Task Documentation

Task 1 : Thief and Cops

This task involves a pursuit scenario where a thief is hiding inside city blocks while cops are stationed at various locations with a given field of view (FoV). The goal is to determine:

1. Which cops can see the thief.
2. The closest safe block where the thief can move without being seen.

Solution Overview

The solution consists of two primary sub-tasks:

1. **Checking if a Cop can see a Block**
2. **Finding the Nearest Safe Block for the Thief**

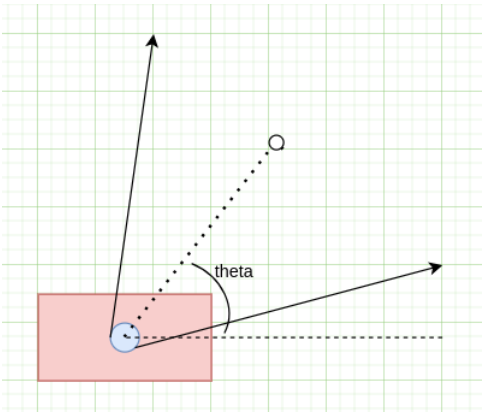
Sub-Task 1 :: Check if a Cop is looking at a Block

Each cop is positioned in a specific grid cell with a defined orientation and FoV. The key idea is to determine whether any part of a given block falls within a cop's view.

Check Point-in-View

To check visibility for a point, we compute the orientation angle of the block from the cop's position:

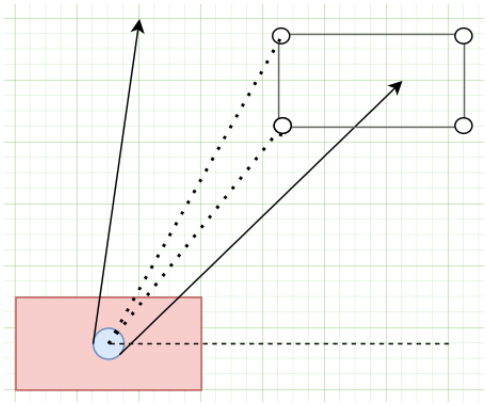
1. Find the angle between the cop's position and a point in the block.
2. Check if this angle lies within the cop's FoV.



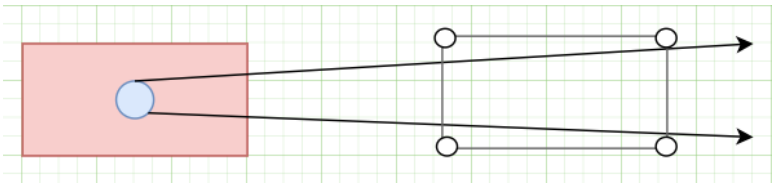
Check Block-in-View

For the block, we check if any of its four corners are visible:

- Compute the angles from the cop to each of the four corners.
- If any of these angles fall within the cop's FoV, the block is considered visible.



Edge Case Handling



In cases where the FoV is small, none of the block's corners may be directly in view, but part of the block might still be seen. To handle this:

- Check if any edge of the block intersects with the cop's view boundary.
- View boundaries are defined by two lines originating from the cop's position at the extreme angles of their FoV.
- If an edge of the block crosses one of these lines, the block is considered visible.

Sub-Task 2 :: Finding the Nearest Safe Block

A **safe block** is a block where the thief is not visible to any cops.

Step 1: Create a Visibility Grid

- Compute which blocks in the grid are visible to each cop.

- Store this information in a **visibility matrix** where each entry contains the list of cops seeing that block.

Step 2: Breadth-First Search (BFS) for the Closest Safe Block

- Since BFS explores all directions evenly, the first safe block found is guaranteed to be the closest one in terms of Manhattan distance.

Results

```
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]
fovs = [60, 60]

cops_viewing_thief, safe_block = thief_and_cops(grid, orientations, fovs)

print(cops_viewing_thief, safe_block)
```

Output — [2] [2, 2]

Task 2 : Object Tracking

The objective of this task is to track a soccer ball being juggled by a player in a video. The algorithm should:

1. Detect and track the ball in each frame.
2. Store the ball's center coordinates and bounding box size in a CSV file.
3. Generate an output video with the bounding box drawn around the ball.

Solution Approach

The approach involves two main components:

1. **Tracking the Ball across Frames**
    - A **CSRT Tracker** is used for robust tracking.
    - If tracking fails, the ball is **re-detected using a YOLO object detector**.
2. **Handling Occlusions and Lost Tracking**
    - When tracking confidence drops, **YOLO is used to re-detect the ball**.
    - **Structural Similarity Index (SSIM) and Abs Difference Score** are used to check if the tracked object still resembles the ball.
        - SSIM —
          SSIM is a perceptual metric that quantifies image similarity by comparing luminance, contrast, and structure. It is used here to determine whether the tracked ball still resembles the reference ball, with values closer to 1 indicating higher similarity.
        - Difference Score —
          This measures the absolute pixel-wise difference between the tracked ball and the reference ball, thresholded to highlight significant changes. A high difference score suggests that the tracked object has deviated significantly from the expected appearance, indicating potential tracking failure

Results

Video —

AIM Task 2 - Ball Tracking (Parth Shah)

https://youtu.be/KQzd6QUj2Ww

CSV File —

https://drive.google.com/file/d/1ioscvw2UQ18su_RZD_OrEoz8sA_cj-pj/view?usp=sharing

Average Time for each frame— 0.03s

FPS — 30 FPS