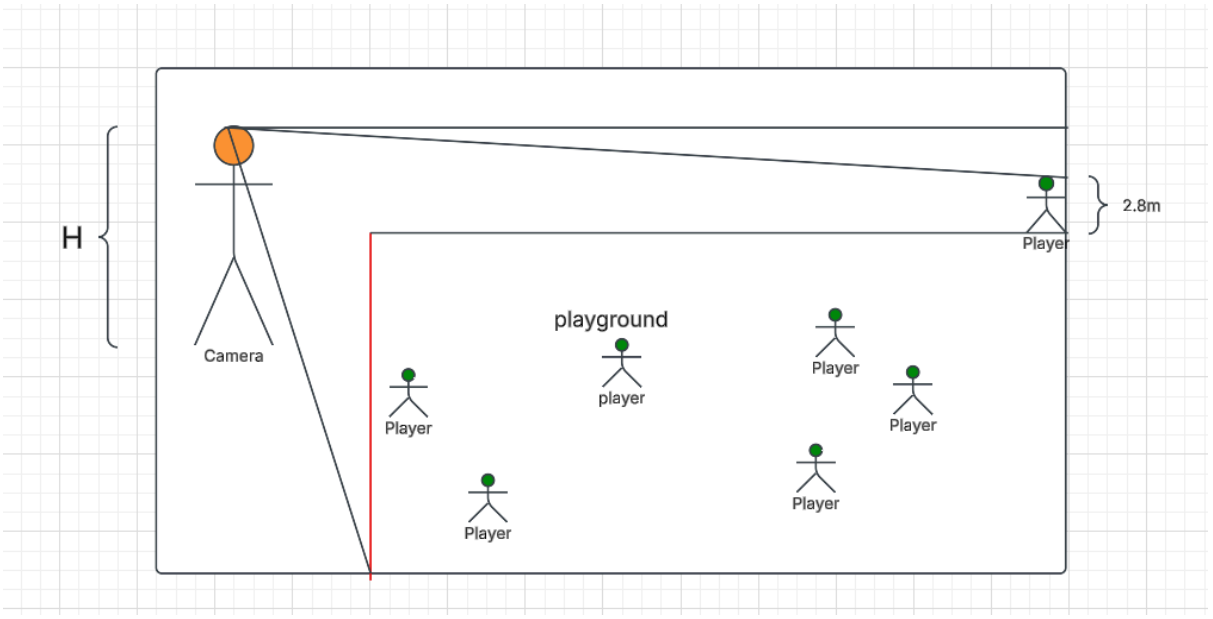
Problem statement 1:
Squid Game Challenge - RED Light Green Light System

Objective:
Design and implement a "Red Light Green Light" detection system inspired by Squid Games. Participants wearing green (or any contrasting color that stands out against the background) must remain still during the "Red Light" phase. Any detected motion during this phase should result in identifying the participant's location on the ground. Ensure guards are not mistakenly targeted.

Solution :

1. **Camera Position**



Things to keep in mind:
1) The camera should be fixed.
2) It needs to be established at a height H, such that its FOV covers every player.

2.Player detection

Input : A frame from camera feed

Output: A list of contours representing the detected players

Things to note :

1) We will work on an HSV frame so we need to convert input frames to HSV frame.

HSV corresponds to:
Hue is color
Saturation is the grayness
Value is the brightness

Below is the sample code of to change frames into HSV frame, we will put this code in an while loop so that it applies to each frame in a video :

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  while True:
7.      _, frame = cap.read()
8.      hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

And since, we will be targeting only the persons wearing green , we need to detect the green color:

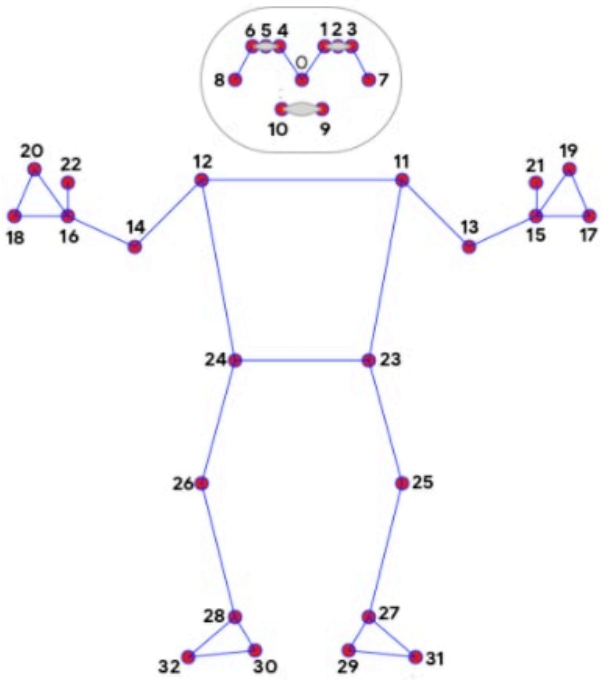
```
7.      # Green color
8.      low_green = np.array([25, 52, 72])
9.      high_green = np.array([102, 255, 255])
10.     green_mask = cv2.inRange(hsv_frame, low_green, high_green)
11.     green = cv2.bitwise_and(frame, frame, mask=green_mask)
12.
```

