Task 1:

Usage:

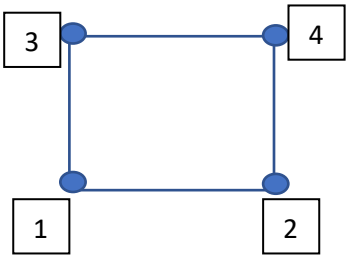
python task1.py --Grid Grid(str) --Orientations Orientations(str) --FoV FoV(str)

Sample Input (These values are also stored as default inputs):

python task1.py --Grid `'[[0,0,0,0,0],[T,0,0,0,2],[0,0,0,0,0],[0,0,1,0,0],[0,0,0,0,0]]'` --Orientations `'[180,150]'` --FoV `'[60,60]'`

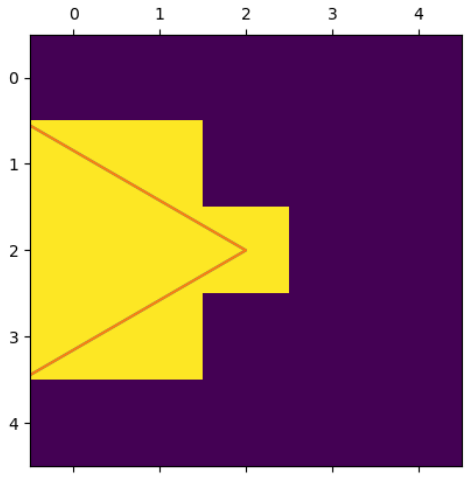
The idea used to approach task 1 in writing a function for thief_and_cops(grid,orientations,fov) is to break down the grid into assigning a "-1" number to the thief and finding the number of cops in the grid. For each cop in the grid, I construct a field of view cone based on the cops position and orientation.

To make sure that for each grid cell, the line cuts through is included in the field of view, I am considering four different possible notations to represent the grid. In my representation of the grid, the x and y axis start at -0.5 and end at len(grid)-0.5. (Shifted by -0.5). The reason for the shift is to align the indices of the grid cells as a coordinate. From the image below, the four possible notations to represent a cell are by fixing one of the four points for all the cells in a grid. I then check if the point lies towards an interior point or away from an interior point to construct the cones. The interior point is a small displacement from the center towards the line direction. The point 1,2,3,4 on the figure represent the notations used for four different grids in the code. If one of the four points is interior with respect to the cone, the entire cell lies within the cone.

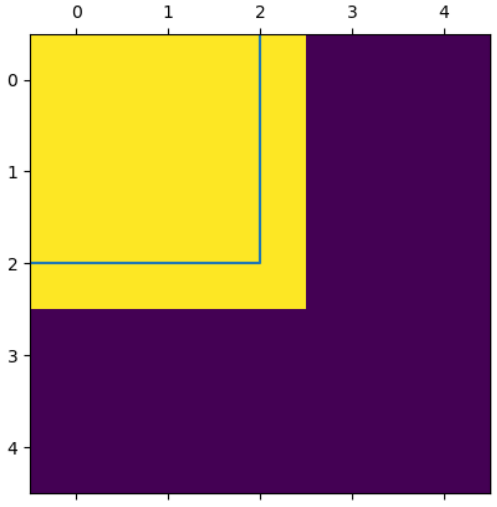


In the next two pages, I also include some tests I have done for these cones, which are the crucial part of the problem. Given more time to spend on the project, I would test these cones more rigorously and at different positions and orientations. The current tests included are limited but span across field of view dimensions. Once the cones are computed, I check if the thief lies in any cop's field of view. If it doesn't, the thief does not move, else, I compute a joint mask for all the caps and take a negative mask to find the minimum displacement of the thief.

