

RSEC Project Report

Drawing with Gretchen

Group project by:

Myriam Eschenlohr

Andriy Popov

Fabio Schmidt

Date: 17 February 2024

Abstract

This report covers the different ideas and the processes which inspired the direction of the project, and which helped to reach the final result.

The end goal was to create a “live” drawing board: Gretchen recognizes a pen and follows it, when necessary, while the pen’s movements are drawn on a separate view. In the end, the drawing is saved upon the termination of the program. Drawing and tracking is possible through the motors of the Gretchen unit itself and by the python libraries OpenCV and NumPy.

The text is divided into multiple sections. A short introduction presents the initial ideas and use cases of the work. The next section introduces the technologies which were implemented. Subsequently the present implementation is described. The final part discusses reflections made throughout the work, summarizes the project development and any possible future improvements are evaluated.

Most initial goals were reached. However, various additional features, such as an additional object detector for a drawing utensil, handwriting detection and recognition, and interaction with Gretchen/the drawing board through hand gestures, were either scrapped due to time constraints, or general limitations of the hardware used.

The implemented detection of the pen uses a simpler approach, adjusting the ball detection class provided within the course material.

Introduction to the Project and its Goal

The main goal is to use Gretchen and its camera to create a drawing board. The camera should track a hand or a distinguishable object, such as a pen, and use this input to „draw“ or fill up spaces on a 2D plane.

Ideas

For the drawing utensil to be tracked, it needs to be detected. The shape or color of an object could be used as potential parameters for said recognition. Color ranges in the HSV spectrum can be isolated to facilitate the easier detection of a green cap of a pen (as an example). Background separation is also a technique / library which can aid in detecting the foreground or motion in a frame. Contour drawing and filters will get rid of noise and make the object more distinguishable.

For the actual drawing, a representation of the detected objects and their coordinates will be saved in an array. Those representations of a place in a 2D space will be filled or used to draw. This can be done by displaying those objects (circles) in cv2 or drawing lines between them. Alternatively, a canvas can be used to represent our frame, which will be filled out in the spaces where the object is detected. This is potentially more efficient than using cv2 to redraw every image of the camera (with a for-loop for example).

Use Cases

The likely use case for the project is similar to a drawing board or a virtual graphics tablet. It can potentially be used for note taking, drawing mind maps or as a whiteboard for presentations. A more direct example would be, when a tutor writes text on a blackboard. In such a case the student would use the time to think and learn instead of focusing on writing everything down.

Other use cases for the tracking and drawing are augmented reality or tracking something in a 3d environment. Or even tracking the paths of objects or people in order to create heatmaps or data of the paths they use.

Technologies Used

The technologies that are used for Gretchen are the camera and ROSEnvironment. The OpenCV library can be used extensively for tracking, framing and drawing. Potentially we can use a canvas in order to draw on it and overlay it over the preview frame. A function called `cv2.findContours()` will detect contours and `cv2.inRange()` will make it easier to detect certain colors within a range e.g. green. The technologies are implemented with Python.

NumPy is used for an array of 1s and 0s to represent the canvas. NumPy is also used for certain transformations of the coordinates.

