BLG453E
Homework-3

Asst. Prof. Yusuf Huseyin Sahin
sahinyu@itu.edu.tr

The doll was so utterly devoid of imagination that what we imagined for it was inex- haustible. For hours, for weeks on end, we must have been content to lay the first fine silk of our hearts in folds around this immobile mannequin, but I have to believe there were certain abysmally long afternoons when our twofold inspirations petered out and we suddenly sat in front of it, expecting some response. (...) It remained silent then, not because it felt superior, but silent because this was its established form of evasion and because it was made of useless and absolutely unresponsive material. It was silent, and the idea did not even occur to it that this silence must confer considerable importance on it in a world where destiny and indeed God himself have become famous mainly by not speaking to us. (...) That we did not then make you into an idol, you brat, and perish from fear of you, was because —I must tell you— it was not you we had in mind. We were thinking of something quite different, invisible, something we held at arm's length from you and from ourselves, furtively, with vague anticipation, something for which both of us were in a way only pretexts. We were thinking of a soul, the soul of the doll.

Some Reflections on Dolls, Rainer Maria Rilke

Ja, ich reiß' der Puppe den Kopf ab
Und dann beiß' ich der Puppe den
Hals ab Es geht mir nicht gut, nein!

Till Lindemann, Puppe

Part 1: Hand tracking

Both Rilke and Lindemann were angry with the dolls, and you will be angry with them too. However, instead of a toy doll, in this homework we will work on a ragdoll which is used for character animation. With the homework document, I uploaded a video file: biped_1.avi. containing a walking ragdoll animation with a black background.

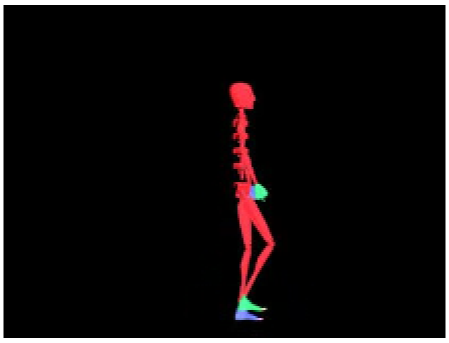


Figure 1: Example frame from the ragdoll video

To read the videos frame by frame you can benefit from the *moviepy* library. An example for reading frames is given below.

```
import moviepy.video.io.VideoFileClip as mpy
import cv2

biped_vid = mpy.VideoFileClip("biped_1.avi")

# Number of frames
frame_count = biped_vid.reader.nframes

# Frames per second
video_fps = biped_vid.fps

# Iterate over frames
for i in range(frame_count):
    walker_frame = biped_vid.get_frame(i * 1.0 / video_fps)
    # walker_frame is a NumPy array (RGB)
```

First, define coordinate points for right hand of the ragdoll (green one). Then, for consecutive frames, implement Lucas-Kanade algorithm locally to find the OF vector of the hand. Using cv2.arrowedLine function, place **one** arrow on the hand according to the OF vector[1]. You should not misplace the arrows in any frame. Along the walking sequence they should remain on the hand.

[1] Hint: You can use another frame from the sequence to make a better prediction if you want (e.g. to predict the motion vectors of Frame n, instead of only using Frame n+1, you can also use Frame n+2. It is not necessary but it may increase your findings.)

Part 2: Dice Game

- You are allowed to use image reading/writing operations, warpPerspective and Harris corner detector from OpenCV at this point. However, if you want to use another function, please ask for it using the Message Board.

- You can download the material from https://web.itu.edu.tr/sahinyu/TermProjectFiles.zip

Input: *A simple computer game prepared in Unity3D*

Expected Output: *Good scores*

In this part of the project, you will write a Python code to play a dice game. Your script will use pyautogui to take screenshots from the game and simulate pressing the keyboard buttons. In the game, three dice are shown to the player and the player should select the die with the greatest number. There are two game types depending on whether the dice are randomly rotating. The Game 1 and Game 2 screens are as given in Figure 1 and Figure 2. You will select one of them, switch to your IDE/terminal and start your program, switch back to game to watch your program playing the game. You should simulate A, S and D buttons to select a die.
Here, the following strategies should be followed:

- **Game 1:** Every die should be detected using edge detection. Then, you should implement Hough Circle Detection algorithm to count the dots on these dice. In addition to Hough Circle detection, you can also make a connected components analysis to be sure about your results.