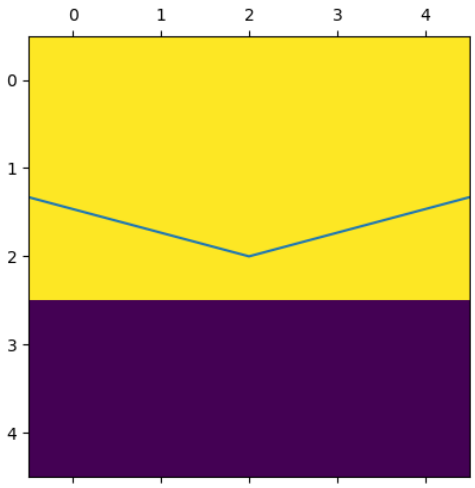
```
## For testing the cones (Calculated points manually using line equations)
# fig, ax = plt.subplots()
# ax.matshow(mask1,)


grid = np.array(
[[0,0,0,0,0],[-1,0,0,0,2],[0,0,1,0,0],[0,0,0,0,0],[0,0,0,0,0]])
orientation = [180,150]
fov = [60,60]
```



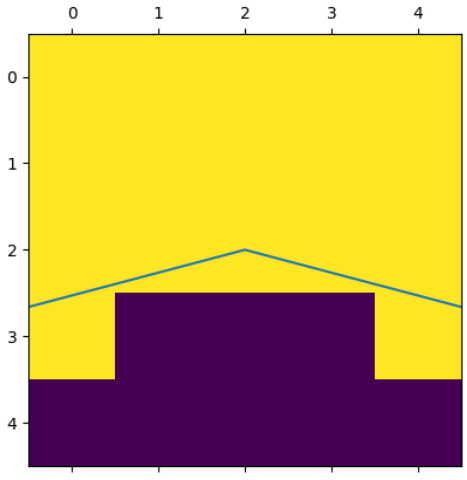

```
grid = np.array(
[[0,0,0,0,0],[-1,0,0,0,2],[0,0,1,0,0],[0,0,0,0,0],[0,0,0,0,0]])
orientation = [135,150]
fov = [90,60]
```



```
grid = np.array(
[[0,0,0,0,0],[-1,0,0,0,2],[0,0,1,0,0],[0,0,0,0,0],[0,0,0,0,0]])
orientation = [90,150]
fov = [150,60]
```




```
grid = np.array(
[[0,0,0,0,0],[-1,0,0,0,2],[0,0,1,0,0],[0,0,0,0,0],[0,0,0,0,0]])
orientation = [90,150]
fov = [210,60]
```



For a quick check, I calculated the lines by hand, but if I were to spend more time on the project, I would write a code to also compute the triangles shown in the plot.

Task 2:
There are three version included in the task 2 folder.
-**track_yolo_kcf.py**, is a faster approach but inaccurate compared to the other two.
-**track_yolo_csrt.py,** is the slowest of the three, and is an accurate tracking model until occlusions occur.
Both first and second method use a pretrained yolo network and an object tracker, and I use the pretrained network to reinitialize the object tracker.
-**yolov7/detect.py,** A refined version of the pretrained yolo network predictions. The speed of this version is between the first two methods.
For a numerical comparison apart from model loading(~3s) in the setup phase, the three methods take 13.78s, 23.24s and 18.86s respectively. For 958 frames in the video, the fps of the three methods are 69.5, 41.22,50.79.
Also included in the task_2 folder are:

- video_frames_csrt.csv
- video_frames_kcf.csv
- video_frames_yolo.csv
- yolov7 -- > directory of the yolov7 repo. Weights(yolov7.pt) in the google drive folder or from the official repo need to be added to this folder to run all the three versions
- task2_extract.py – script to generate images from the video, which we use for the tracker+yolo methods
- yolo_makecsv.py – script to generate csv from the labels generated from the yolo predictions.

In the google drive folder, there are videos for the three methods.There are two videos for yolo, before and after removing the bad frames(after frame 631). The images folder for the output of task2_extract.py and the weights for the yolov7 network in yolo_weights/yolov7.pt.

