

# TENNIS BALL COLLECTING ROBOT

California State University Bakersfield

Senior Project II

Professor Mostafa Abdelrehim

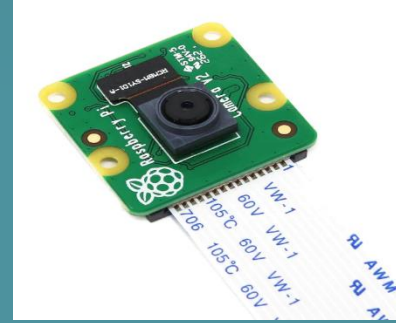
By:

Jose Martinez Campas

Christopher Kuhlhoff

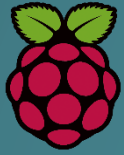
Julius Cardenas

# HARDWARE COMPONENTS



- Raspberry Pi 4
  - Used to provide the processing power for the objection detection and main controller of the robot. Also helpful because Python code can be written onto the Pi itself.
- Raspberry Pi Camera Module 2
  - Essential for OpenCV and Python code to capture images and stream data.
- Old RC Car Frame and Wheels
  - Used as the base for our robot, the RC already had 2 motors connected to the wheels.
- Lithium Ion Battery (2S)
  - Provides a maximum of 8.4 V down to 6 V of power for the robot to be fully wireless.

# SOFTWARE COMPONENTS



- Raspbian Buster OS
  - Older version of Raspbian OS, the newest version is Bullseye. Buster is currently more stable and has much more working open source software.



- Python Coding Language
  - Effective coding language that allows quick work and integration of systems more effectively.



OpenCV

- OpenCV
  - An open source computer vision and machine learning software library. Provides a common infrastructure for computer vision applications.



TensorFlow

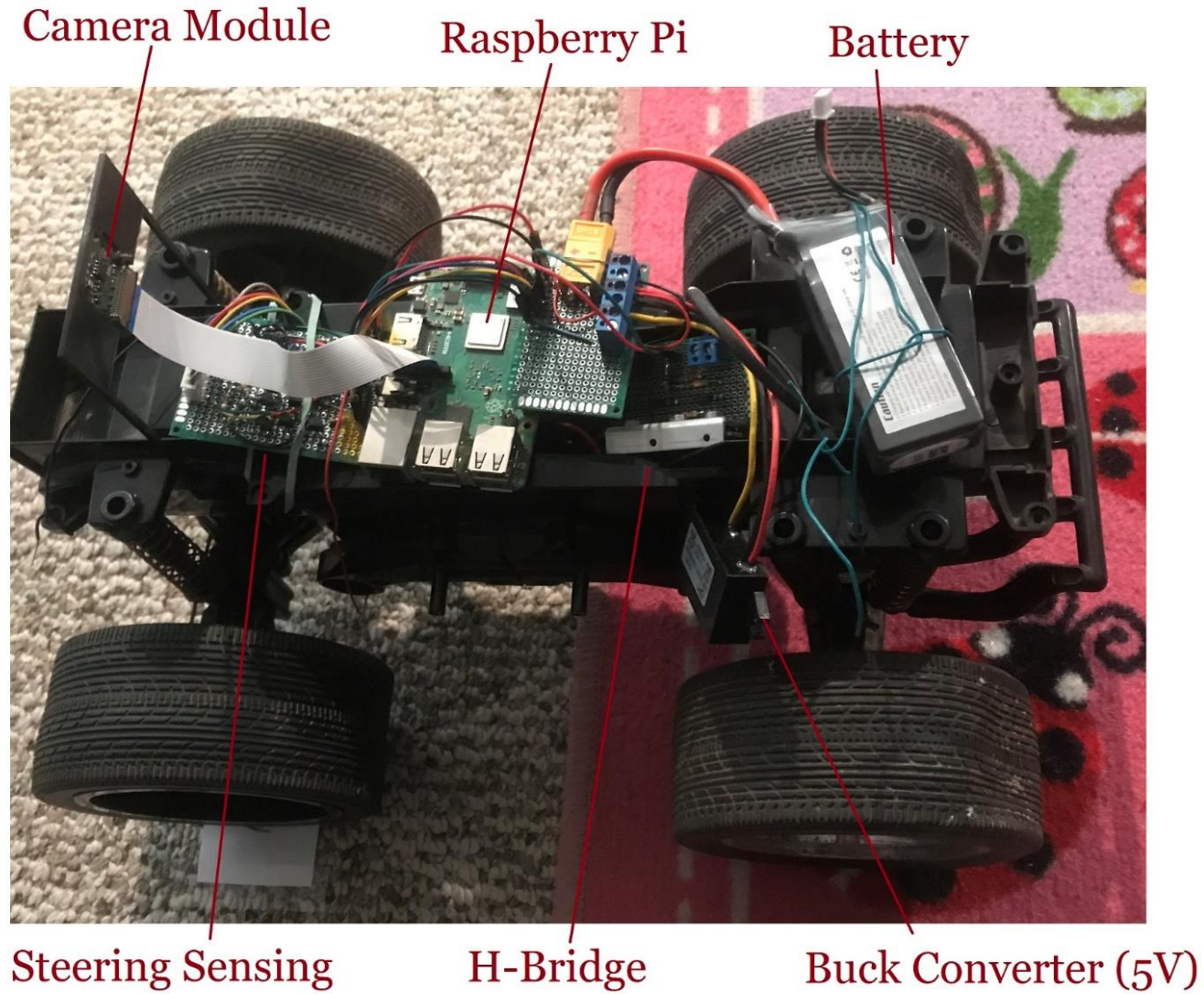
- TensorFlow Lite
  - Used to implement object detection and tracking in Python APIs



