Technical University of Munich

Department of Informatics

Sustainable Process Automation: Humans, Software and Mediator Pattern

Developing an Automated Cocktail Machine: Detecting the User Order

Final Report

Deniz Yüksel 03737440

Supervisor & Lecturer: Dr. Jürgen Mangler

Professor: Prof. Dr. Stefanie Rinderle-Ma

March 30, 2023

Table of Contents

1. Introduction 3
2. Requirement Analysis 3
3. Implementation 4
    3.1 Pros and Cons of the Approach 6
4. Setup Information 7
    4.1 Experiments 8
5. User Manual 10
6. References 11

1. Introduction

In the course, Sustainable Process Automation: Humans, Software and the Mediator Pattern (IN2106) [1] our purpose was to build a robot composed of several parts. The end goal is that every part works on its own but more importantly, all the parts are able to be integrated independently of each other. In this brief report, I explain the end result of my task, the algorithm I used, some pros and cons of my approach, and the experiment results and under which conditions the project works optimally. Also, documentation to run and build the project is also included in the appendix.

2. Requirement Analysis

The robot will serve drinks according to user preference. The user input will be gathered from two sources:

- A website for users to choose which cocktail they would like to order
- A physical user interface that would serve the user the drink according to which position they place the glass

My task was to implement the second bullet point, by using a USB camera that will take a bird's-eye view photo. Then, this photo would be analyzed and the user's drink order is determined. The system would comprise 2 REST endpoints, to be deployed to ProcessHub to work in an interdependent fashion. These REST endpoints were to be implemented using FastAPI [2]. The two services would include:

- Two endpoints with two paths "/" and "/{photo_id}" and methods GET. The first endpoint would be to capture a photo and save it to the current directory. The second endpoint would be to display the current photo, given the path parameter.
- A single endpoint with path "/" and method POST that would accept a URL in the body parameter, download the image in the URL, analyze the image, and return a JSON array to indicate which grid the user put their glass on. For example, if there are 3 drink choices and 2 sizes, we would present the user with a 2x3 grid, and the endpoint would return an array composed of "1" to indicate where the glass is placed, and all other elements would be "0".

Because this was not a computer vision course, I was allowed to build a system that would work under very specific conditions. These conditions were:

- The camera will take the picture in a certain lighting condition
- The background color of the grid is white
- The lines on the paper are blue
- The bottom of the glasses are marked with red dots

The task then was to identify which grid the red dot was an element of.

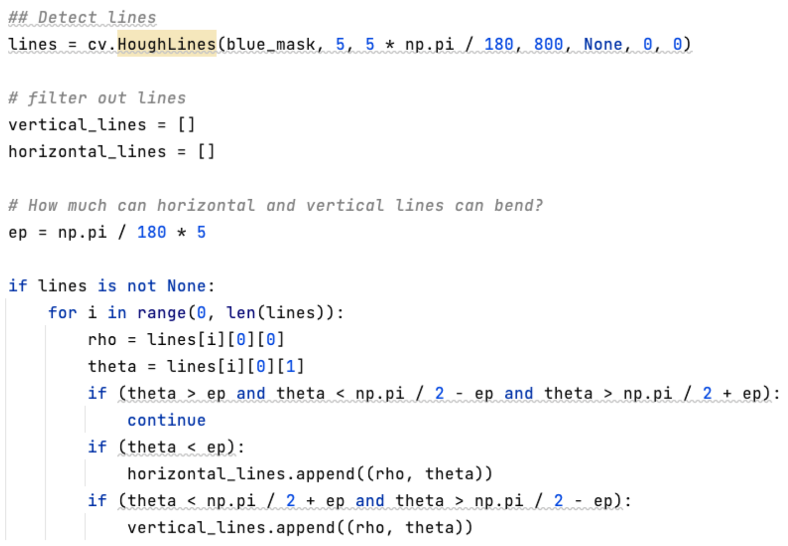
3. Implementation

First, the first REST service with 2 GET endpoints was implemented. The function names were “save” and “read_photo”. In these two functions, some additional FastAPI libraries were used. This service would use a hardware camera, so in order to run this service successfully in the ProcessHub, port forwarding is used. This allowed the USB camera to function in the local host of lehre’s machine, and this way the camera acted as if it was connected to lehre’s computer.

