTS

1 INTRODUCTION 1

2 GENERAL RECOMMENDATIONS 2
- 2.1 Remote access . . . 2
  - 2.1.1 Raspberry Pis . . . 2
  - 2.1.2 Access via SSH . . . 2
  - 2.1.3 First steps after access . . . 3
  - 2.1.4 Time slots . . . 3
- 2.2 Development . . . 4
- 2.3 Coding . . . 4
- 2.4 The fake GPIO script . . . 4
- 2.5 Testing on Raspberry Pi . . . 4
- 2.6 Practice . . . 5

3 THE WEB INTERFACE 6
- 3.1 Development instructions . . . 6
  - 3.1.1 The folders . . . 6
  - 3.1.2 The app script . . . 7
  - 3.1.3 The main client script . . . 7
  - 3.1.4 The index template . . . 8
  - 3.1.5 Testing the Raspberry Pi . . . 8
- 3.2 Recommended tutorials . . . 9

4 TASK 1: OPENCV CONTROL 10
- 4.1 Development instructions . . . 10
  - 4.1.1 opencv_controller.py . . . 10
  - 4.1.2 camera.py . . . 11
  - 4.1.3 main.py . . . 11
  - 4.1.4 Testing on the Raspberry Pi . . . 11
- 4.2 Displaying the results via the Web interface . . . 12
  - 4.2.1 The app script . . . 12
  - 4.2.2 The main client script . . . 12
  - 4.2.3 The index template . . . 12
- 4.3 Recommended tutorials . . . 13

5 TASK 2: MOTOR CONTROL 14
- 5.1 Development instructions . . . 14
  - 5.1.1 motor_controller.py . . . 14
  - 5.1.2 main.py . . . 15
- 5.2 Controlling the motor via the Web interface . . . 15
  - 5.2.1 The app script . . . 15
  - 5.2.2 The main client script . . . 15
  - 5.2.3 The index template . . . 15
- 5.3 Possible improvements . . . 16

5.4 Recommended tutorials 16

6 TASK 3: DISTANCE SENSOR CONTROL 17
6.1 Development instructions 18
6.1.1 sensor_controller.py 18
6.1.2 main.py 18
6.2 Controlling the sensor via the Web interface 19
6.2.1 The app script 19
6.2.2 The main client script 19
6.2.3 The index template 19
6.3 Recommended tutorials 19

BIBLIOGRAPHY 20

1 INTRODUCTION

This document contains a guide for the Raspberry Pi project as part of the Academic Preparation Course (APC). The project consists of four tasks, which address various media technology aspects, such as image processing, software development, hardware controlling, project management, and teamwork. Tasks 1 to 3 are independent of each other,. You will start with Task 1. Then you will continue with Task 2 or 3 as you prefer.

We will offer regular (digital) consultations during the semester. Also, we will support you via a forum of the APC Moodle course. We will provide each group with a Git repository, which should be correctly administered and evidence the work of each group participant, i.e., via Git commits and issues. We will monitor the progress of the groups via the repositories.

The hardware setup for the project is ready to use. Thus, you only need to code the expected functionalities. You can access the Raspberry Pi via SSH. Each group will get a user account for this purpose.

You will watch the effects of your inputs on the test bed via a webcam stream[1]. Your group should book time slots in advance via the Moodle to access your user account at the Raspberry Pi. After the time expires, the Raspberry Pi will reboot and access rights will be provided to the corresponding group. So please make sure, you continuously push your code to the provided Git server.

Deadlines will be set for the submission of each task to ensure the progress of the project. After each submission, there will be an oral discussion session, in which your group should show the code working and answer questions regarding the implementation made by you. The questions are to make sure that you understood everything that you coded. Each submission will be graded in a 3-levels scale:

- **Insufficient (-1):** it means that you did not met the expected goal or you did not demonstrated that you understood what you did.
- **Sufficient (0):** it means that you met the expected goals and did well during the oral discussion.
- **Better than expected (1):** it means the expected goals are achieved, you did well during the oral discussion, and demonstrated additional work; e.g., by employing good programming practices, working on optimize your solution, etc.

Please note that this guide only provides a general approach on how to implement the project. You can solve the tasks in different ways. Be creative and dare to make decisions on specific solution variants.

The rest of this document presents guidance on remote access to the Raspberry Pi in Chapter 2. Then, general guidelines to complete Task 1, Task 2, Task 3 are presented in Chapter 4, Chapter 5 and Chapter 6 respectively. Note that a web interface is going to be developed along the three tasks, general details about it are described in Section 6.2. Each task has an associated set of tutorials listed at the end of each Chapter. Make sure to check the relevant tutorials before starting to work on each task.

1 Streaming: http://apc-stream.rz.tu-ilmenau.de/

2 GENERAL RECOMMENDATIONS

2.1 REMOTE ACCESS

Each group will receive a *username*, similar toe.g., *sose2021groupa*, and a *password* to access the Raspberry Pi via SSH. If you are not familiar with SSH, you can watch this video [1].