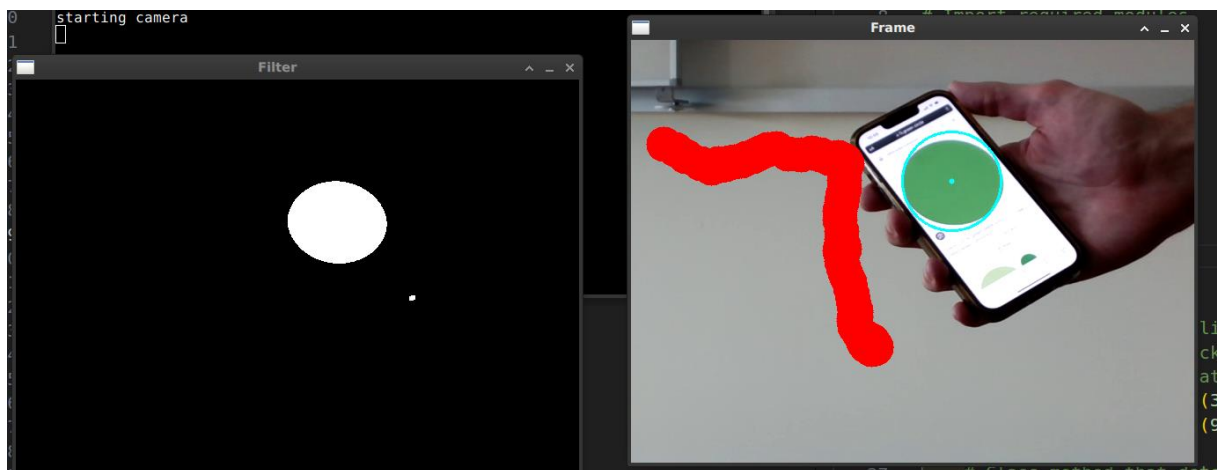
YOLO (You Only Look Once) is an object detection model. In this case it can be trained on a green pen in order to detect it more consistently. For text detection the same can be used.

Implementation

Features were incrementally implemented into the project, and it was based on the skeleton.py template, which was provided by the tutors. It contains some basic code including robotics and camera functions.

First Iteration: Proof of Concept

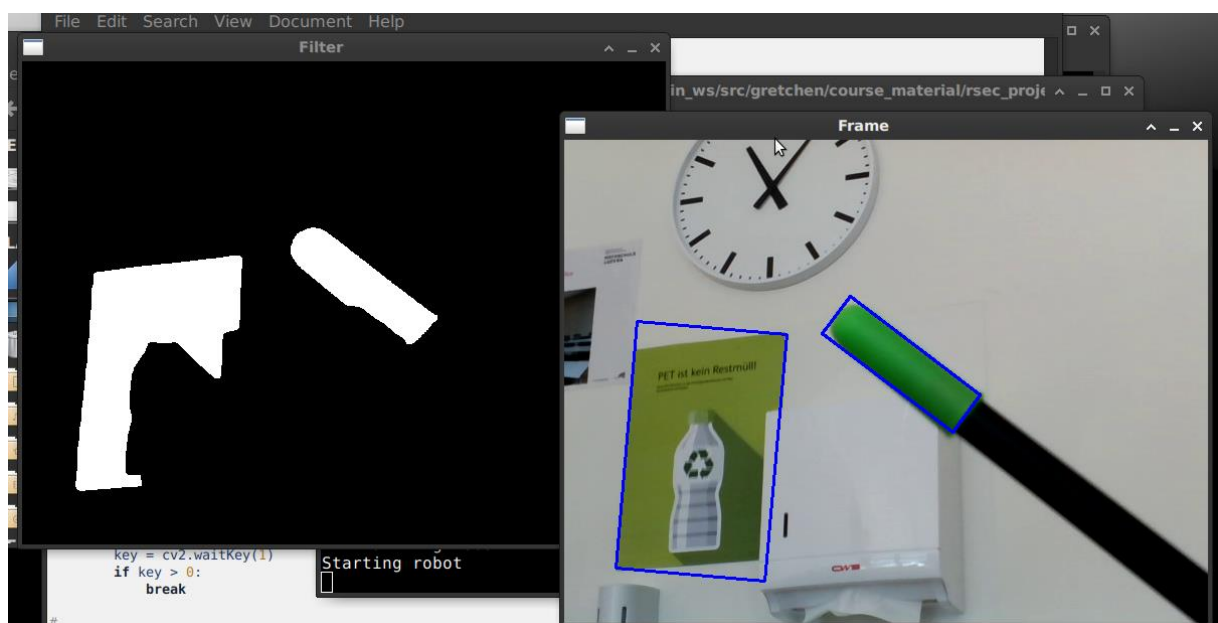
The first step is to use code from example_2_detect_ball.py to detect a circle with the camera. OpenCV is used to make a point at the center of the detected circle and draw a closed circle around it. The locations of the centers are stored in an array. This shows that it is possible but likely will be subject to optimization problems because it's running inside a VM.