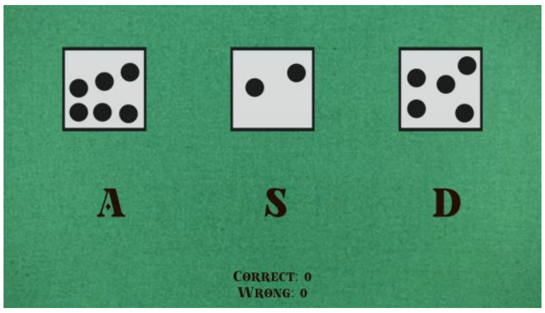


Figure 2 Dice Game: Game 1

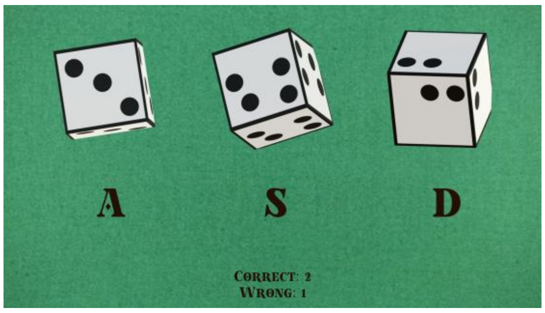


Figure 1 Dice Game: Game 2

- **Game 2:** Since the dice are rotating, you should calculate a perspective transform for the square on top surface to flatten the surface. Then you should use your Hough Circle Detection algorithm on this flattened image. As in the previous part, you can also use connected component analysis. (Hint: It is a good idea to use more than one screenshots. You can also define an error rate for Hough Circle Detection.)

Part 3: Mine Game

Input: *A simple computer game prepared in Unity3D,* ***Expected Output:*** *The character should reach the last grids.*

I prepared another game in which the main character is trying to reach the end of the map given in Figure 3. By default the character can be controlled using arrow keys and shift key makes him run faster. You should also use *pyautogui* to take screenshots of the board, decide on the next action and simulate these actions.

The problem with these grids is, some of them are mined. However, if our character is close to a mined grid (according to grid borders), a face image having a "shocked" expression appears on the bottom right as given in Figure 4.

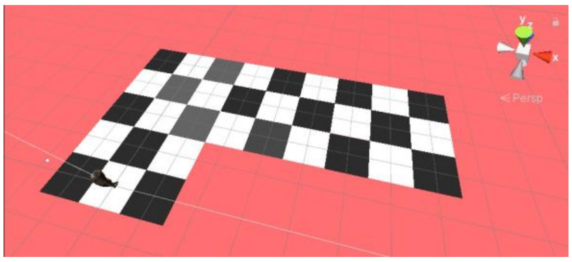


Figure 3 Mine Game: The Map

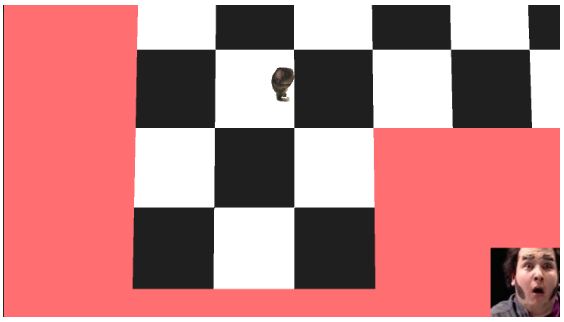


Figure 4 Mine Game: The character is very close to a mined grid.

Here, the following strategies should be followed:

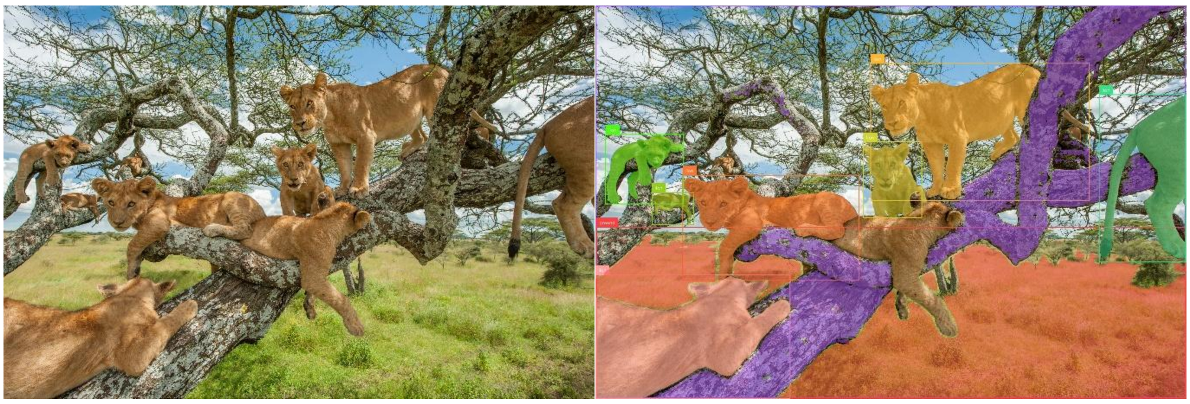
- Using facial landmark detection, detect the difference between "shocked" face and "normal" face. Try to not enter mined grids.
- You can keep a list of visited grids so that, the character should not tend to visit some grids over and over. Thus, you should benefit from corner and edge detection to detect whether the character is entered to another grid.
  - Make the character reach the last grids safely.