3.Motion detection :

We will use an ESP32-CAM with OpenCV in Python to detect motion and return the
coordinates of moving players. The ESP32 will capture frames, perform basic motion detection,

and send coordinates to a Python server via WiFi.
1) The ESP32 will continuously capture frames using its camera module.
2) It will send frames to a Python server via HTTP/UDP/WebSocket for processing.

Now once we get the video we have to locate its landmarks to detect body movements. For this, we are going to use mediapipe library using this we are able to find landmarks of a person standing in front of the camera as in the image below. For now we will use the model with 33 landmarks to understand the process. Later on we can use more advanced models. (for more details, refer to “Room for improvements” section)



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

(Source : Google)

Now, change in distance between any two points beyond a threshold should be considered as movement
● Compute the **Euclidean distance** between all pairs of landmarks at time t and (t−1).
● If the distance between **any pair of landmarks** changes beyond a threshold, motion is detected.

D = $\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$, where (x1,y1) and (x2,y2) are landmark

coordinates in two locations on body. We will calculate this distance at time t and at time t-1. The change in its value signifies motion.
Define a threshold T above which is a significant motion.
If $|dt-dt-1|>T$ for any pair, **flag movement**.
The output will be the coordinate of motion. Next we need to change those coordinate to real world position using **Homography matrix** of opencv.
homography is a 3×3 matrix we can write it as :

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

Now Calculation of H :

1) Place four or more reference points on the ground and note their real-world (X,Y) coordinates.
2) Get the coordinate of movement (u,v) from ESP32 cam
3) Use OpenCv.findHomograpy(original_coordinate, real_world_coordinate)
   Below is the pseudocode to calculate H:

```
START
Place at least four known reference markers on the ground
Manually measure real-world coordinates (X, Y) for each marker
Capture corresponding pixel coordinates (u, v) from ESP32-CAM feed
Store both sets of coordinates

Compute Homography Matrix H:
    H, _ = cv2.findHomography(pixel_points, real_world_points)

Store H for future transformations
END
```

PSEUDO CODE FOR ESP32 Cam :

```
START

Initialize ESP32-CAM
Connect to WiFi
Initialize Mediapipe Pose Model
Load Homography Matrix H  # Precomputed above
Set motion_threshold for movement detection
Set previous_landmarks to None

WHILE True:
    Capture frame from camera
    Convert frame to RGB format
    Convert frame to HSV format for color detection

    Apply green color mask:
        Threshold HSV image to detect green regions
        Create a binary mask where green is detected
        Perform morphological operations to reduce noise

    Process frame with Mediapipe to extract landmarks of the whole body

    Filter landmarks:
        Keep only landmarks whose pixel coordinates fall within green mask

    IF previous_landmarks exist:
        Motion_detected = False
        Moving_participant_pixel_coords = []

        FOR each pair of landmark points (i, j) belonging to green-detected persons:
            Compute Euclidean distance at time t:
                D(t) = sqrt(xi(t)-xj(t)^2) + (yi(t)-yj(t))^2)
            Compute Euclidean distance at time t-1:
                D(t-1) = sqrt(xi(t-1)-xj(t-1)^2) + (yi(t-1)-yj(t-1))^2)
            IF abs(D_t - D_t_minus_1) > motion_threshold:
                Motion_detected = True
                Store (xj(t), yj(t)) in Moving_participant_pixel_coords

    Store current landmarks as previous_landmarks

    IF Motion_detected:
        Moving_participant_real_coords = []
        FOR each (u, v) in Moving_participant_pixel_coords:
            Compute real-world coordinates using Homography:
                [Xr_h, Yr_h, Wr] = H * [u, v, 1]  # Matrix multiplication
                Xr = Xr_h / Wr  # Normalization
                Yr = Yr_h / Wr
                Zr = fixed value Zr  # Assume Zr based on player height

            Store (Xr, Yr, Zr) in Moving_participant_real_coords

        Send Moving_participant_real_coords to Python server via WebSocket/UDP

    Delay for stability
END
```