The second endpoint which analyzes the image had a single POST endpoint with the function name “download_and_analyze_image”. The function included other helper methods which were responsible for the following operations:

- Scaling the image: All images are to be scaled to a fixed size before being processed.
- Create blue mask: A blue mask in HSV color space is created, the mask is dilated and eroded to fill the gap between the blue pixels, as a result, the blue lines are detected given an image input.
- Detec red center: A red center mask in HSV space is created to find the red center in a given image.
- Test line: A method that evaluates a single line, if the point is on the left-hand side of the line. This method is called for every detected blue line.

In order to detect the lines on the image, Hough Line Transform [3] is used. The implementation for the detection of horizontal and vertical line sets is shown below. There is an epsilon value of 5 degrees to be chosen to tolerate how much a line can bend.



```
## Detect lines
lines = cv.HoughLines(blue_mask, 5, 5 * np.pi / 180, 800, None, 0, 0)

# filter out lines
vertical_lines = []
horizontal_lines = []

# How much can horizontal and vertical lines can bend?
ep = np.pi / 180 * 5

if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        if (theta > ep and theta < np.pi / 2 - ep and theta > np.pi / 2 + ep):
            continue
        if (theta < ep):
            horizontal_lines.append((rho, theta))
        if (theta < np.pi / 2 + ep and theta > np.pi / 2 - ep):
            vertical_lines.append((rho, theta))
```

Figure 1: Detecting lines with Hough Line Transform

The picture below shows the steps of detecting the red dot, namely first applying the mask and detecting the liens, and then labeling the lines and detecting the red center:

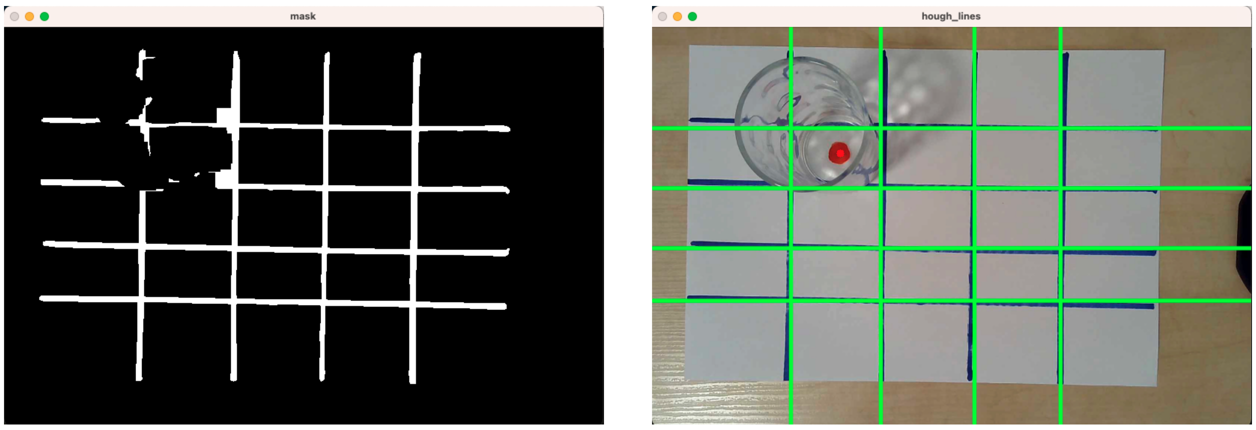


Figure 2: The process of detecting the lines and the red center

3.1 Pros and Cons of the Approach

The implementation I pursued involves classical computer vision methods, and there are no machine learning models included in this project. Hence, the code works only under certain conditions. These conditions and the experiments will be explained further in the report.
There were two approaches for detecting the red dot in a specified grid. The method proposed by Professor Mangler involved support vector machines [4] and creating a dataset with many pictures, containing a glass at a specific position on the white paper. This method required processing all the grids in the picture (in our case 6) separately, to decide if that grid contained the glass or not. The pros of this approach are that the machine-learning model would become more stable by having more trials, and more data would be generated while people come and use our bartender. This approach also didn't require detecting a red dot in the first place, so only a glass detection would be enough. E.g. the picture would be a white sheet of paper, or a white sheet of paper with a glass in it, so it would not be difficult to detect if the current picture had a glass in it (every grid is processed separately). The cons of this approach are that grids are fixed, therefore the menu cannot tolerate a change. For example, if we were to add another drink, the grid would now be a 2x4 one.
My approach was different, in the case that it did not involve any machine learning methods. My motivation to pursue my approach stemmed from various reasons. Firstly, despite the fact I am a software engineering student who took machine-learning and deep-learning courses, I never took any computer vision courses at TUM, and I wanted to learn about them. Also, I wanted to create a solution, maybe not too reliable as the first one, but an extendable one. So if the menu was to change, my solution would be able to adapt to it. Moreover, I wanted to use math and analytical geometry instead of the hard labor of data labeling. A table for comparisons of different aspects of the two implementations is given below:

Aspect \ Method	Support Vector Machines	Classical Computer Vision Methods
Manual Labor	Labeling dataset of images	Sticking red dots under glasses
Implementation	Process predefined grids separately in a single image	Detect lines in dynamic grids, process the whole image

Reliability	Reliable after training, and given a single condition (menu is fixed), gets more reliable with usage	Reliable given more conditions (lighting, tilting of the image, camera angle, glass height), fixed reliability
Training	Training is required	No training is required
Extendability	Menu cannot change (grid size is fixed)	Menu can change, the code can adapt to different menus

Table 1: Comparisons of Different Approaches

4. Setup Information

In order to make the correct setup, some requirements are needed:

- A hardware USB camera,
- A transparent glass
- A red dot stitched in the bottom of the glass
- A white paper with a desired number of grids drawn with a blue board marker

The example setup looks like the one in below:

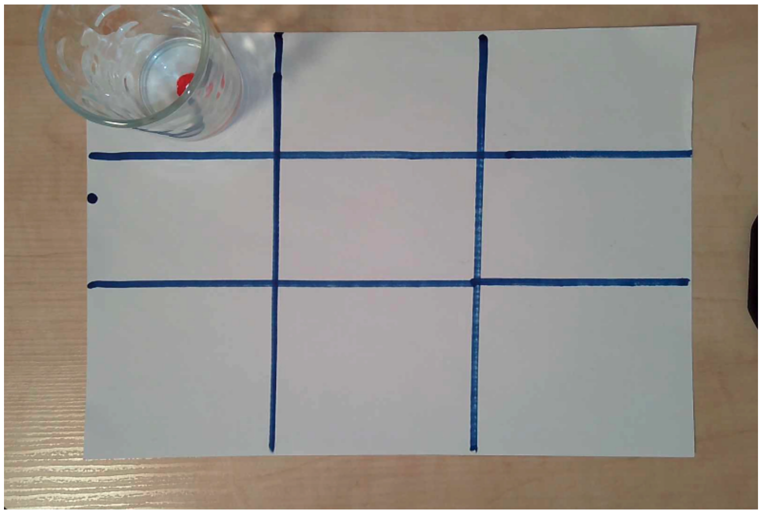


Figure 3: Sample setup

After completing the setup as in Figure 3, the system is ready to function

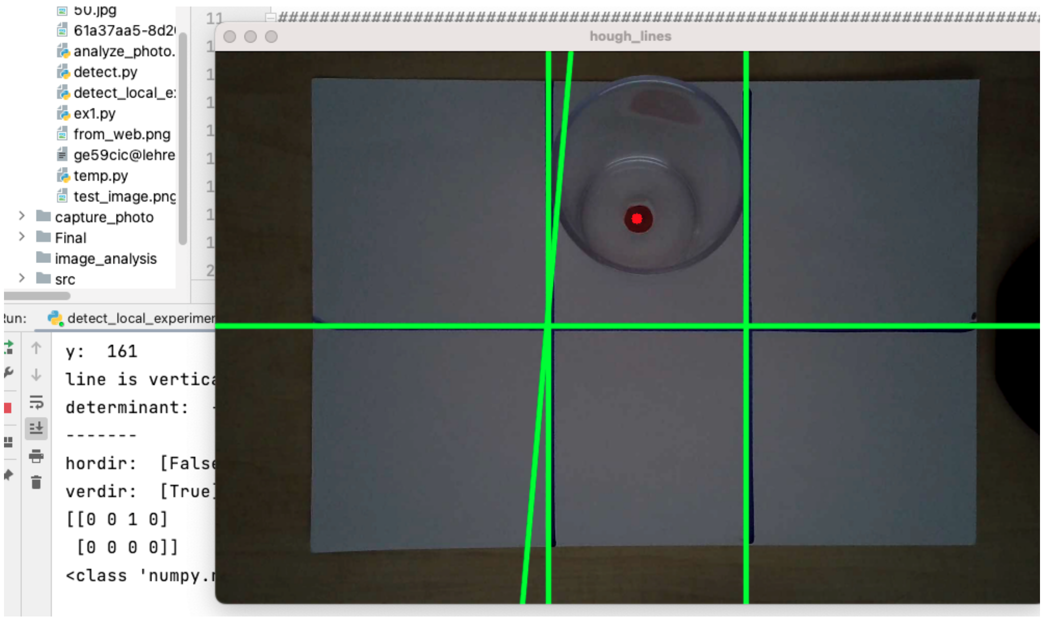
4.1 Experiments

This section of the report further evaluates the experiments under which conditions the systems work, particularly the lighting. The Light Meter App [5] on the AppStore is used to do measurements, and a light diffuser is built on the front camera of the iPhone. Table 2 depicts the experiment results in which the light value, description of the medium and system functionality is elaborated.

Light Value (lux)	Description	System function
1.2	Dark room	Camera not able to focus
6.5	Dark room	Extra line detection, not functioning correctly (see Figure 4)
18	Dark room with windowblinds slightly open	System functions, camera needs to be focused before taking the picture (see Figure 5)
25	Dark room with a computer screen in front of the setup	System functions properly
40	Room with a lamp	System functions properly
100	A room with daylight entering the medium	System functions properly

Table 2: Results of the Experiment

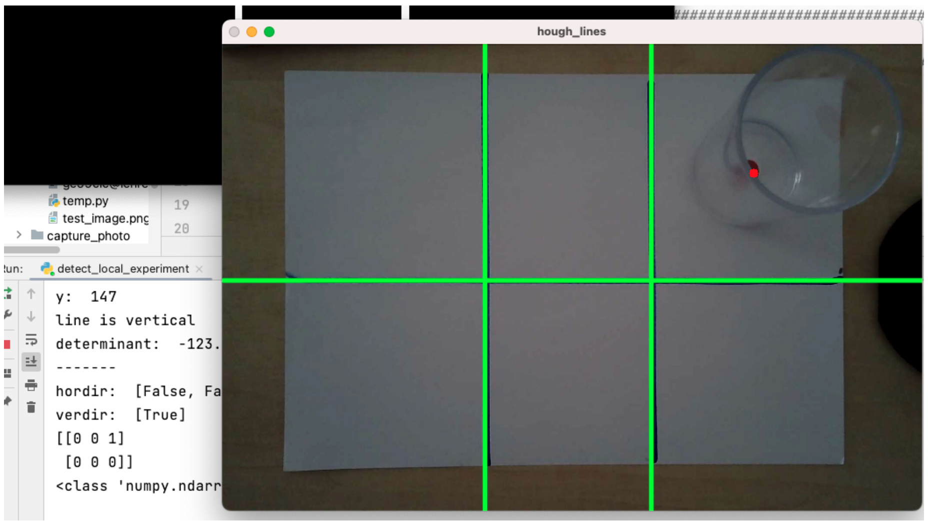
These conditions are valid under the assumption that the system setup in Section 4 is done properly. A setup with lines that have more than 5 degree angle with the horizontal and vertical axis would not function, because those lines would not be considered horizontal and vertical lines according to the implementation. Figure 4 shows the experiment result with a light value with 6.5 lux, which took place in a dark room.


```
50.jpg
61a37aa5-8d2
analyze_photo.
detect.py
detect_local_e:
ex1.py
from_web.png
ge59cic@lehre
temp.py
test_image.png
capture_photo
Final
image_analysis
src
```

```
y:  161
line is vertica
determinant:  -
-------
hordir:  [False
verdir:  [True]
[[0 0 1 0]
 [0 0 0 0]]
<class 'numpy.n
```

