TECHNISCHE UNIVERSITÄT
ILMENAU

RASPBERRY PI PROJECT DESCRIPTION, AUTOMATA

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

Version: April 1, 2023

CONTENTS

1 INTRODUCTION

This document contains a guide for the Raspberry Pi project as part of the Academic Preparation Course (APC). The project consists of three tasks, which address various media technology aspects, such as image processing, software development, hardware controlling, project management, and teamwork. The tasks are independent of each other, but you can work on them in parallel if desired.

1.1 REMOTE TESTBED AND PROJECT MANAGEMENT

The hardware setup for the project is ready to use. Thus, you only need to code the expected functionalities because you will access a Raspberry Pi remotely via SSH. For this, each group will get a user account.

You will watch the effects of your code on the testbed via a webcam stream[1]. Your group should book time slots in advance, as we announce via Moodle, to access your user account at the Raspberry Pi. After your time slot 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.

1.2 CONSULTATIONS AND SUPPORT

We will offer regular[2] consultations during the semester. Also, we will support you via a forum of the 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.

1.3 GRADING

By completing this project you can get at most 62 points which are 78% of the points you can get in this course. You can get points for each task as shown in Table 1.1. In the description of each task, you will find out how exactly you can get those points.

Table 1.1: Maximum amount of points to get after completing each task.

TASK	BASIS POINTS	BONUS POINTS	TOTAL POINTS
Task 1: Motor control	17	3	20
Task 2: OpenCV control	19	3	22
Task 3: Distance sensor control	17	3	20

Review deadlines will be set for the submission of each task to ensure the progress of the project. After each submission, there will be a feedback session, in which you can show the progress you

1 Streaming: http://apc-stream.rz.tu-ilmenau.de/
2 Digitally via Webex with the access data published via Moodle

have made. We will give you recommendations to improve the project so that you can get all the points for the project.

At the end of the course, there will be an oral discussion, 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.

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

1.4 DOCUMENT ORGANIZATION

The rest of this document presents guidance on remote access to the Raspberry Pi in Chapter 2. In this project a web interface is going to be developed along the three tasks, general details about it are described in Chapter 3. Then, general guidelines to complete Task 1, Task 2, and Task 3 are presented in Chapter 5, Chapter 4, and Chapter 6 respectively. 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.

2 GENERAL RECOMMENDATIONS

2.1 REMOTE ACCESS

Each group will receive a *username*, similar to *wise2022groupa*, 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 are for the whole group, and are different from those for the GitLab server. Furthermore, we recommend you use Visual Studio Code as the development environment, 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 testbed. These are connected to a stepper motor, a distance sensor, and a camera. However, the motor of one of them, i.e., *Raspberry Pi B* is just for testing purposes, because its motor is not directly connected to the main setup. That is, the project can be tested on both devices, but the final demonstration should be carried out on *Raspberry Pi A*. Also, you should use Git properly to be able to have the same version of the project code on both devices and your local machines.

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. It is possible 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] -L 5000:localhost:5000
   ```

   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.

2.1.2.1 *Access outside without university network, including FEM*

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.

1 SSH Video: https://www.youtube.com/watch?v=qWKK_PNHnnA
2 SSH in vscode: https://code.visualstudio.com/docs/remote/ssh
3 VPN: https://intranet.tu-ilmenau.de/site/unirz/SitePages/TUILM-VPN.aspx (TU Ilmenau login required)

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 it, you will find two folders in your home directory:

1. *rod_env*: this folder contains the Python environment for your project. **You must activate it** before starting 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.

2.1.4 *Time slots*

You should book time slots to access the Raspberry Pi as we will announce 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. If you have booked consecutive time slots, the system will reboot only after the last time slot finishes.

2.2 DEVELOPMENT

1. You should use Python to complete this project.
2. Use a virtual environment for your project [17]. Activate it 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, e.g., using Visual Studio Code.
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 when it is required. Note that the second file will be used in the Raspberry Pi because some libraries are not available for operating systems like Windows, e.g., *picamera*. These will allow you to develop locally and for the Raspberry Pi without problems.

4 https://whatismyipaddress.com/

2.3 THE FAKE GPIO SCRIPT

You will find a fake script called *fake_gpio.py* in the folders of **Motor control task Chapter 4 and Sensor control task Chapter 6** of your project. **It is intended for development purposes** outside the Raspberry Pi. This package has an implementation of the methods from the *RPi.GPIO* class, which 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*. However, the behavior that these methods will produce is completely random and might not make any sense, again, these are just for testing purposes.

You can use this fake module while developing on 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
```

2.4 TESTING THE SCRIPTS ON THE RASPBERRY PI

1. Follow the steps presented in 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]. Also, consider using the *print()* method of Python to debug your code.