Take into account that the username and password is for the whole group, and is different to those for the GitLab server.

Furthermore, if you are using Visual Studio Code, you can follow this tutorial[2] to connect via SSH directly there.

2.1.1 *Raspberry Pis*

You will have access to 2 Raspberry Pis so that you have enough time to test your code in our test bed. As depicted in Table 2.1, the *Raspberry Pi B* is not connected to the stepper motor; thus, if you want to test the motor, you should use the *Raspberry Pi A*

Table 2.1: Hardware availability per Raspberry Pi

DEVICE	STEPPER MOTOR	DISTANCE SENSOR	PI CAMERA
Raspberry Pi A	Yes	Yes	Yes
Raspberry Pi B	No	Yes	Yes

2.1.2 *Access via SSH*

You should use the provided username and password to access the Raspberry Pi via SSH and conduct your project. Then:

1. Open a console like the *Terminal* in Linux or Mac, or the *Cmd* or *Gitbash* in Windows. Note that you might have to enable the OpenSSH Server in Windows.
2. Run the following command:

   ```
   ssh [username]@emt15.rz.tu-ilmenau.de -p [port]
   ```

   Note that you should replace *[username]* as needed. Also, set the *-p* parameter to *4442* for *Raspberry Pi A* and *4441* for *Raspberry Pi B*.
3. Enter your password and start coding.

1 SSH Video: https://www.youtube.com/watch?v=qWKK_PNHnnA
2 SSH in vscode: https://code.visualstudio.com/docs/remote/ssh

2.1.2.1 *Access outside without university network, including FEM*

Take into account that you should use the university VPN access if you are not connected to the university network. Even if you are connected through the FEM network, you should use the VPN. Take the following into account:

1. Check the information presented in [3] to configure the VPN access.
2. If everything is configured correctly when you access [4], your IP should be located in Germany, Ilmenau, and the ISP should be Deutsches Forschungsnetz
3. You should be able to access the Raspberry Pi as explained above.

2.1.3 *First steps after access*

When you have already accessed, you will find three folders in your home directory:

1. *rod_env*: this folder contains the Python environment for your project. **You must activate it** before staring to work, running the following command:

   ```
   . rod_env/bin/activate
   ```

   The home directory should be the current location in the Terminal.
2. *src*: this folder contains the source code from your Git repository. So, here you can pull, commit, push, etc.
3. *logs*: this folder is only relevant for Task 4. Here you will be able to read the logs of the server tools for the web application, i.e., Nginx and gunicorn. You need to run *ls* command to list the available logs. If you have not run the web application, this folder will be empty.

2.1.4 *Time slots*

You should book slot times to access the Raspberry Pi via the Moodle course. You will have access only during those times. 5 minutes before your time is over, you will see a message similar to the one below:

```
Broadcast message from root@raspberrypi on pts/0 (Tue 2020-11-17
    12:15:53 CET):

The system is going down for reboot at Tue 2020-11-17 12:20:53 CET!
```

You will **not have access until your next time slot** after the system reboots. Although your files and folders will not be lost, **you must commit and push your work** to your GIT repository. Also, you might want to work on your computer and pull changes to the Raspberry Pi. If you have booked consecutive time slots, the system will reboot only after the last time slot finishes.

3 VPN: https://intranet.tu-ilmenau.de/site/unirz/SitePages/TUILM-VPN.aspx (TU Ilmenau login required)
4 https://whatismyipaddress.com/

2.2 DEVELOPMENT

2.3 CODING

1. You should use Python to complete this project.
2. Use a virtual environment for your project [16]. Activate it always to be able to run your code.
3. If you are not familiar with Terminal text editors like *nano* and *vim*, you might want to code on your computer and not directly on the Raspberry Pi.
4. You can implement your auxiliary methods and import the libraries you consider necessary to achieve the expected behavior. But make sure that the suggested methods do what they are meant to, thus, when running the *main.py* scripts of each task everything works correctly.
5. If you require external libraries, add them to the *requirements.txt* and/or *requirements_raspi.txt* files so that they get installed every time it is required. Note that the second file is to be used in the Raspberry Pi because some libraries, e.g., *picamera* are not available for operating systems like Windows. These will allow you developing locally and for the Raspberry Pi without problems.

2.4 THE FAKE GPIO SCRIPT

In you project you will find a fake script called *fake_gpio.py* in the folders of **Task 1 Chapter 5 and Task 2 Chapter 6**. **It is intended for development purposes** outside the Raspberry Pi. This package contains the following files: It has an implementation of the methods from the *RPi.GPIO* class, that you should use to control the stepper motor and ultrasonic sensor of this project. This script will help you implement and test your code without needing to access the Raspberry Pi. This script include 5 methods; i.e., *setmode*, *setup*, *output*, *input* and *cleanup*.

