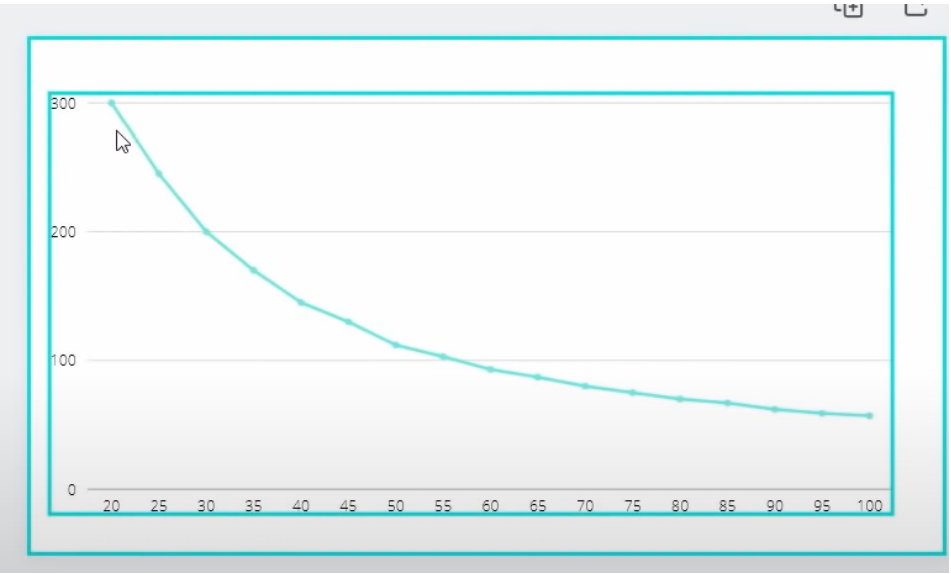
Graph between palm distance and hand distance




```
if counter:                                  #here the color chnage to green and
gain to purple
    counter +=1
    color = (0,255,0)   #green
    if counter == 3:
        cx = random.randint(50,600)
        cy = random.randint(50,300)
        color = (255,0,255)
        score +=1
        counter = 0
```

In the above code the target gets a random position and when we press it its color changes

Project Breakdown:

Vedio capture code

Detect hand

Use of lmList to detect hand points

Distance finding

Diagonal distance

Display: Score hud, time hud, target button

Counter: button pressable when distance is less than 50 cm

Random location to target button

Testing

Add piints / score in the score hud

Add time countdown in time hud

Game Over

Final score display

Reset button creation for the game

End

Concept of AI:

OpenCV contains a tool kit for the implementation of image based AI algorithms.

OpenCV was built for maximum efficiency and performance of computing-intensive vision tasks. Therefore, it has a strong focus on real-time applications of AI vision. The software is written in optimized C and is able to take advantage of multicore processors (multi-threading).

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library

MediaPipe Solutions provides a suite of libraries and tools for you to quickly apply artificial intelligence (AI) and machine learning (ML) techniques in your applications. You can plug these solutions into your applications immediately, customize them to your needs, and use them across multiple development platforms.

About the accuracy:

The code you provided appears to be a simple hand-tracking game where you have to press a button that appears on the screen when your hand gets close enough to it. The accuracy issue you mentioned is likely related to the way distance is calculated and how the button press is detected.

The code calculates the distance between two landmarks on the hand using the Euclidean distance formula:

pythonCopy code

distance = int(math.sqrt((y2-y1)**2 + (x2-x1)**2))

Then, it uses a polynomial fit to convert this distance to centimeters:

pythonCopy code

A, B, C = coff distanceCM = A * distance**2 + B * distance + C

The issue with accuracy can arise from a few factors:

1. Camera Calibration: The accuracy of distance measurement depends on the camera's intrinsic parameters (focal length, sensor size, etc.) and its position. If the camera is not calibrated or the parameters are not accurately set, the distance calculations may not be precise.
2. Hand Landmarks: The accuracy of hand landmark detection can vary based on the lighting conditions, background, and the hand's orientation. Small variations in landmark detection can lead to significant errors in distance calculation.
3. Polynomial Fit Coefficients: The coefficients (A, B, and C) for the polynomial fit may not be accurate for your specific camera and hand-tracking setup. You might need to fine-tune these coefficients based on empirical observations.
4. Button Interaction Logic: The code uses a simple rectangular region to check if the hand is within proximity of the button. The accuracy of this interaction logic can be influenced by factors such as the hand's orientation and the size of the button.

To improve accuracy:

- Calibrate your camera if possible to obtain accurate distance measurements.
- Experiment with different hand landmark detection models or parameters to improve the accuracy of landmark localization.
- Fine-tune the polynomial fit coefficients based on empirical data to better convert the distance to centimeters.
- Consider more sophisticated interaction logic that takes into account the hand's position and orientation relative to the button.

By addressing these factors, you can potentially improve the accuracy of the hand-tracking game and make it more robust to variations in hand positioning and camera conditions.