

Introduction

In this project we developed a web-based system that loads an image that includes "image-on-wall".

Then the system allows a user to draw a bounding box around an image on the wall and the system detects the accurate boundary of the image using edge detection and Hough transform.

We used OpenCV , scikit-image: image processing in Python, NumPy, flask, SQLite3 HTML and Bootstrap.

Code Explanation

The code consists of two parts: the photo editing in Python language, and the web-based application in Python, HTML languages and SQLite3 database.

Photo editing (runAlg.py file):

The code begins with drawing a bounding box around the image with a mouse pull from the top left corner to the bottom right corner and cropping the image according to the mouse movements. For this purpose, we used a code from Life2Coding which contains cv2 events.

Then, cv2.Canny- an edge detection algorithm- is called and returns a Grayscale edge detected image.

Later, to detect the frame, we combined two Hough Transform algorithms:

1. hough_line and hough_line_peaks from skimage.
2. cv2.houghLines.

We started with the first, defined lines from the information the algorithm returned (angle and distance for each line), then we tested each line whether it was at an appropriate angle, and is a candidate to form one of the walls of the frame. Each line we found appropriate we inserted to an array, then checked for each pair of lines if their intersection point is in the image range and if so, added the point to another array. We continued this process by increasing the variable 'slope' that constitutes the upper boundary of the angle of the line, until there are points in the intersection points array.

Then, we dealt with edge cases by calling the second algorithm- cv2.houghLines- with different values for the parameter 'max_slider'. The cases we wanted to address were:

1. We found less than 4 points in the intersection array.

- After choosing the 4 candidates frame points (which we assumed will match to the 4 corners of the original image), we had to check if the 4 points are proper and can suit the frame's corners, by checking if there is a pair that are too close to each other- in case the 4 lines that were discovered are on the same side of the image.

Finally, after we've got 4 points that are suitable corners, we went through all the 6 possible lines formed from those points, chose the 4 that construct the frame and called cv2.lines to draw the lines on the image.

web-based application (app.py, main.py, HTML files and SQLite3 database):

The app.py file is responsible for creating flask object and building the app. The flask object is responsible for creating the HTML pages, using the templates files- upload.HTML, uploaded.HTML and framed.HTML- enable the user several functionalities, like choosing an image, displaying, going back to home page, running the photo editing algorithm on an image and more.

main.py file contains definitions and applications for the functionalities, routing, redirecting and connecting this web application with our frame detection algorithm by calling 'run' function from runAlg.py file.





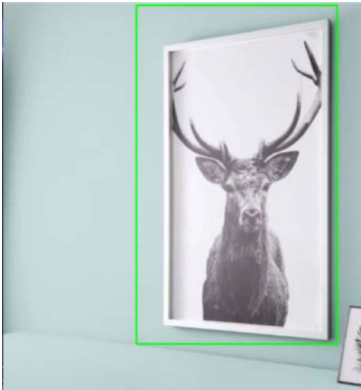

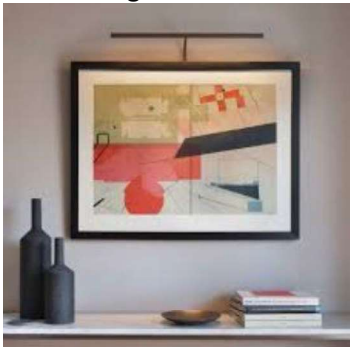


In addition, we manage a database that contains 3 types of information for each image: KEY_NAME (which is unique), KEY_IMAGE- binary representation of the image - and KEY_COORDS- list of lists, each list represents a polygon (that the algorithm has been tested on) from the image by 4 absolute points of its frame's corner. We emphasize that there can be no two images with the same name in the same folder. For each new photo the user chose to submit, there is a check whether the image is in the database. If not, the binary representation of the image is inserted with its corresponding name.