You can use this fake module while developing in your own PC by importing it in the desired script as follows:

```
from fake_gpio import GPIO
```

However, to test in the Raspberry Pi you should import the proper library as follows:

```
import RPi.GPIO as GPIO
```

2.5 TESTING ON RASPBERRY PI

1. Follow the steps presented above (Section 2.1) every time you need to access the Raspberry Pi.
2. Pull your code from your repository in the *src* folder.
3. Activate the *rod_env* as explained in Section 2.1.3.
4. If you have added libraries to the *requirements_raspi.txt* file you should install them in your environment using the following command.

```
pip install -r requirements\_raspi.txt
```

Make sure that your environment was activated and that the current location of the terminal is the *src* folder.

5. You can watch the effects of your scripts via the online streaming[5].
6. Consider using the *print()* method of Python to debug your code.

Check Section 3.1.5 to know how to test the web interface.

2.6 PRACTICE

In your repository you will find a folder called *practice_room*. Please use this as a sandbox to learn and play with Python and JavaScript.

5 Streaming: http://apc-stream.rz.tu-ilmenau.de/

3

THE WEB INTERFACE

You will develop a simple web interface using Flask [8] to show the results of each task via a web browser. In the description of each task, we detail how you should present those results. Figure 3.1 depicts an example of the expected finished web interface. It also highlights which elements correspond to each task.

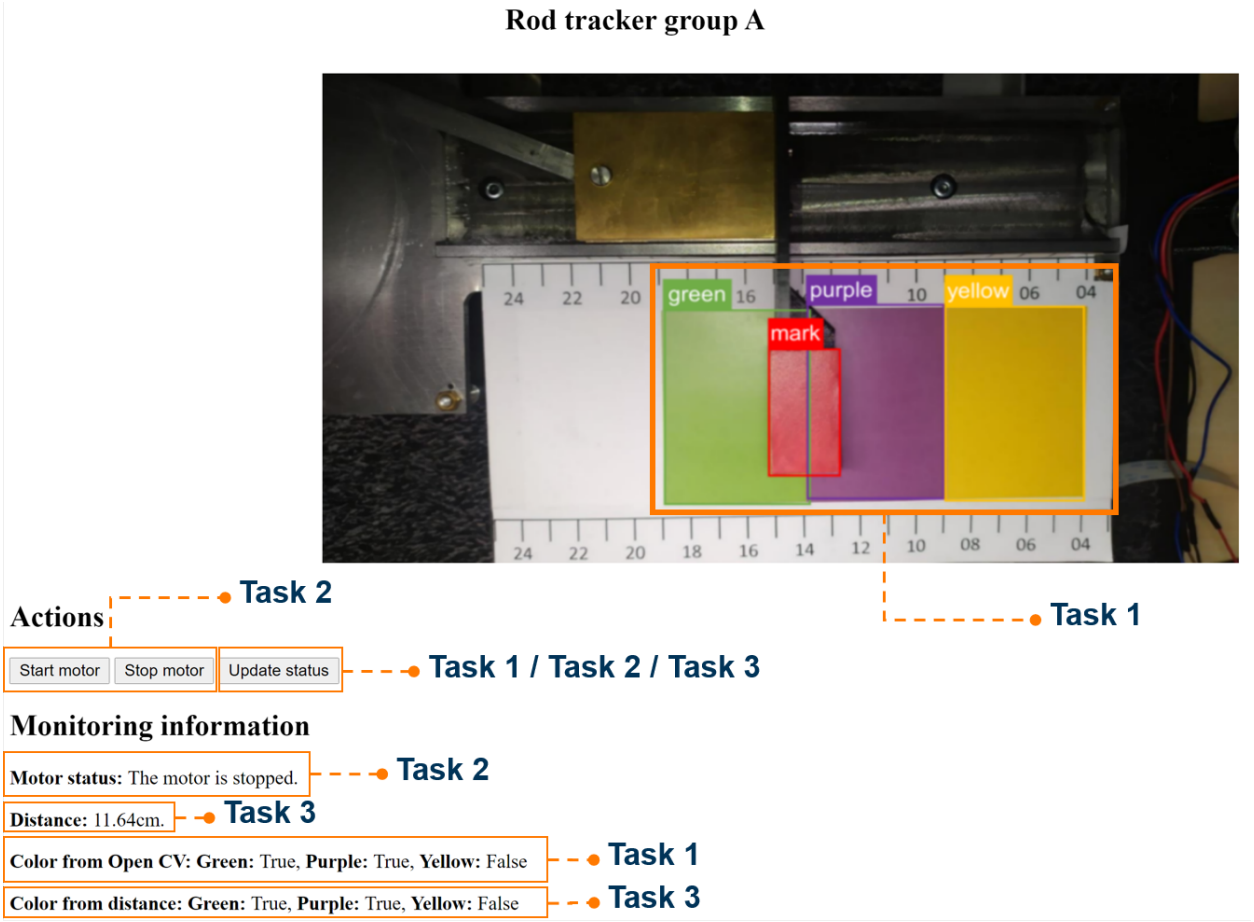

Figure 3.1: Expected result of the whole web interface and the relationship between the visual elements and each Task

3.1 DEVELOPMENT INSTRUCTIONS

Take into account the general recommendations presented in Section 2.2 for managing the web app properly. In the following, we describe the content of the base code that we provide to you.