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

2.5 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 [9] 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.

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*: It is meant to be used by you as a place to test code and learn Python. The code in this folder will **not be considered** for grading your project.
2. *task1_motor_control*: It contains the scripts to complete the Task 1 described in Chapter 5.
3. *task2_opencv_control*: It contains the scripts to complete the Task 2 described in Chapter 4.
4. *task3_sensor_control*: It contains the scripts 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 other 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 [9] application locally at http://localhost:5000/, or at http://localhost:5001/ if ran from the Raspberry Pi. 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 with 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 5, Chapter 4 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 distance, and the digit on the web interface.

It contains 10 methods that you will implement along with 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 [13] 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 to start the motor, or force 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 digit detected with Open CV*, use these as an example to add the missing ones.

3.1.5 *Testing in the Raspberry Pi*

To test the web app which is running on the Raspberry Pi via SSH, 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. Make the desired changes
4. Run the *app.py* script [9] **in the Raspberry Pi**
5. Load the application **in your PC's web browser** at http://localhost:5001/.
6. Error or debugging logs will appear **on the Terminal/Console/Bash of the Raspberry Pi**

Notice that your app is accessible only within your booked time slots.

3.2 RECOMMENDED TUTORIALS

1. Flask installation [4]
2. Flask quick start tutorial [9]