Partial photo of the database for illustration:

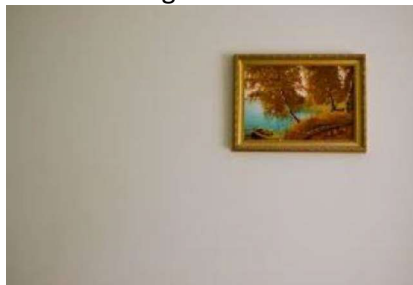
KEY_NAME		KEY_IMAGE	KEY_COORDS
Filter	Filter	Filter	
1	image_birds.jpg	BLOB	[[[13, 16], (289, 16), (289, 189), (13, 189)]]
2	image_blur.jpg	BLOB	[[[86, 85], (396, 33), (396, 420), (86, 386)]]
3	image1.jpg	BLOB	[[[249, 61], (582, 61), (578, 507), (253, 507)]]
4	image2.jpg	BLOB	[[[315, 84], (647, 31), (642, 665), (319, 637)]]
5	image3.jpg	BLOB	[[[77, 65], (342, 65), (342, 272), (77, 272)]]
6	image5.jpg	BLOB	[[[95, 59], (285, 59), (284, 197), (96, 197)]]
7	image7.jpeg	BLOB	[[[93, 60], (304, 60), (302, 332), (91, 332)]]
8	image8.jpg	BLOB	[[[72, 86], (326, 86), (326, 309), (72, 309)]]
9	image9.jpg	BLOB	[[[514, 48], (866, 48), (862, 506), (510, 506)]]
10	image10.jpg	BLOB	[[[70, 28], (326, 28), (326, 201), (70, 201)]]
11	image11.jpg	BLOB	[[[8, 18], (232, 18), (232, 309), (8, 309)]]
12	image12.jpg	BLOB	[[[49, 75], (220, 29), (223, 327), (49, 348)]]
13	image13.jpg	BLOB	[[[91, 193], (276, 81), (280, 497), (87, 614)]]
14	image15.jpg	BLOB	[[[582, 112], (977, 84), (973, 437), (571, 456)]]
15	image14.jpg	BLOB	[[[81, 153], (434, 153), (294, 454), (81, 454)]]
16	image20.jpg	BLOB	[[[499, 111], (710, 111), (710, 416), (367, 416)]]
17	image6.jpg	BLOB	[[[90, 222], (370, 153), (431, 401), (151, 470)]]
18	New_mona-lisa.jpeg	BLOB	[[[265, 39], (416, 39), (416, 257), (267, 257)]]
19	New_mona-lisa2.jpeg	BLOB	[[[42, 90], (136, 90), (137, 210), (43, 210)]]

Detection Results

In this section we will present original photos, each one with the mouse marks and the result – the image cropped with the frame detected.

<p>Original Photo</p> 	<p>Mouse Marked Photo</p> 	<p>Frame Detected Image</p> 
<p>Original Photo</p> 	<p>Mouse Marked Photo</p> 	<p>Frame Detected Image</p> 
<p>Original Photo</p> 	<p>Mouse Marked Photo</p> 	<p>Frame Detected Image</p> 