3.1.1 *The folders*

The base code contains the following folders for the project to work properly:

1. *practice_room*:this folder is meant to be used by you as a place to test code and learn Python.
2. *task1_opencv_control*: this folder contains the scripts required to complete the Task 1 described in Chapter 4.
3. *task2_motor_control*: this folder contains the scripts required to complete the Task 2 described in Chapter 5.
4. *task3_sensor_control*: this folder contains the scripts required to complete the Task 3 described in Chapter 6.
5. *templates*: it contains only the index template because this is a one-page application (See Section 3.1.4 for more details).
6. *static*: it contains three sub-folders:
   a) *js*: it contains the main client-side (browser) script of this application (See Section 3.1.3 for more details).
   b) *css*: it includes the style sheet for this application. You are free to modify it as desired to change the visual aspect and layout of the application.
   c) *vendor*: it contains the external JavaScript libraries that you require. In this case, you will find a library called *axios*, which is used by the browser client to make requests to the server. You can add your libraries if required. These libraries are client-side or front-end and are imported in the index template. They are not the same as the server-side libraries imported in Python files and registered in the *requirements.txt* or *requirements_raspi.txt* file.

3.1.2 *The app script*

The *app.py* script should be used to run a Flask [8] application locally at http://localhost:5000/. This script has three global variables:

1. *motor_controller*: which is an instance of the *MotorController* class.
2. *opencv_controller*: which is an instance of the *OpenCVController* class.
3. *sensor_controller*: which is an instance of the *SensorController* class.

Additionally, this script contains 1 method and 8 *views* [2] to answer the requests made from the web interface in a browser. You will implement all these views along the whole project. Take into account that you need to replace the #... marks with your own code to achieve behavior expected for each task as described in Chapter 4, Chapter 5 and Chapter 6.

3.1.3 *The main client script*

The main client script is contained in */statict/js/main.js* file. This script makes the requests to the server and updates the status of the motor, the current distance and color zone on the web interface.

It contains 10 methods which you will implement along the whole project. For this, you should replace the //... marks with your own code.

3.1.4 *The index template*

The index template is similar to a normal HTML file, except that you can use the *Jinja* template language [12] to determine content based on server variables. For instance, in the provided code the *url_for* expression is used to determine the paths of the required resources.

Take into account that the page is divided into three sections identified by the following HTML *id*s:

1. *streaming_viewer*: it should display the streaming video.
2. *actions*: it should contain the HTML elements that allow the user starting the motor, or forcing it to stop if the red mark is in the target area. The buttons should be available in the main client script via the ids of the elements. In the provided code, only the *Update status* button is included, use this button as an example to add the missing elements.
3. *monitoring_info*: contains the HTML elements that allow displaying the information of the current status of the system. These elements might be accessed in the main client script via their *ids*. The provided code only includes the HTML elements to display the *Current color from Open CV*, use these as an example to add the missing ones.

3.1.5 *Testing the Raspberry Pi*

You **should not** run the command **python app.py** or **flask run** to test the web app on the Raspberry Pi, because it is already running, every time you have made changes and want to test, you should:

1. Access via SSH as explained in Section 2.1
2. Pull your code from your Git repository to the *src* file
3. Every time you make changes, run the the following command to reload the app in the server:

   ```
   sudo systemctl restart app.service
   ```

   This command requires to be run as **sudo** and the **.service** extension to work.
4. To know whether the app is running or identify errors, run the the following command:

   ```
   sudo systemctl status app.service
   ```

   To check the whole log of errors related to the app starting, run:

   ```
   sudo journalctl -u app.service
   ```

   To filter the content of the log errors, you can combine the last command with *grep*. Check [14] to know how to use this functionality.

   Furthermore, you can look for errors in the *logs* folder as explained in Section 2.1.3.
5. Access your app in your own browser at http://emt15.rz.tu-ilmenau.de:[port]/. Your app is accessible only within your booked time slots.

   Note that you should replace the *port* value with *8082* for *Raspberry Pi A* and *8081* for *Raspberry Pi B*.

3.2 RECOMMENDED TUTORIALS

1. Flask installation [4]
2. Flask quick start tutorial [8]
3. Flask + Video Streaming + OpenCV (for face detection) [1]
4. JavaScript for beginners [5]
5. Server requests using axios and flask [6]
6. CSS for beginners [3]

4

TASK 1: OPENCV CONTROL

The Raspberry Pi is connected to a Camera module. The aims of this task are to:

1. Process the video captured by the Camera module using OpenCV.
2. Detect, highlight and label a red mark and three colors zones as shown in Figure 4.1.
3. Determine in which color zone the red mark is currently.
4. Stream the frames processed via OpenCV.