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]

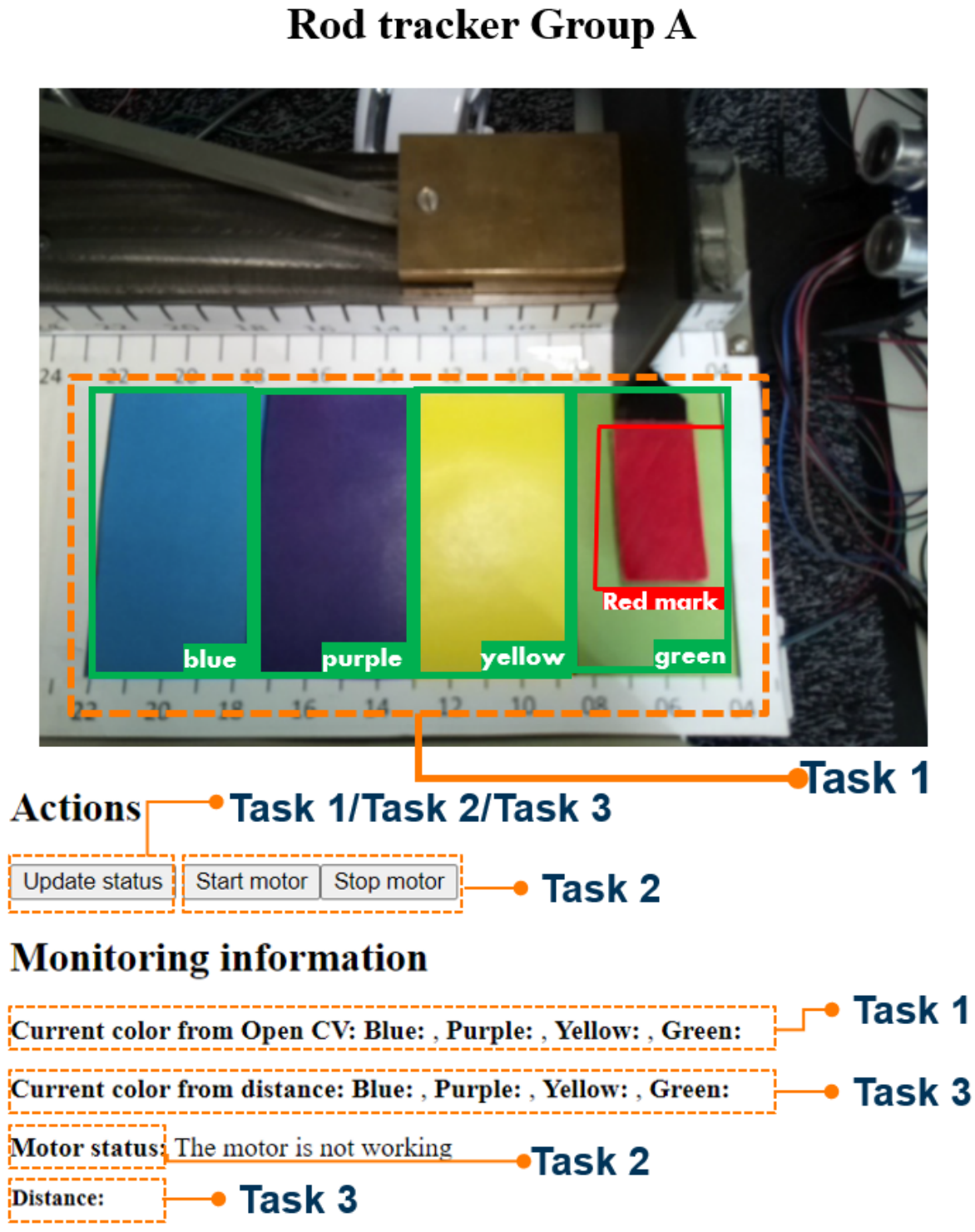


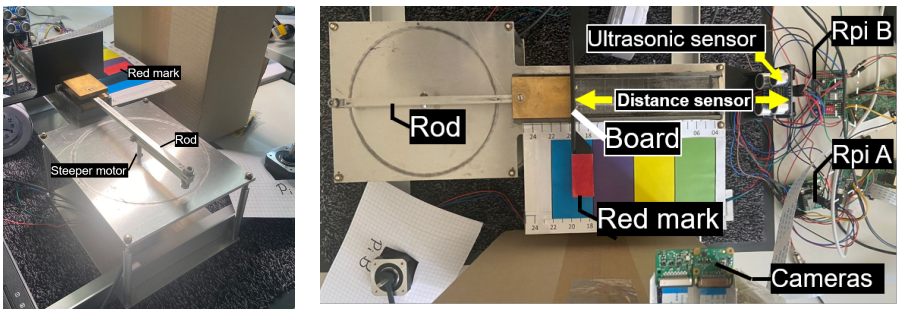
Figure 3.1: Expected result of the whole web interface and the relationship between the visual elements and each Task

4

TASK 1: MOTOR CONTROL

In this task, you can get **at most 20 points** by controlling a stepper motor to move a red mark attached to a rod (See Figure 4.1). The points are distributed as follows:

1. Allow starting the motor and move 180° or 360° **(4 points)**.
2. Every time the motor starts, it should rotate clockwise or counterclockwise randomly **(2 points)**.
3. Allow stopping the motor **(3 points)**.
4. Include buttons to start and stop the motor in the Web App **(3 points)**.
5. Update the Web App interface to show whether the motor is rotating, if so, the degrees and direction it is doing it **(2 points)**.
6. The task works locally with fake scripts **(3 Point)**.
7. Additional work **(1 to 3 points)**.



(a) View from one side (b) View from above

Figure 4.1: Stepper motor's views within the experiment set up. The stepper motor, the rod, and the attached red mark are shown.

4.1 DEVELOPMENT INSTRUCTIONS

You should employ the *task1_motor_control* folder for this task, which contains the two scripts described below. Also, 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 2.3.

4.1.1 *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 rotate 180° or 360°, 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* 180° *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.

4.1.2 *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. We will use this script to test and check the correctness of your code. Thus, do not change the content of this script.

4.2 CONTROLLING THE MOTOR VIA THE WEB INTERFACE

Remember to check Chapter 3 to get more information about the scripts mentioned below.

4.2.1 *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.