6. Gun Position

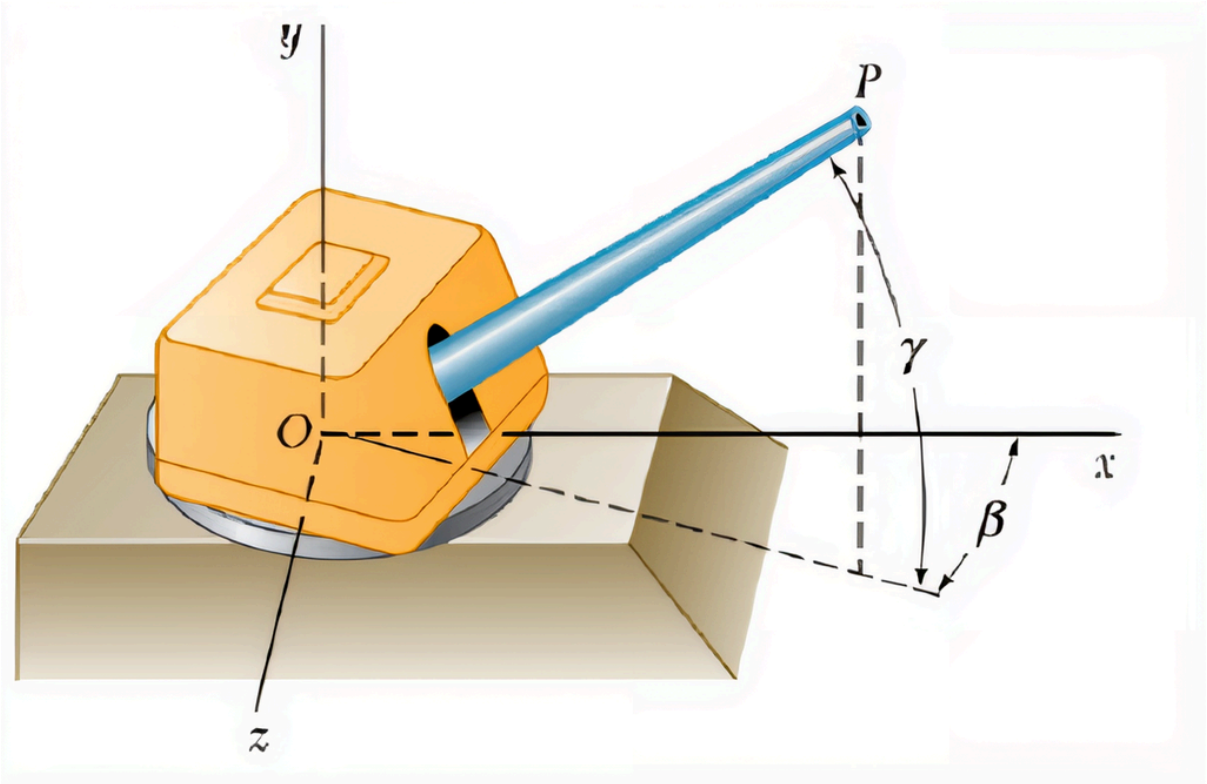
In order to shoot accurately at a moving player:
1) We need to mount gun at a height

2) The real world positions we got from the third task need to be sent to arduino which will keep a list of all the coordinates.
3) Those coordinates from arduino will be sent to servo one by one which will map those positions with respect to that of the gun and align the barrel according to that.
4) Only when one target has been shot then only the next coordinate is to be sent by arduino.
5) We need to know the angles at which gun should be kept so that it aims perfectly
6) We need to use servo motors for rotation of gun to cover the whole area
7) At last we need to shoot at the right moment so that it does not miss any player who moves.