Part 4: Comparative Image Segmentation with Grounded-SAM

“O-o-o-o-oh
See the lions in the cage
O-o-o-o-oh
See the victims of the rage
Lions, lions in a cage
Fifty years behind a wall
Lions, lions in a cage
See those lions in the cage”

-Pentagram

Grounded-SAM is an image segmentation framework that combines two different neural networks to perform **text-guided object segmentation**. The first network, **Grounding DINO**, is responsible for understanding the **text prompt** given by the user and locating the corresponding objects in the image. It does this by predicting **bounding boxes** around the regions that match the described objects.
The second network, **Segment Anything Model (SAM)**, focuses on **pixel-level segmentation**. Instead of working directly with text, SAM takes the bounding boxes produced by Grounding DINO as input and refines them into accurate segmentation masks. In this way, Grounded-SAM separates the tasks of *object localization* and *segmentation* into two stages.
This two-stage pipeline allows Grounded-SAM to segment a wide variety of objects described in natural language, without requiring predefined object classes or additional training.



In this part of the homework, given the input image *lionsintree.jpg*, you should do the following steps:

A) Segmentation of the original image

- Apply Grounded-SAM directly on the original image.
- Use a text prompt of your choice. For example, in the code shared with you, prompt is “lion. tree. ground.” But it can be more detailed or more abstract.

- Save the resulting segmentation masks and visualized outputs.

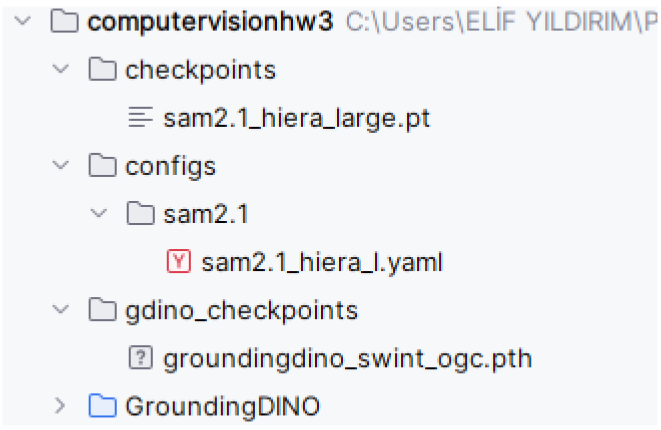
B) Segmentation of a geometrically distorted image

- Modify the input image by changing its **aspect ratio** (e.g., stretching or compressing the image along one axis).
- Apply Grounded-SAM on this **non-proportionally resized image**.
- Observe and report how geometric distortion affects the segmentation results.

C) Segmentation using edge-based image representation

- First, extract the **edge map** of the original image using **Canny Edge Detection**.
- Use the resulting edge map as the input to Grounded-SAM.
- Perform segmentation and save the outputs.

(Hint: Please download the model weights for sam and groundingdino from huggingface and by using git, you should import GroundingDINO repo)



```
computervisionhw3 C:\Users\ELİF YILDIRIM\P
    checkpoints
        sam2.1_hiera_large.pt
    configs
        sam2.1
            sam2.1_hiera_l.yaml
    gdino_checkpoints
        groundingdino_swint_ogc.pth
    GroundingDINO
```

For further information, you can check the resources below:

- Ren, T., Liu, S., Zeng, A., Lin, J., Li, K., Cao, H., ... & Zhang, L. (2024). Grounded sam: Assembling open-world models for diverse visual tasks. arXiv preprint arXiv:2401.14159.
- Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., ... & Zhang, L. (2024, September). Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In European Conference on Computer Vision (pp. 38-55). Cham: Springer Nature Switzerland.
- Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., ... & Girshick, R. (2023). Segment anything. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 4015-4026).