When working locally you should use the provided the fake Camera. Meanwhile, when working in the Raspberry Pi, you should use the real Python modules to control the connected camera.

4.1 DEVELOPMENT INSTRUCTIONS

You should use the *task3_opencv_control* folder, which contains three scripts described below.

4.1.1 *opencv_controller.py*

This script contains a class called *OpenCVController*. This script should allow monitoring the behavior of the red mark using the OpenCV image processing library. You are expected to monitor the position of the red mark and determine in which color zone it is located. You should implement the following method:

1. *process_frame()*: This method has a *frame* parameter/argument, which is a frame captured by the Raspberry Pi camera, or returned by the fake camera if you are working on your local computer. It should process the frame and identify in which color zone the red mark is located. Use OpenCV for color detection [11] to track, highlight and label both elements in a frame similar to the one presented in Figure 4.1.

   Moreover, the method should set value of *current_color* variable with a string indicating the current color zone using Boolean values as follows:

   - *"[True,False,False]"* if the red mark is within the green zone.
   - *"[False,True,False]"* if he red mark is within the purple zone.
   - *"[False,False,True]"* if he red mark is within the yellow zone.

   It is possible that the red mask is overlapping two colors. In that case, the method should return two True values and one False. For instance, if the red mark is located between the purple and yellow zone, the method should return *"[False,True,True]"*.

2. *get_current_color()*: it returns the value of *current_color*. This method is already implemented in the provided code.

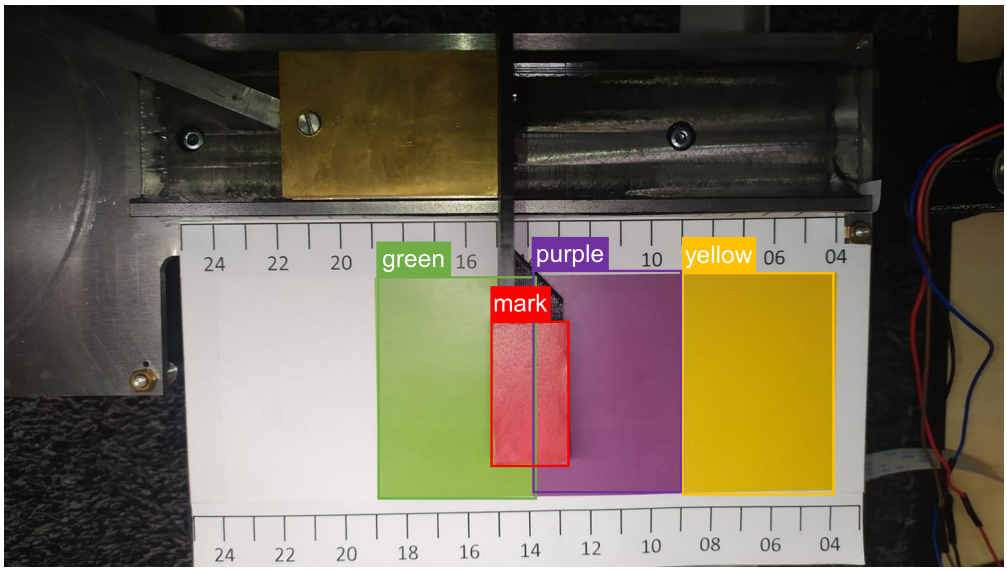

Figure 4.1: Image captured with the Camera Module connected to the Raspberry Pi B. The red mark and the zones are highlighted and labeled using OpenCV

4.1.2 *camera.py*

This script implements a fake camera that returns the images contained in the same folder. It is just for testing your code while working in your local machine. **You should implement the *pi_camera.py*** script using the *picamera* module, so that you can stream the real setup.

4.1.3 *main.py*

You should run this script to test your code. If the method *process_frame()* is implemented properly, it should behaves as described in Table 4.1 for the fake camera. We will use this script to test and check the correctness of your code. Thus, do not change the content of this script.

Table 4.1: Expected results for the test frames

TEST FRAME	EXPECTED OUTPUT
1.jpg	Current color from OpenCV: [True,True,False]
2.jpg	Current color from OpenCV: [True,False,False]
3.jpg	Current color from OpenCV: [False,True,True]
4.jpg	Current color from OpenCV: [False,False,True]

You can stop the script using $CTRL + C$ on the console.

4.1.4 *Testing on the Raspberry Pi*

Check the Section 2.1 to know how to access the Raspberry Pi. While working on the Raspberry Pi, you cannot use the Pi camera unless you stop the web application that is running always during your slots. Otherwise, you will get an *Out of resources error*. Even if you have not started working

on the web application, it will run automatically during your slots. So, you should stop the app as follows to test properly:

```
sudo systemctl stop app.service
```

To start it again, use:

```
sudo systemctl start app.service
```

4.2 DISPLAYING THE RESULTS VIA THE WEB INTERFACE

The functionalities of the web interface for this task are already implemented, and you should use them as example for the future tasks. If you have implemented everything as indicated in the previous sessions, this should work correctly with the fake camera and with the Raspberry Pi camera.