Flask

- Flask
  - Micro web framework written in Python. Used to setup a we server to display camera video stream.

# ROBOT DESIGN





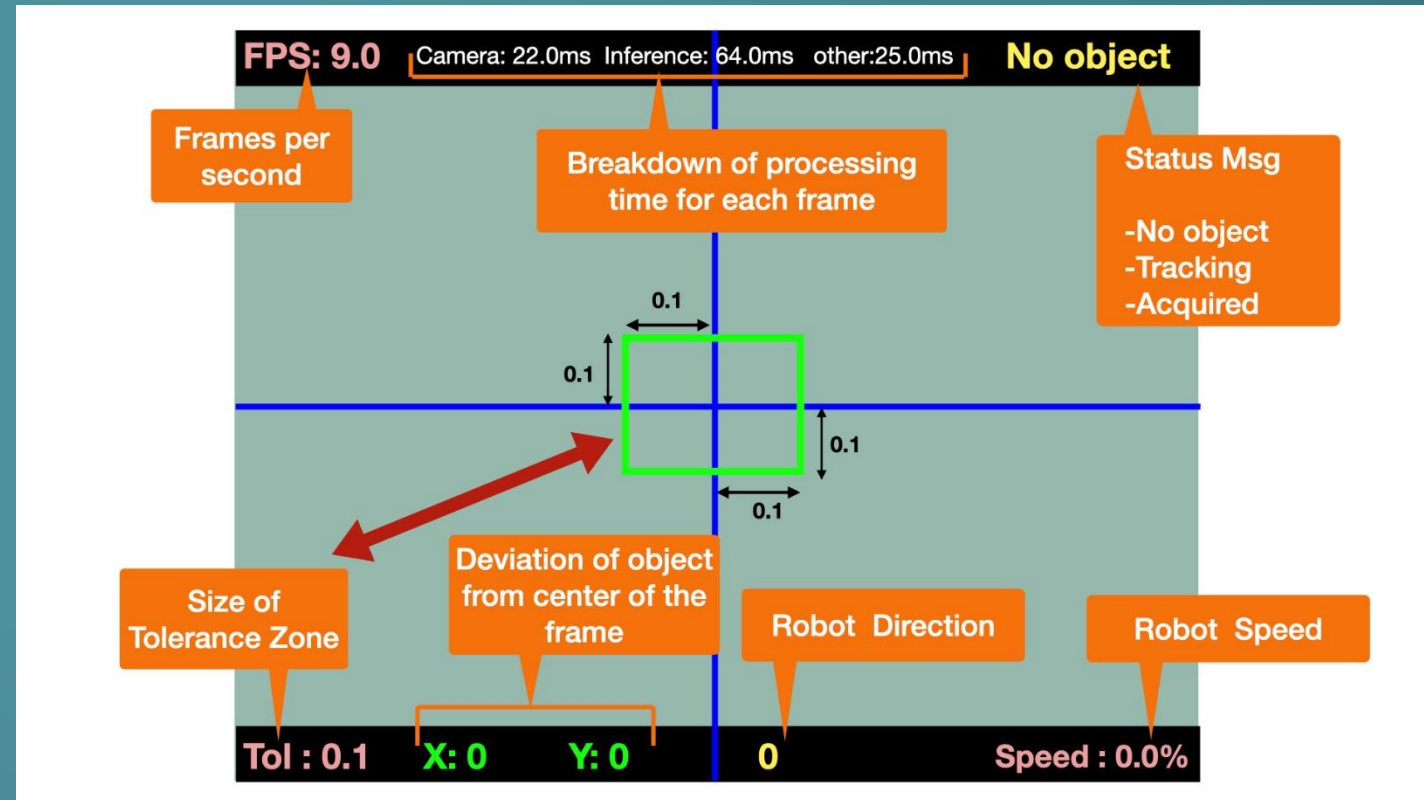