The drawing (red part) is choppy and gets slower the longer it is running. The reason is that the drawing part was inside a for-loop. The for-loop draws every single recorded point for every iteration; thus it uses way too many resources quickly.

Detecting different shapes

The example_ball_detector.py class is also a good baseline for our object detection. The object of interest is a hand or a pen. The color of interest is green for better detectability.

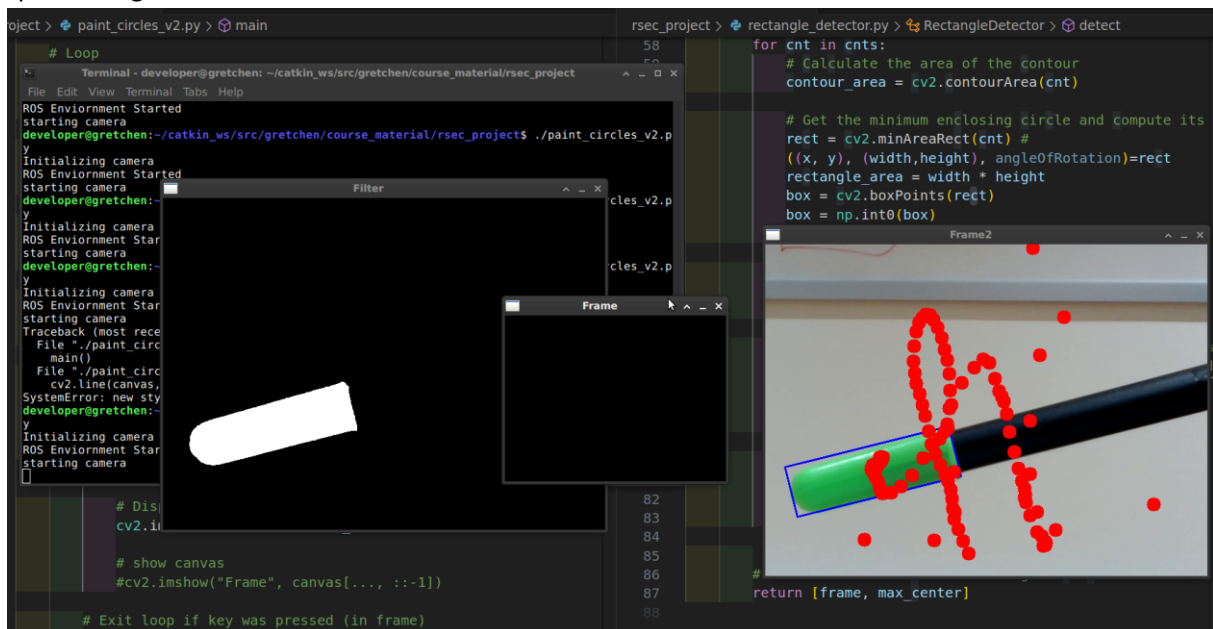


Optimization

To draw effectively it needs to have a high or at least consistent framerate. Utilizing a canvas can potentially speed the drawing process up and use less resources. A comparison of RGB channels is used to bundle 3 channels into 1. The canvas can be seen as a black and white overlay or an array of black and white pixels.