4.2.2 *The main client script*

You should replace the //... marks with your code to implement the expected behavior. Use the *requestDigitFromOpenCV* and *updateCurrentDigitOpenCV* methods (see Section 5.2.2) 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.

4.2.3 *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.

4.3 POSSIBLE IMPROVEMENTS

Some ideas to improve the project are:

- Control the motor to move until a new digit is found.
- Control the motor to move to a certain digit.

4.4 RECOMMENDED TUTORIALS

1. Virtual environments in python [17]
2. Raspberry Pi stepper motor tutorial [11]

5

TASK 2: OPENCV CONTROL

In this task, you can get **at most 22 points** by processing the video captured by the Camera module connected to the Raspberry Pi to:

1. Detect a red mark and the four colors **(6 points)**
2. Label and highlight the colors and the red mark as shown in Figure 5.1 **(4 points)**
3. Determine on which colors is the red mark currently positioned **(2 points)**
4. Every time the motor starts, it should rotate clockwise or counterclockwise based on where the red mark is **(2 points)**.
5. Update which is the current color on the Web App **(3 points)**
6. Make the task work locally with fake scripts and fake camera **(2 Point)**
7. Produce additional work **(1 to 3 points)**

When working locally you should use the provided fake Camera. Meanwhile, when working in the Raspberry Pi, you should use the frames captured with the connected camera.

5.1 DEVELOPMENT INSTRUCTIONS

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

5.1.1 *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 it is positioned. 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 to which color the red mark is pointing. Use OpenCV to track, highlight and label the red mark and the color in a frame similar to the one presented in Figure 5.1. You can use [8] as a starting point for learning how to detect colors using OpenCV.

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

   - *"[True,False,False,False]"* if the red mark is over the color green.
   - *"[False,True,False,False]"* if the red mark is over the color yellow.
   - *"[False,False,True,False]"* if the red mark is over the color fuchsia.
   - *"[False,False,False,True]"* if the red mark is over the color blue.

It is possible that the red mark overlaps two colors. In that case, the method should set two True values and two False. For instance, if the red mark is over the colors green and yellow, the method should set value of *current_color* to *"[True,True,False,False]"*.

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

3. Modification of *start_motor* in the *MotorController* class: As in Chapter 4, start the stepper motor, it should choose randomly, whether the motor should rotate 180° or 360°, but it should rotate clockwise; or counterclockwise based on the current position of the red mark:

   - Blue and yellow: random 180° or 360° *counterclockwise*
   - Purple and green: random 180° or 360° *clockwise*

   This decision of clockwise or counterclockwise should be taken when clicking on "Start motor" only. Take into account that the motor rotates 0.225° per step.