Take into account the recommendations presented in Section 6.2 to test everything.

4.2.1 *The app script*

The following views are relevant for this task:

1. *get_frame*: it should constantly return the frame processed by the *opencv_controller*. This view is already implemented. But for it to work, you should modify the *process_frame* method of *opencv_controller* script to return a frame that can be displayed on the web interface every time it is called.
2. *video_feed*: it should return a response containing the frame returned by *get_frame*. This will be employed by the *img* HTML element of the *index.html* template to render the streaming.

4.2.2 *The main client script*

Check Section 3.1.3 for more details about this script. The following methods are relevant for this task:

1. *requestColorFromOpenCV*: this method should request the server to get the current color using the *Open CV controller*. Check this tutorial [13] to know how to handle *axios* requests in JavaScript.
2. *updateCurrentColorOpenCV*: it updates the current colors based on OpenCv on the web interface.
3. *updateStatus*: this method calls the *updateCurrentColorOpenCV* to perform the update.

4.2.3 *The index template*

Check Section 3.1.4 for more details about this file. This file defines the following HTML elements.

1. *streaming_viewer*: it should display the streaming video. If the previous steps are done correctly, this you should already work.

2. *monitoring_info*: this contains *span* element with the id *color_open_cv*, this should show the current color in which the red mark is located based on the *get_current_color()* method of the *opencv_controller*.

3. *actions*: it contains a button called *Update status*, when it is clicked the JavaScript method *updateStatus* is called and the monitoring data should be updated. This button is partly implemented and updates the current color based on Open CV.

4.3 RECOMMENDED TUTORIALS

1. OpenCV in Python [7]
2. Install OpenCV under environment [15]
3. Color detection using OpenCV [11]
4. The tutorials presented in Section 6.2

5

TASK 2: MOTOR CONTROL

For this task, the Raspberry Pi is connected to a stepper motor to move a rod on a track. The rod has a red mark attached to one end (See Figure 5.1). The goals of this task are:

1. Control the motor via the web interface to allow performing one of three movements described below.
2. Keep the status of the motor updated indicating whether the motor is rotating or not and display it via web interface.

(a) View from one side

(b) View from above

Figure 5.1: Stepper motor's views within the experiment set up. The stepper motor, the rod and the attached red mark are shown.

5.1 DEVELOPMENT INSTRUCTIONS

You should employ the *task1_motor_control* folder for this task, which contains the two scripts described below. Take into account the recommendations presented in Section 2.2.

Remember that you can use the fake *GPIO* script of your project to simulate the motor controlling outside the Raspberry Pi as explained in Section 2.4.

5.1.1 *motor_controller.py*

This script contains a class called *MotorController* which should allow controlling the behavior of the stepper motor. This class has two constant attributes, i.e., *PIN_STEP* and *PIN_DIR*. **Do not change these values**, otherwise your code will not work.

The *MotorController* class contains the following methods:

1. *start_motor*: it allows starting the stepper motor, it should choose randomly, whether the motor should to rotate 90° or 270°, and whether it should rotate clockwise; or counterclockwise. Take into account that the motor rotates 0.225° per step.

   Use the *print* method of Python to describe the rotation that has been chosen with a message similar to *"Rotating* 90° *degrees in counterclockwise direction"*.

   Moreover, it should set the *working* variable to indicate whether the motor is working (True) or already finished (False).

2. *is_working*: it returns the value of *working*. This method is already implemented in the provided code.

5.1.2 *main.py*

Run this script to test your code. If you have implemented the method and run this script in the Raspberry Pi, the motor should start working and perform one rotation selected randomly. Check the Section 2.1 to know how to access the Raspberry Pi. Remember that the motor can only be tested on the *Raspberry Pi A*. We will use this script to test and check the correctness of your code. Thus, do not change the content of this script.

5.2 CONTROLLING THE MOTOR VIA THE WEB INTERFACE

Remember to check to get more information about the scripts mentioned below.

5.2.1 *The app script*

You need to replace the #... marks with your own code in the following views:

1. *start_motor*: it should start the motor using the *motor_controller*.
2. *stop_motor*: it should stop the motor using the *motor_controller*.
3. *motor_status*: it should return a Boolean indicating whether the motor is currently rotating.

5.2.2 *The main client script*

You should replace the //... marks with your code to implement the expected behavior. Use the *requestColorFromOpenCV* and *updateCurrentColorOpenCV* methods (see Section 4.2.2) as examples for implementing the following ones:

1. *requestStartMotor*: it should request the server to start the motor. Also it should update the motor status.
2. *requestStopMotor*: it should request the server to stop the motor. Also it should update the motor status.
3. *updateMotorStatus*: it should update the web interface to indicate whether the motor is rotating or not.
4. *updateStatus*: moreover you should modify this method to also update the motor status every time the *Update status* button is clicked. This method should be called every time the motor has finished a rotation.