Second Iteration: Further Optimization and Text Detection

After combining the optimized drawing script with the rectangle detector, we can use a green pen to draw markings in the frame. The framerate has increased and doesn't slow down over time. This means smoother edges and not only more drawing points but also more recorded points for further processing.



```
# Loop
# Terminal - developer@gretchen: ~/catkin_ws/src/gretchen/course_material/rsec_project
File Edit View Terminal Tabs Help
ROS Environment Started
starting camera
developer@gretchen:~/catkin_ws/src/gretchen/course_material/rsec_project$ ./paint_circles_v2.py
y
Initializing camera
ROS Environment Started
starting camera
developer@gretchen:~/catkin_ws/src/gretchen/course_material/rsec_project$ ./paint_circles_v2.py
y
Initializing camera
ROS Environment Started
starting camera
developer@gretchen:~/catkin_ws/src/gretchen/course_material/rsec_project$ ./paint_circles_v2.py
y
Initializing camera
ROS Environment Started
starting camera
Traceback (most recent call last):
  File "/catkin_ws/src/gretchen/course_material/rsec_project/paint_circles_v2.py", line 10, in <module>
    cv2.imshow("Frame", canvas[::1])
  File "/usr/lib/python3.10/site-packages/cv2/imshow.py", line 10, in imshow
    raise SystemError("new style")
SystemError: new style
developer@gretchen:~/catkin_ws/src/gretchen/course_material/rsec_project$ ./paint_circles_v2.py
y
Initializing camera
ROS Environment Started
starting camera
# Dis
cv2.i

# show canvas
#cv2.imshow("Frame", canvas[::1])

# Exit loop if key was pressed (in frame)

rsec_project > rectangle_detector.py > RectangleDetector > detect
for cnt in cnts:
    # Calculate the area of the contour
    contour_area = cv2.contourArea(cnt)

    # Get the minimum enclosing circle and compute its
    rect = cv2.minAreaRect(cnt) #
    ((x, y), (width,height), angleOfRotation)=rect
    rectangle_area = width * height
    box = cv2.boxPoints(rect)
    box = np.int0(box)

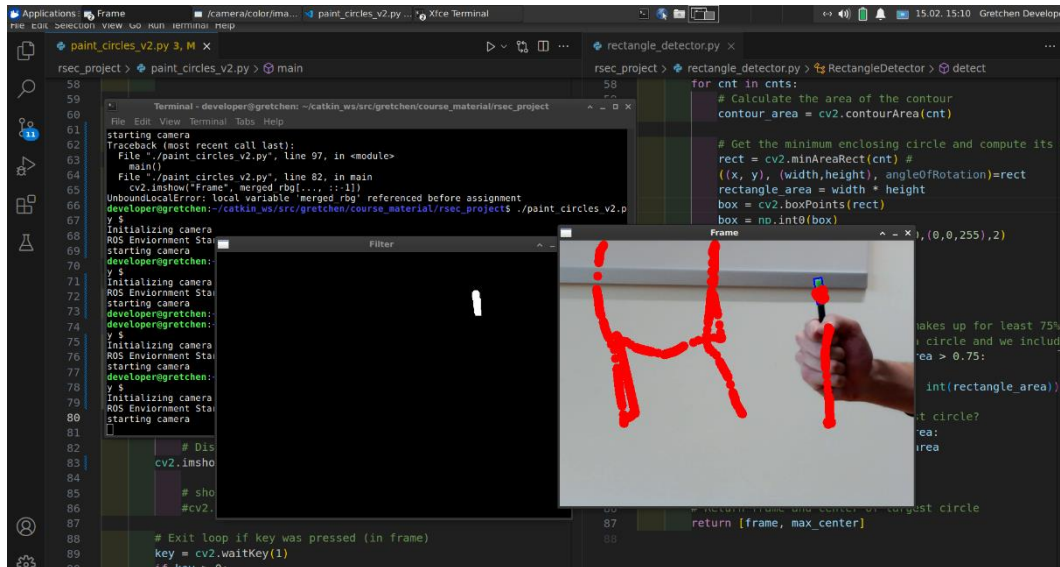
    #
    return [frame, max_center]
```