Original Photo



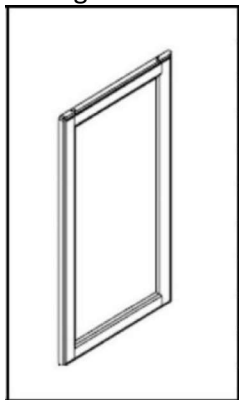
Mouse Marked Photo



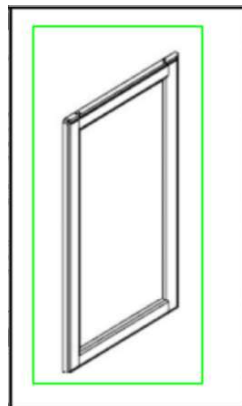
Frame Detected Image



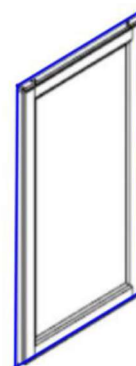
Original Photo



Mouse Marked Photo



Frame Detected Image



Original Photo



Mouse Marked Photo



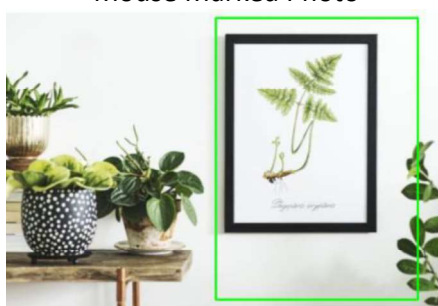
Frame Detected Image



Original Photo



Mouse Marked Photo



Frame Detected Image



Original Photo



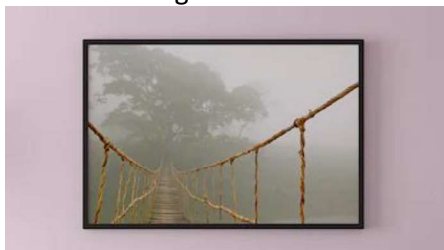
Mouse Marked Photo



Frame Detected Image



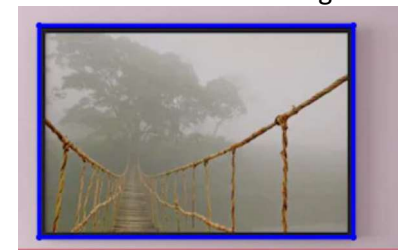
Original Photo



Mouse Marked Photo



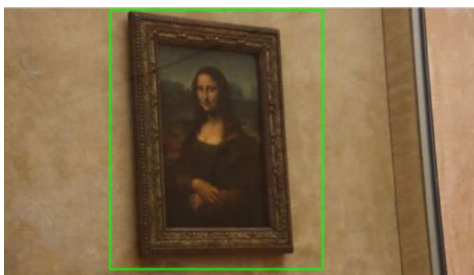
Frame Detected Image



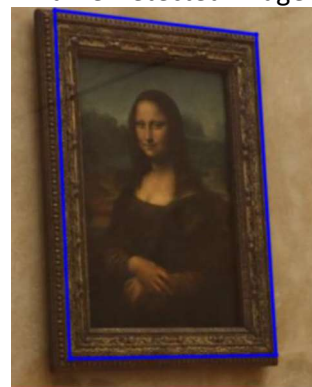
Original Photo



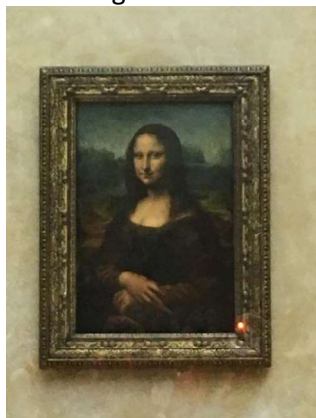
Mouse Marked Photo



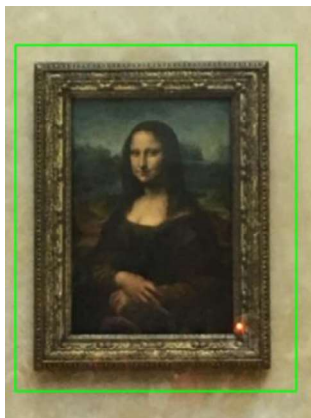
Frame Detected Image



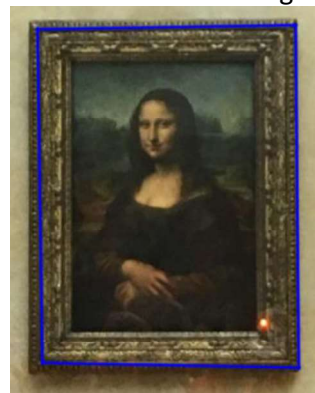