5.2.3 *The index template*

Replace the <!– ... –> marks to add the HTML elements needed to display the status of the motor and include the following two buttons.

1. *Start motor*: it should request the server to start the motor.
2. *Stop motor*: it should request the server to stop the motor.

5.3 POSSIBLE IMPROVEMENTS

Some ideas to improve the project are:

- Control the motor to allow performing a certain amount of steps to move the rod until the red mask is within the next color zone.
- Control the motor to move to a certain color

5.4 RECOMMENDED TUTORIALS

1. Virtual environments in python [16]
2. Raspberry Pi stepper motor tutorial [10]

6

TASK 3: DISTANCE SENSOR CONTROL

For this task, you should control a ultrasonic distance sensor to:

1. To measure the distance between the sensor and the board attached to the rod (See Figure 6.1).
2. Keep a variable updated indicating the last calculated distance and display it via the web interface.
3. Get the current color zone based on the calculated distance.

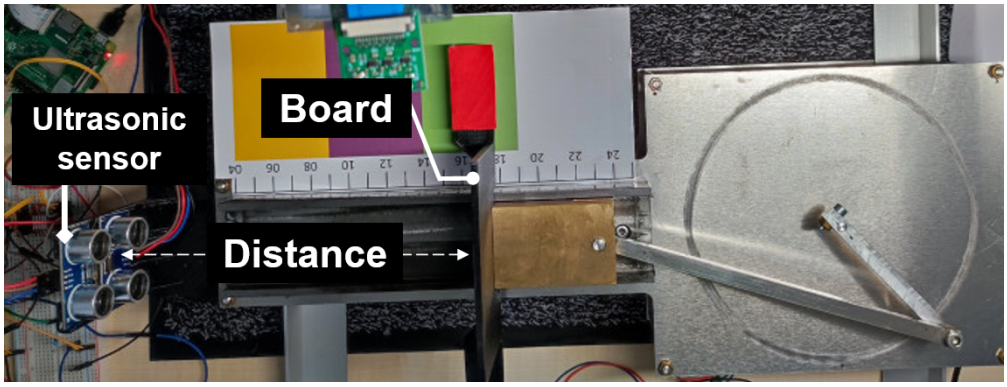

Figure 6.1: Ultrasonic sensor role in the experiment. The ultra sensor, the board, and the distance to be measured are depicted

Note that it is not required to have finished the other tasks to start working on this one. Take into account that the setup includes 3 tape measures as shown in Figure 6.2, so that you can guide your test for the distance sensor via your streaming with the Raspberry Pi camera or the streaming provided by us. Take into account that the video resolution of the streaming and some numbers might not be totally visible, so save this image as a reference for you. The measures are in cm. The minimum possible distance is approximately 4 cm, and the maximum distance is approximately 19 cm.

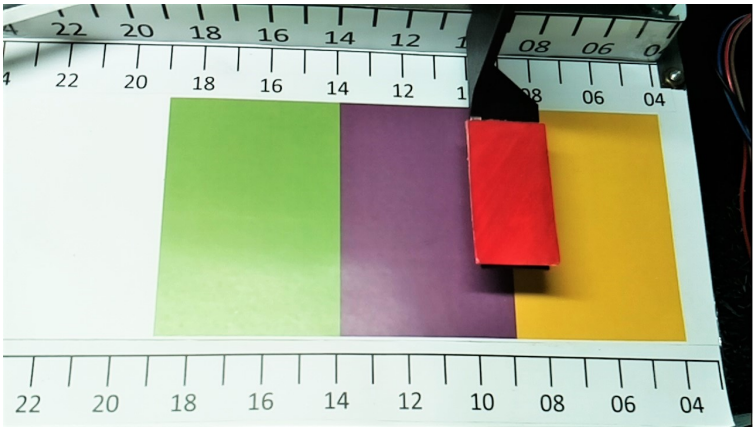

Figure 6.2: Tape measures included in the test bed.

6.1 DEVELOPMENT INSTRUCTIONS

Use the *task2_sensor_control* folder, which contains three scripts described below. As always, take into account the recommendations presented in Section 2.2.

Remember that you can use the fake *GPIO* script of your project to simulate the motor controlling outside the Raspberry Pi as explained in Section 2.4.

6.1.1 *sensor_controller.py*

It contains a class called *SensorController* that should detect the distance from the rod to the ultrasonic sensor. This class has two constant attributes, i.e., *PIN_TRIGGER* and *PIN_ECHO*, which **you should not change**, otherwise your code will not work.

The *SensorController* class contains the following methods:

1. *get_distance:* it should allow tracking the distance between the sensor and the the board. Also, it should set the value of the *distance* variable based on the data provided by the sensor.

   You should make **20 measurements** and the distance should be the median of those measurements.

2. *get_color_from_distance*: it should return the value of the *color_from_distance* variable. It should calculate the color based on the distance only, i.e., not using OpenCV. Take into account the distance ranges presented in Table 6.1 to determine the current color.