Text Detection

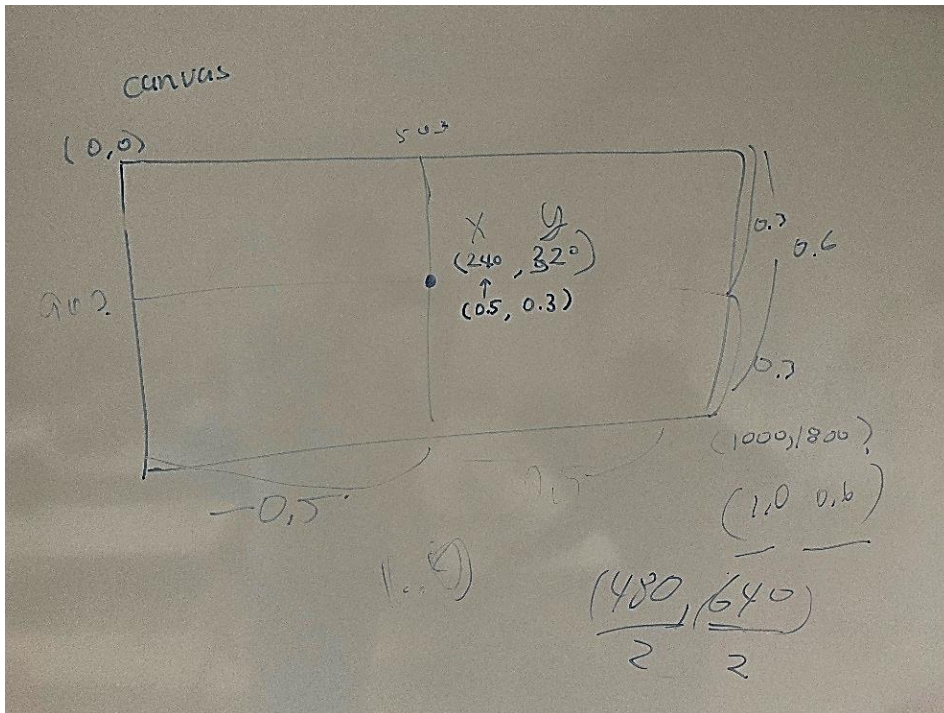
PyORC can be used as an additional library to detect text within the frame. For handwritten texts trained YOLO weight. Google colab is used for the training.

Third Iteration: Lines and more polish

The `cv2.line()` function is used to draw lines between the dots. The dots are stored in a queue of up to two spaces. Additional optimization for responsiveness. Precision tracking of the pen / rectangular green object.



Last Iteration: Bigger Canvas and Saving the Drawing



For a bigger drawing board, we transform Gretchen's coordinate system to the canvas coordinates. The camera's initial position is then centered in the middle of the bigger canvas. The end result is combined with the robot's ability to move when the pen is detected in the outer edges of the frame.

A final line of code was added, which defines the `cv2.imwrite()` function. Thanks to this function, a screenshot of the canvas is taken every time the user terminates the program (*if key > 0*).

Due to time constraints, the function to draw lines between the drawn circles and the “reality overlay” (show the drawing on the camera feed) are not implemented in the application drawing on a larger canvas. There were errors in the execution of both functionalities due to the change of the canvases size. Hence, the `cv2.line()` function was removed in the final implementation.

Results

The resulting files are the *rectangle_detector.py* and the *writing_detection_large_canvas.py*.

Name	Last commit	Last update
__pycache__	semi final stage	23 hours ago
archived	changed folder structure	22 hours ago
backup	clean up 2	16 hours ago
.README.md.swp	clean up 2	16 hours ago
+++ README.md	clean up 2	16 hours ago
rectangle_detector.py	Update rectangle_detector.py	1 day ago
screenshot.jpg	clean up 2	16 hours ago
writing_detection_large_canvas.py	clean up 2	16 hours ago

Any files, which were used as prototypes or can be seen as unfinished assets, can be found in the *archived* folder.