Original Photo



Mouse Marked Photo



Frame Detected Image



Original Photo



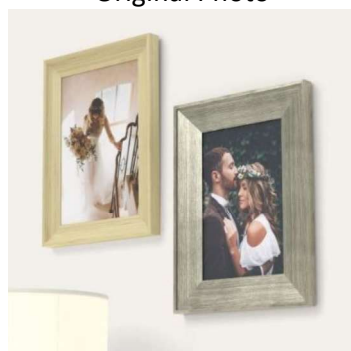
Mouse Marked Photo



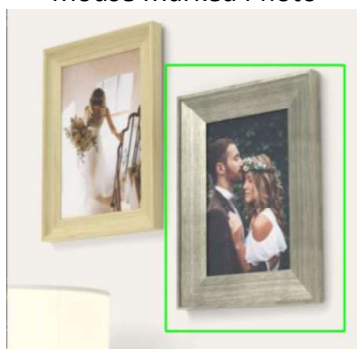
Frame Detected Image



Original Photo



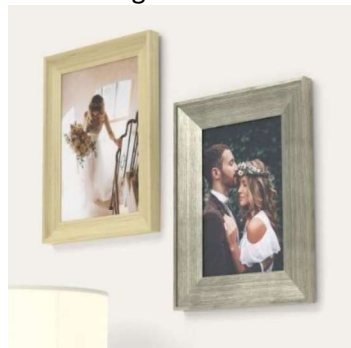
Mouse Marked Photo



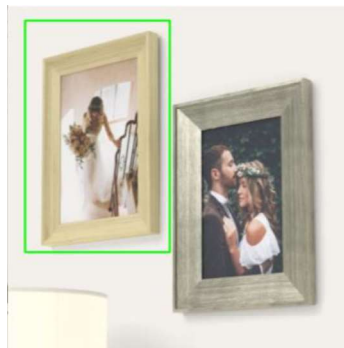
Frame Detected Image



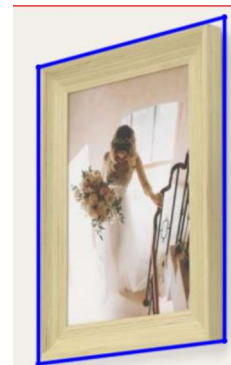
Original Photo



Mouse Marked Photo



Frame Detected Image



Original Photo











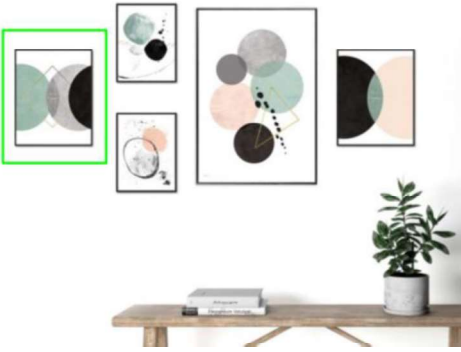



Mouse Marked Photo



Frame Detected Image



<p>Original Photo</p> 	<p>Mouse Marked Photo</p> 	<p>Frame Detected Image</p> 
<p>Original Photo</p> 	<p>Mouse Marked Photo</p> 	<p>Frame Detected Image</p> 
<p>Original Photo</p> 	<p>Mouse Marked Photo</p> 	<p>Frame Detected Image</p> 
<p>Original Photo</p> 	<p>Mouse Marked Photo</p> 	<p>Frame Detected Image</p> 