Table 6.1: Distance ranges of each color zone

COLOR ZONE	DISTANCE RANGE
Green	14 -19 cm
Purple	9 -14 cm
Yellow	4 - 9 cm

6.1.2 *main.py*

This script will print "*Distance: x*" on the console using the *SensorController* class. *x* is replaced by the calculated value. If you have not implemented the methods properly, it will probably print *None* instead. At the same time, This script will print "*Current color from sensor: [True or False,True or False, True or False]*" on the console depending on the identified color. As you notice, it is an array a Boolean values, the first position should be True if the current color is green, other wise it should be False. The second position is for the purple zone, and the third one for the yellow color. It is possible that the red mask is overlapping two colors. In that case, the method should return two True values and one False. You can stop the script using $CTRL + C$ on the console. We will use this script to test and check the correctness of your code. Thus, do not change the content of this script.

Check the Section 2.1 to know how to test on the Raspberry Pi.

6.2 CONTROLLING THE SENSOR VIA THE WEB INTERFACE

Remember to check to get more information about the scripts mentioned below.

6.2.1 *The app script*

You need to replace the #... marks with your own code in the following views:

1. *get_distance*: it should get the distance from the rod to the ultrasonic sensor.
2. *get_color_from_distance*: it should get the color based on the distance from the rod to the ultrasonic sensor.

6.2.2 *The main client script*

You should replace the //... marks with your code to implement the expected behavior. Use the *requestColorFromOpenCV* and *updateCurrentColorOpenCV* methods (see Section 4.2.2) as examples for implementing the following ones:

1. *updateDistance*: it updates the current distance based on the result of calling *requestDistance*.
2. *requestDistance*: it request the server to calculate the current distance using the *sensor_controller*.
3. *updateCurrentColorDistance*: it updates the current colors based on the result of calling *requestColorFromDistance*.
4. *requestColorFromDistance*: this method should request the server to get the current color using the *sensor_controller*. Check this tutorial [13] to know how to handle *axios* requests in JavaScript.
5. *updateStatus*: moreover you should modify this method to display the current distance and the current color based on that distance every time the *Update status* button is clicked.

6.2.3 *The index template*

Replace the <!– ... –> marks to add the HTML elements needed to display the current distance and the current color based on the calculated distance.

6.3 RECOMMENDED TUTORIALS

1. Virtual environments in python [16]
2. Raspberry Pi Distance Sensor using the HC-SR04 [9]

BIBLIOGRAPHY

[1] Anmol Behl. *Video Streaming Using Flask and OpenCV*. Feb. 2. URL: https://medium.com/datadriveninvestor/video-streaming-using-flask-and-opencv-c464bf8473d6 (visited on 08/06/2020).

[2] *Blueprints and Views*. URL: https://flask.palletsprojects.com/en/1.1.x/tutorial/views/ (visited on 09/04/2020).

[3] *CSS: Cascading Style Sheets*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (visited on 08/06/2020).

[4] *Installation - Flask Documentation*. URL: https://flask.palletsprojects.com/en/1.1.x/installation/ (visited on 05/20/2021).

[5] *JavaScript — Dynamic client-side scripting*. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript (visited on 08/06/2020).

[6] Chitrank Mishra. *How to make POST call to an API using Axios.js in JavaScript?* Apr. 2020. URL: https://www.geeksforgeeks.org/how-to-make-post-call-to-an-api-using-axios-js-in-javascript/ (visited on 08/06/2020).

[7] *OpenCV-Python Tutorials*. URL: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html (visited on 05/20/2021).

[8] *Quickstart - Flask Documentation*. URL: https://flask.palletsprojects.com/en/1.1.x/quickstart/ (visited on 05/20/2021).

[9] *Raspberry Pi Distance Sensor using the HC-SR04*. URL: https://pimylifeup.com/raspberry-pi-distance-sensor/ (visited on 11/18/2020).

[10] *Raspberry Pi Stepper Motor Tutorial*. URL: https://www.rototron.info/raspberry-pi-stepper-motor-tutorial/ (visited on 11/18/2020).

[11] Adrian Rosebrock. *OpenCV and Python Color Detection*. Aug. 2014. URL: https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/ (visited on 08/06/2020).

[12] *Template Designer Documentation*. URL: https://jinja.palletsprojects.com/en/2.11.x/templates/ (visited on 08/20/2020).

[13] Joy Warugu. *Asynchronous Javascript using Async - Await*. Aug. 2018. URL: https://scotch.io/tutorials/asynchronous-javascript-using-async-await (visited on 11/20/2020).

[14] *https://www.cyberciti.biz/faq/howto-use-grep-command-in-linux-unix/*. URL: https://www.cyberciti.biz/faq/howto-use-grep-command-in-linux-unix/ (visited on 07/30/2021).

[15] *opencv-python 4.3.0.36*. URL: https://pypi.org/project/opencv-python/ (visited on 08/06/2020).

[16] *venv - Creation of virtual environments*. URL: https://docs.python.org/3/library/venv.html (visited on 08/06/2020).