Figure 4: Sample setup

As can be seen clearly in the screenshot, our grid dimension is 2x3. However, in the dark detecting the blue color is much more difficult. In the result, there is an additional grid detected where the erroneous line is present, therefore it results in a 2x4 grid which is not correct. On the other hand, the setup works fine in a dark room with windowblinds slightly open, so a little amount of daylight is present in the medium. Figure 5 shows the results of the trial with 18 lux which system starts to function. However it is unreliable.


```
temp.py
test_image.png
capture_photo
```

```
y:  147
line is vertical
determinant:  -123.
-------
hordir:  [False, Fa
verdir:  [True]
[[0 0 1]
 [0 0 0]]
<class 'numpy.ndarr
```

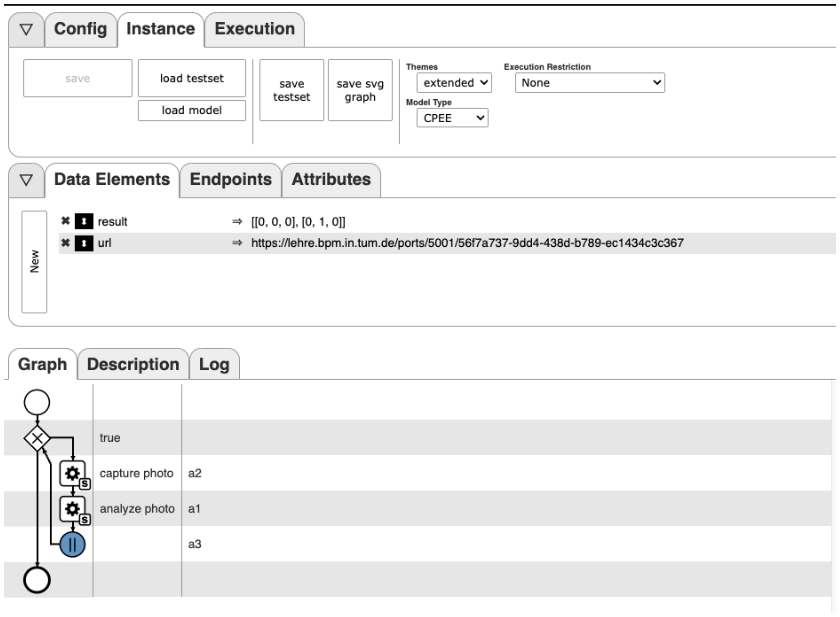
Figure 5: Results of the trial with 18 units of lux

5. User Manual

This project is implemented with Python, so make sure Python and pip are installed on the user's computer.

Running the project in Lehre's server in order to run it on the process engine requires two steps. First, the RESTful service that analyses the picture needs to be copied to the lehre's machine with the command "scp". Also, the file "requirements.txt" must also be copied in order to install all the OpenCV and other dependencies required to run the code. After running the service, the first service which captures the picture can be run using port forwarding from the localhost to the lehre's server in order to run the hardware camera successfully.

The screenshot below shows a sample execution from the BPM engine:



Themes: extended

Execution Restriction: None

Model Type: CPEE

```
result ⇒ [[0, 0, 0], [0, 1, 0]]
url    ⇒ https://lehre.bpm.in.tum.de/ports/5001/56f7a737-9dd4-438d-b789-ec1434c3c367
```

true	
capture photo	a2
analyze photo	a1
	a3

Figure 6: Graph of two Processes in the Process Engine

6. References

[1] München, T.U. (no date) *LV - Detailansicht*, *Course - Detailed View - TUMonline - Technische Universität München*. Available at: https://campus.tum.de/tumonline/wbLv.wbShowLVDetail?pStpSpNr=950634777&pSpr acheNr=2 (Accessed: April 2, 2023).

[2] *FASTAPI*. Available at: https://fastapi.tiangolo.com/ (Accessed: April 2, 2023).

[3] *Hough Line transform* (no date) *OpenCV*. Available at: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html (Accessed: April 2, 2023).

[4]*Support Vector Machines* (no date) *scikit*. Available at: https://scikit-learn.org/stable/modules/svm.html (Accessed: April 2, 2023).

[5] GmbH, L.I. (2021) *Light meter LM-3000*, *App Store*. Available at: https://apps.apple.com/app/id1554264761 (Accessed: April 2, 2023).