Detailed Mechanism:

We already got the target's **final real-world position** as: (Xr,Yr,Zr)
Let's say **the gun's coordinate** is : (Xg, Yg, Zg)
So the target's position with respect to that of gun is :

r = $\sqrt{(Xg - Xr)^2 + (Yg - Yr)^2 + (Zg - Zr)^2}$



(Source : Google)

Next, In order to say the exact alignment of the gun, we need to say about β (Azimuthal angle)
and γ (Elevation angle)

For the perfect aim ,β = $tan^{-1}(\frac{Yg-Yr}{Xg-Xr})$ and γ = $tan^{-1}\frac{Zg-Zr}{\sqrt{(Xg-Xr)^2+(Yg-Yr)^2}}$

Additionally, we need a **pan-tilt platform** in order to move the barrel in horizontal and vertical
axes. Means we need two servo motors , one moving the gun horizontally (say, pan servo) and
one moving the gun vertically (say, tilt servo).

Below is the pseudo code needed for this step:

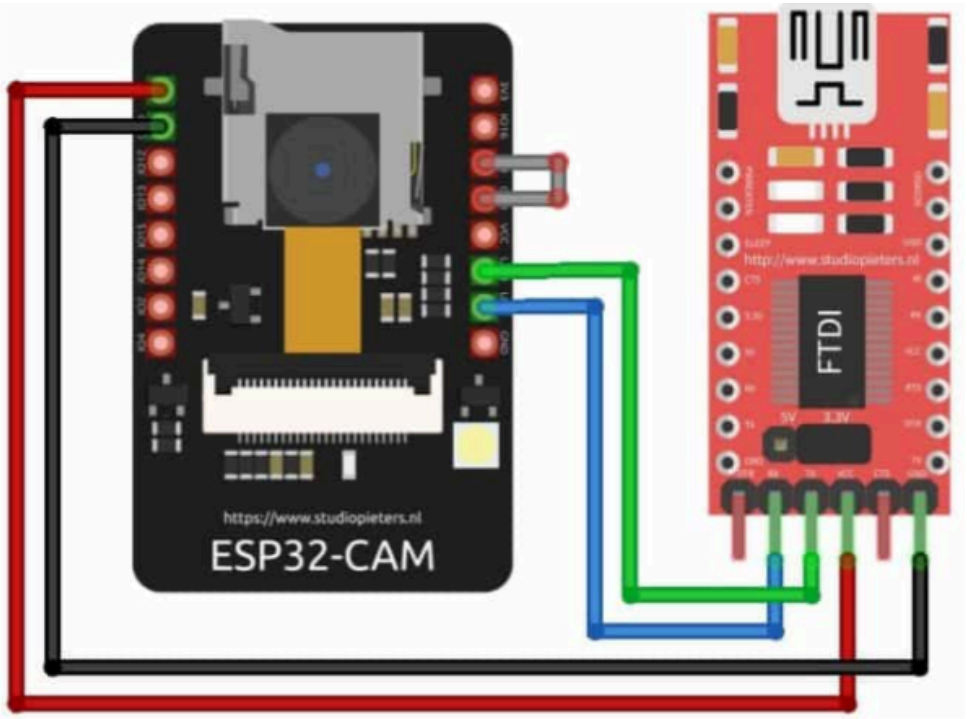
```
## Arduino Pseudocode Logic
START
Initialize Serial Communication
Initialize Servo Motors for Yaw (Horizontal) and Pitch (Vertical) control
Mount gun at predefined height (Zg)

WHILE True:
    IF new target coordinates received from camera system:
        Store (Xr, Yr, Zr) in queue

    IF queue is not empty AND gun is idle:
        Retrieve next (X_target, Y_target, Z_target)
        Compute required horizontal and Vertical angles
        Send computed angles to servo controller
        Wait until servos confirm alignment
        Trigger shooting mechanism
        Remove target from queue after successful shot
END
```

```
## Servo Motor Control logic
START
Initialize Servo Motors for:
    - horizontal rotation
    - vertical rotation

WHILE True:
    Receive (X_target, Y_target, Z_target) from Arduino
    Compute gun angles:
        ΔX = X_target - Xg
        ΔY = Y_target - Yg
        ΔZ = Z_target - Zg
        azimuthal_Angle (β) = tan inverse of (ΔY / ΔX)
        Elevation_Angle (γ) = tan inverse of (ΔZ / sqrt(ΔX² + ΔY²))
    Move pan servo to β
    Move tilt servo to γ
    Wait for stabilization

    IF target is still in position:
        shoot()
    ELSE:
        Recalculate angles OR move to next target
END
```

Hardware implementation :

The overall flow of the project is : [ESP32-CAM] → [WiFi] → [Python Server] → [Real-World Mapping] → [Arduino] → [Servo Motors] → [Gun Fire]

First we need to upload code on ESP32. For that we will make connection between FTDI module and ESP32 cam module.