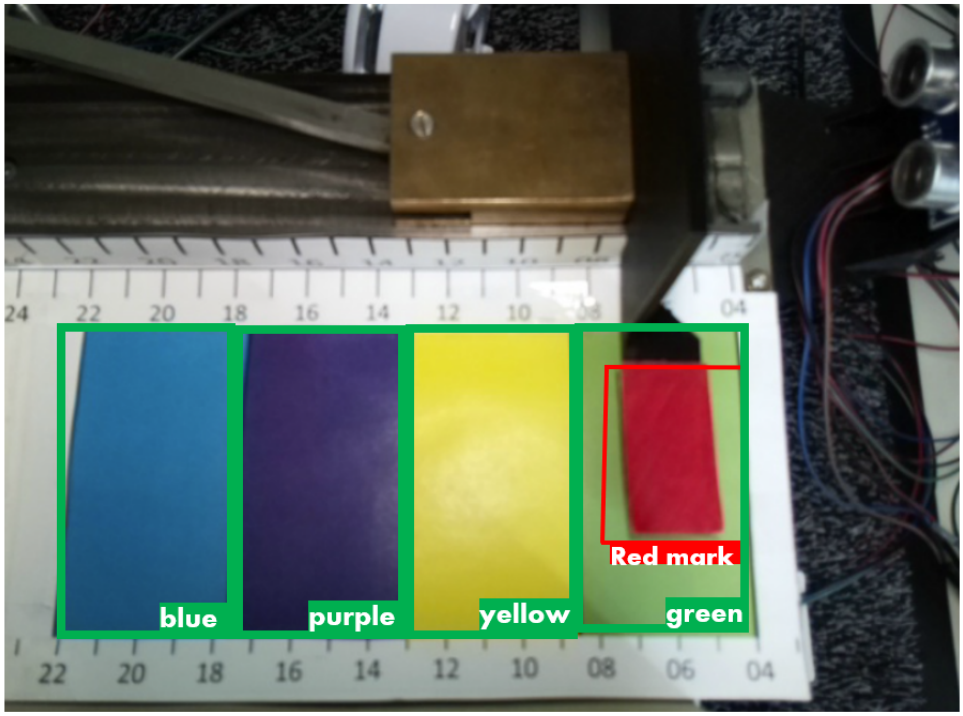


Figure 5.1: Image captured with the Camera Module connected to the Raspberry Pi B. The red mark and the colors are highlighted and labeled using OpenCV

5.1.2 *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 on your local machine.

For the Raspberry Pi, **you should import the *pi_camera.py*** so that you can capture the frames from cameras connected to the Raspberry Pis.

5.1.3 *main.py*

You should run this script to test your code. If the method *process_frame()* is implemented properly, it should behave as described in Table 5.1 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 5.1: Expected results for the test frames

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

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

5.1.4 *Testing on the Raspberry Pi*

Check the Section 2.1 to know how to access the Raspberry Pi. Run the main file of this task to test your results.

5.2 DISPLAYING THE RESULTS VIA THE WEB INTERFACE

The functionalities of the web interface for this task are already implemented, so you do not need to modify them for this task. You should use them as examples for future tasks. If you have implemented everything as indicated in the previous sessions, this should work correctly.

Take into account the recommendations presented in Chapter 3 to test the web app.

5.2.1 *The app script*

The following views are relevant to 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.

5.2.2 *The main client script*

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

1. *requestColorFromOpenCV*: this method should request the server to get the current color using the *Open CV controller*. Check this tutorial [14] to know how to handle *axios* requests in JavaScript.
2. *updateCurrentColorOpenCV*: It updates the current color based on OpenCV on the web interface.
3. *updateStatus*: This method calls the *updateCurrentColorOpenCV* to perform the update.

5.2.3 *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 should already work.
2. *monitoring_info*: This contains a *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.

5.3 RECOMMENDED TUTORIALS

1. OpenCV in Python [7]
2. Install OpenCV under environment [16]
3. Shape detection using OpenCV [8]
4. The tutorials presented in Chapter 3

6

TASK 3: DISTANCE SENSOR CONTROL

In this task, you can get **at most 20 points** by controlling an ultrasonic distance sensor to:

1. Measure the distance between the sensor and the board attached to the rod as shown in Figure 6.1 **(4 points)**.
2. Determine an accurate distance based on the median of 10 measurements **(3 points)**.
3. Detect current digit based on the final distance **(2 points)**.
4. Include a button to update the distance in the Web App **(2 points)**.
5. Update the Web App interface to show which is the current digit based on the measured distance **(3 points)**.
6. The task works locally with fake scripts **(3 Point)**.
7. Additional work **(1 to 3 points)**.