Original Photo



Mouse Marked Photo



Frame Detected Image



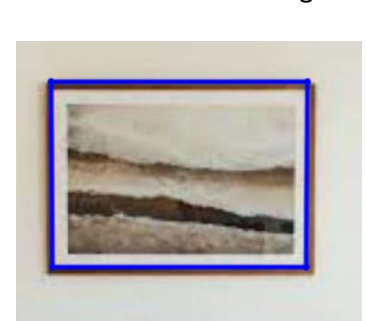
Original Photo



Mouse Marked Photo



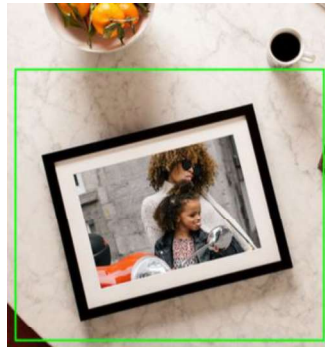
Frame Detected Image



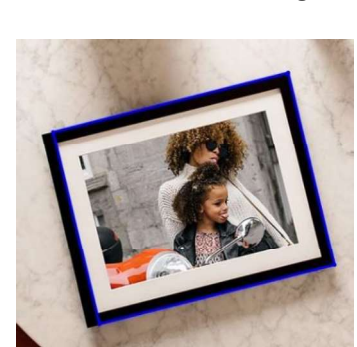
Original Photo



Mouse Marked Photo



Frame Detected Image



Discussion & Conclusion

Throughout the work on the project, we encountered several difficulties and had to deal with edge cases.

For example, for some images, too many lines were detected, and that prevented the continuation of the code from working properly, or lines that crossed the image and were not suitable to be frame lines were selected. We overcame these obstacles through consultation with our project advisor, tried different scenarios on the code and realized the Hough Transform algorithms behave differently, so we found the solution by combining them. Moreover, we had trouble converting the code to the web interface, and to overcome this we used flask, which is best suited for the use and running of Python code.

In conclusion, we believe our project is successful and performs frame detection well, we learned a lot in the field of image processing and prepared a project in a good and satisfactory way.

How To Run the Code?

Before running, you must insert all images you would like to test to the "images" folder.

A link to our images :

https://drive.google.com/drive/folders/1QWH5S7ebO_Jj5MP6GPoXjE-jsJreElid?usp=sharing

In addition, you must import the following libraries:

- cv2
- sqlite3
- NumPy
- scikit-image
- math
- os
- flask
- werkzeug

1. run the program by executing the main file.
2. press on the site's IP from the terminal (local host).

```
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

3. Press '**Choose File**' and choose an image from the "images" folder.

Upload Image

Please chose an image to uploads

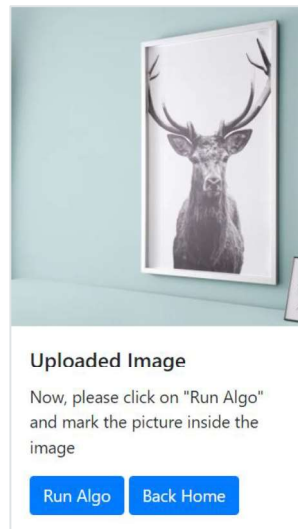
Choose File

No file chosen

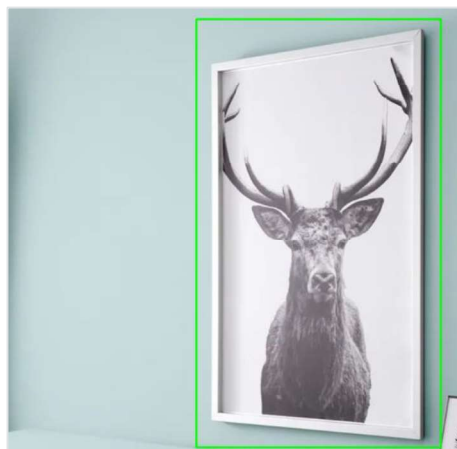
Submit

Image frame detection

4. Press '**Submit**'.
5. Now the image is shown, and you have an option to run the algorithm on this image or go back and choose other one.



6. Choose '**Run Algo**'. A screen pops up to mark a frame in it (drag the mouse from top-left to bottom-right).



7. After marking, the window closes, and the output (the framed photo) appears. The output is saved to "static/uploads". If you want to save its coordinates to the database, you should press '**Submit frame**'. Otherwise press '**Back Home**'.



8. To shut down the app, press CTRL+C in the terminal.

Bibliography

1. <https://www.life2coding.com/crop-image-using-mouse-click-movement-python/>
2. https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
3. https://github.com/scikit-image/scikit-image/blob/main/skimage/transform/hough_transform.pyx
4. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html