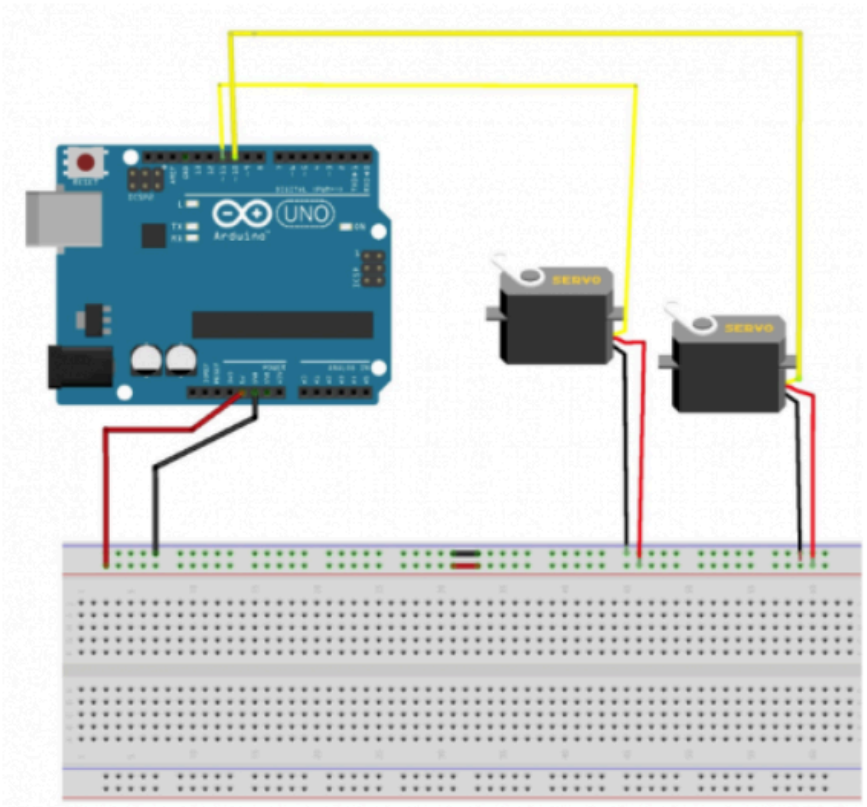
(Source : Refer to reference)

The following connection will be between FTDI and ESP32 :

ESP32-CAM	FTDI Programmer
GND	GND
5V	VCC
U0R	TX
U0T	RX
GPIO0	GND

(source : refer reference)

Next we need to connect arduino uno with two servo motors (pan and tilt servo motors)



(source : refer reference)

Up/Down Servo	Arduino	Right/Left Servo	Arduino
Red Cable	5V	Red Cable	5V
Black Cable	GND	Black Cable	GND
Yellow or White Cable	PWM(4)	Yellow or White Cable	PWM(10)

(source: refer reference)

7. Rooms for improvement :

Problem: The current mediapipe pose we are using only has 33 landmarks . Though this pose works well for all the body parts but this fails slightly in case of face movements like
talking. But According to the Squid game of the series , talking is also considered as movement so we need a more advanced mediapipe pose which has more landmarks.
Solution: We can use the mediapipe Face Mesh Model which has 468 landmarks. So even talking will be detected properly.

(Source : Google)

When we were solving this PS, we faced some difficulty in merging these two mediapipe models. That is why in the solution we just used the 33 landmarks. During the hackathon, we will try to merge the two mediapipe models to improve accuracy. If I elaborate the problem that we are facing is that let’s say a person’s hand is moving as well as he/she is talking then two coordinates will come as output from a single person and according to the whole algorithms the gun will get triggered twice for a single person which is unnecessary.

References:

https://learnopencv.com/homography-examples-using-opencv-python-c/
https://youtu.be/V9bzew8A1tc?si=Wj3qN99k0X1K5-BQ
https://pysource.com/2019/02/15/detecting-colors-hsv-color-space-opencv-with-python/
https://learnopencv.com/getting-started-with-opencv/
https://maker.pro/arduino/tutorial/how-to-set-up-pan-tilt-camera-stand-with-arduino-uno-and-joystick-module
https://how2electronics.com/esp32-cam-based-object-detection-identification-with-opencv/