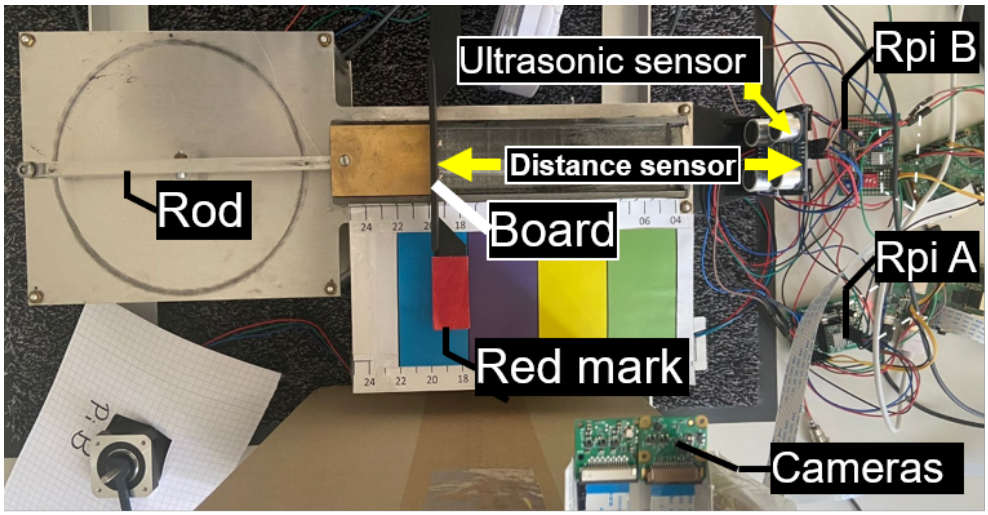


Figure 6.1: Ultrasonic sensor role in the experiment. The ultra sensor, the board, and the distance to be measured are depicted

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.

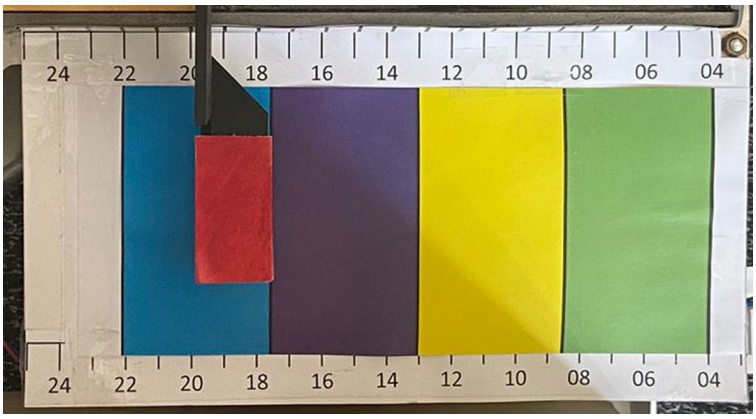


Figure 6.2: Tape measures included in the test bed.

6.1 DEVELOPMENT INSTRUCTIONS

Use the *task3_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 sensor controlling outside the Raspberry Pi as explained in Section 2.3. Remember that this script behaves randomly, and therefore, the distances will not make sense.

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 board. Also, it should set the value of the *distance* variable based on the data provided by the sensor.

   You should make **10 measurements** and the distance should be the median of those measurements.

2. *get_digit_from_distance*: It should return the value of the *digit_from_distance* variable. It should calculate the digit 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 digit.

Table 6.1: Distance ranges of each digits

DIGIT	DISTANCE RANGE
8	15 - 21 cm
3	9 - 15 cm
1	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 distance. If you have not implemented the methods properly, it will probably print *None* instead.

At the same time, this script will print "*Current digit from the sensor: [True or False, True or False, True or False]*" on the console depending on the identified digit. As you notice, it is an array of Boolean values, the first position should be True if the current digit is the **8**, otherwise, it should be False. The second position is for the **3** and the third one is for the **1**. It is possible that the red mask overlaps two digits. 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 Chapter 3 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 board to the ultrasonic sensor.
2. *get_digit_from_distance*: It should get the digit based on the distance from the board 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 *requestDigitrFromOpenCV* and *updateCurrentDigitOpenCV* methods (see Section 5.2.2) 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. *updateCurrentDigitDistance*: It updates the current digit based on the result of calling *requestDigitFromDistance*.
4. *requestDigitFromDistance*: This method should request the server to get the current digit using the *sensor_controller*. Check this tutorial [14] 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 digit 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 digit based on the calculated distance. It should behave similarly to the status done through the OpenCV controller from Task 1.

6.3 RECOMMENDED TUTORIALS

1. Virtual environments in python [17]
2. Raspberry Pi Distance Sensor using the HC-SR04 [10]

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] ProgrammerAH. *Detailed explanation of OpenCV approxpolydp() function*. Apr. 2021. URL: https://programmerah.com/detailed-explanation-of-opencv-approxpolydp-function-24674/ (visited on 03/16/2022).

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

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

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

[12] 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).

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

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

[15] *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).

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

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