Approach Tried:

I tried experimenting determining an HSV Mask, and then fitting a circle to track the ball, but the colors of the stadium, the seats and the Coca-Cola banner, along with the white shoes, make it a difficult problem for generating a good range of values of HSV for the mask of a soccer ball in the initial template.
The next approach I was trying to use was Template Match to detect the object and then extract the bounding box information, but the quality of the video, the

fast movements of the ball within the frames, and the change in orientation of the ball caused some difficulties.
After that, I moved to Object tracking approaches in the OpenCV2 Library, the KCF tracking was extremely quick on 958 frames at around 8 seconds, but was very inaccurate. The next best tracker was CSRT, which is very accurate until the ball goes out of the frame at frame 582. Once the ball goes out of the frames, CSRT trackers lose track of the ball. To resolve this situation, I was initially looking at Template Matching with the Initial Frame to reinitialize the tracker and keep the tracker loop going. But, as the video approaches the end, the ball moves quicker and is most often under occlusions. The final approach tried was to use to a yolov7 pretrained network to reinitialize the tracker. When the tracker fails to update, I use a yolov7 network to find a new bounding box and reinitialize the tracker to track the ball. I also modified the pre-trained network detect.py code to just detect and plot the bounding boxes of a soccer ball. I have also refined the results of the yolo pretrained network by removing the predictions of a smaller ball by computing distance to a running mean of bounding boxes from previous predictions in yolo with a distance threshold. Although querying the yolo model, slows down the kcf tracker, it is accurate than just using the kcf tracker.

For the csrt tracker, the results are good until frame 582, but after even with yolov7 reinitialization, the tracking was inaccurate and slow. So, I decided to use a kcf tracker with yolo reinitialization after occlusions occur, to speed up the tracking and have slightly better accuracy than just using a CSRT Tracker with yolo reinitialization.

I also compare the results with the yolovy pretrained network predictions, where I use the confidence threshold of 0.5 to detect the ball. Even with yolov7, there are multiple predictions in a frame, and sometimes other objects get classified as balls. But, the speed vs accuracy of yolo is very competitive with that of the other trackers. For cases, where there is a ball trail/ multiple detections of the ball in a frame, I compute the distance to the running mean and choose the closest to the running mean.

If I were to spend more time on the project:

- Write an object tracker implementation from recent works, rather than just relying on the opencv2 trackers.

- Try more methods and spend more time on the HSV Masking as well for better object tracking
- Work on more heuristics, to reduce the false detections with yolo.
- Fine tune the yolov7 model by generating more data with soccer ball. Current model uses the COCO dataset for predictions and there is a sports ball in the COCO dataset.
- Try out various pretrained networks , compare and choose the best one for the application (ROLO, DeepSort,MDNet)
- Towards the end of the video, even yolov7 cannot detect the frames of the ball, when the player attempts a goal. There is a need for improvement here. Integrating physics based models with better segmentation models can also be interesting to look at in this direction.

To Run the files:

1. **Include the images folder from the google drive folder shared or run the task2_extract.py file after creating images folder and copying ball_tracking_video.mp4 to task2 folder.**
2. **Copy the weights file from weights in google drive folder to the yolov7 directory.**
3. **To run track_yolo_kcf.py, run the file from the task2 folder, default parameters can be changed from command line.**
4. **To run track_yolo_csrt.py, run the file from the task2 folder, default parameters can be changed from command line.**
5. **To run yolov7/detect.py, copy ball_tracking_video.mp4 to yolov7 directory and run detect.py from yolov7 directory. If –save-txt variable is used while running, labels are generated and stored in runs/exps_num/labels/**
6. **To run yolo_makecsv.py, I also have labels for one of the experiment runs, exp8 on the repo to run the files. But change the path to generate a different csv.**
7. **If using for a different video, bad_frames need to be changed. I refined the detect.py , specifically to soccer_ball.py (which needs to be changed for other applications.)**