main ▾

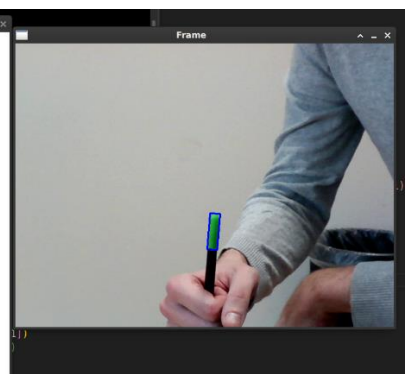
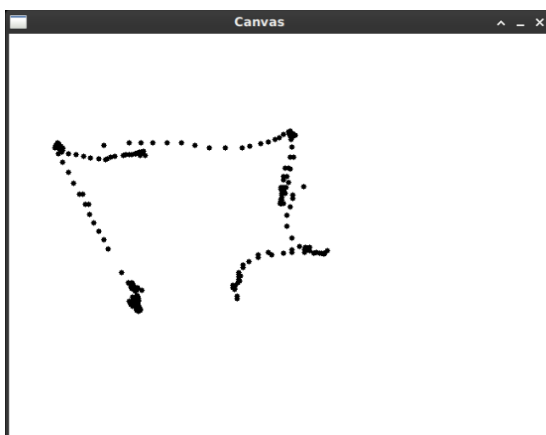
rsec_project / archived /

+ ▾

HistoryFind fileEdit ▾Code ▾

Name	Last commit	Last update
..		
ball_detector.py	cleaned up folder structure	1 day ago
example_1_get_image.py	cleaned up folder structure	1 day ago
example_ball_detector.py	changed folder structure	23 hours ago
main.py	cleaned up folder structure	1 day ago
paint_circles.py	cleaned up folder structure	1 day ago
paint_circles_v2.py	cleaned up folder structure	1 day ago
skeleton.py	cleaned up folder structure	1 day ago
track_rectangles_noMovement.py	cleaned up folder structure	1 day ago
track_rectangles_withMovement.py	cleaned up folder structure	1 day ago
writing_detection.py	changed folder structure	23 hours ago
writing_detection_with_screenshot.py	clean up	1 day ago

The following pictures represent the final stage of the project.



Discussion

There were problems we encountered while iterating on our drawing program. Some problems were fixed or circumvented, while others remained, therefore the scope of the project had to be reduced.

Optimization

It is unknown whether there's technical limitations due to the VM or if the performance can be improved through further optimization. We opted to use OpenCVs shape and color detection instead of YOLO for performance reasons.

Text Detection

Before the training on the handwritten text samples was finished, we ran out of GPU-time in Google colab and the progress was discarded. Thus, due to time constraints the implementation of the whole text detection was not further elaborated upon and scratched completely.

Canvas Size

The most difficult part was expanding the canvas beyond the border of the camera. To accomplish this, many more parameters needed to be considered. The reason for this is the "disproportional" size difference between the actual video (view) and the larger canvas. Furthermore, if Gretchen would recognize the drawing utensil to be inside the outer ~10% border for of the video, it would follow it, but still draw all the points during the process. This meant that some "unwanted" points were drawn when panning the camera. For this we had to take more complex calculations into account and some help from the tutor(s) (thank you Walley!).

Conclusion

The goal was mostly reached. Gretchen's camera input can be used to track a pen and draw. The drawing board extends beyond the camera frame with the use of a canvas. Gretchen's actuators are used to move the camera frame in order to use different areas of the canvas. Further improvements can be made to the object tracking and drawing. A higher frame rate would enable the user to draw more freely and consistently. Finally, object detection could be used to recognize more "general" utensils, instead of being "bound" to a specific green pen.

Inspirational Links:

for gestures:

<https://pub.aimind.so/real-time-hand-gesture-recognition-using-opencv-a-step-by-step-guide-2820618caa08>

<https://www.opencvhelp.org/tutorials/applications/gesture-recognition/>

for handwriting:

https://keras.io/examples/vision/handwriting_recognition/

(also a link to the data set we tried to use (around 3 GB)

<https://pyimagesearch.com/2020/08/24/ocr-handwriting-recognition-with-opencv-keras-and-tensorflow/>

main idea:

<https://qrka.medium.com/gesture-based-visually-writing-system-using-opencv-and-python-ramkrishna-acharya-300e0f20d9cd>