# ROBOT DESIGN CONTINUED



# OBJECT TRACKING OVERVIEW

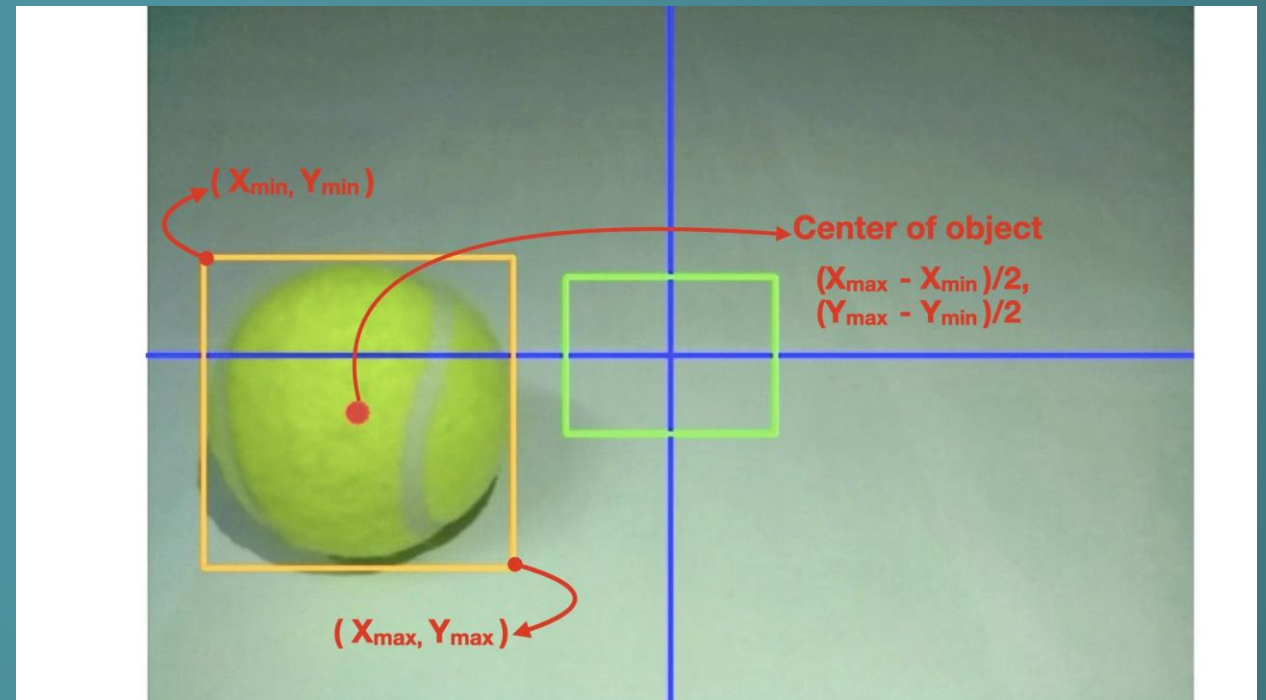
The output of inference comprises of 4 parameters. These parameters are:

1. Location of the object in frame
2. Name of the class the object belongs to
3. Confidence score
4. Number of objects in the frame



# OBJECT TRACKING MECHANISM

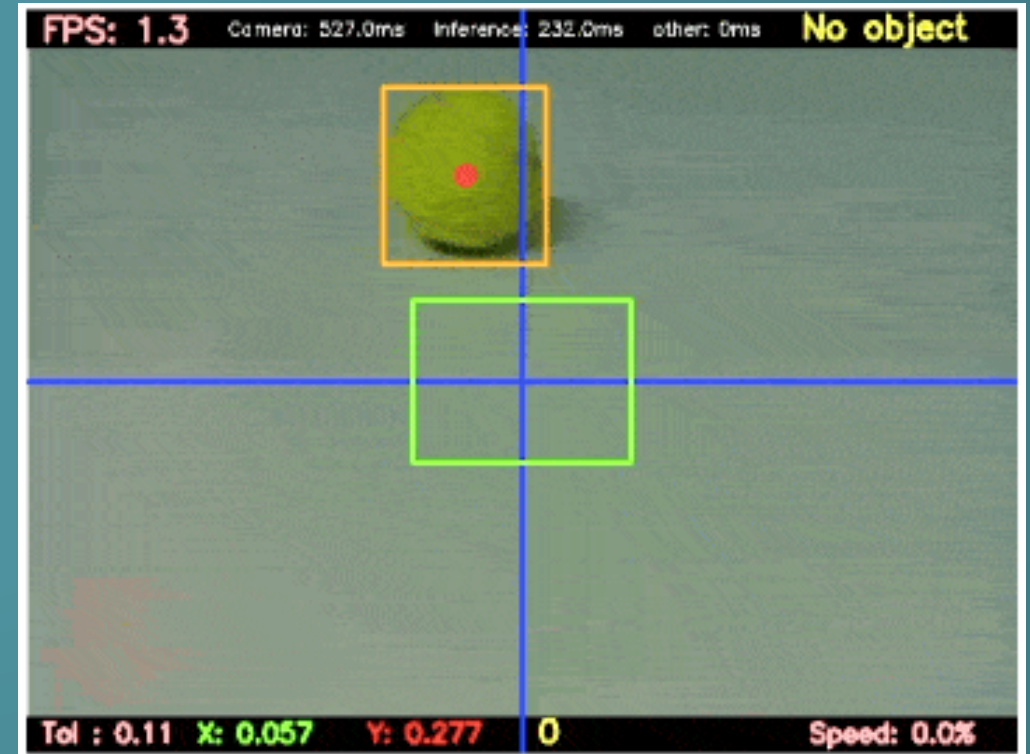
The location parameter returned by the model provides the top left and bottom right coordinates of the object. Which are used to calculate the center of the object as shown. The center is marked with a red dot. The dot follows the center as the object moves.





# DEFINING THE TOLERANCE ZONE

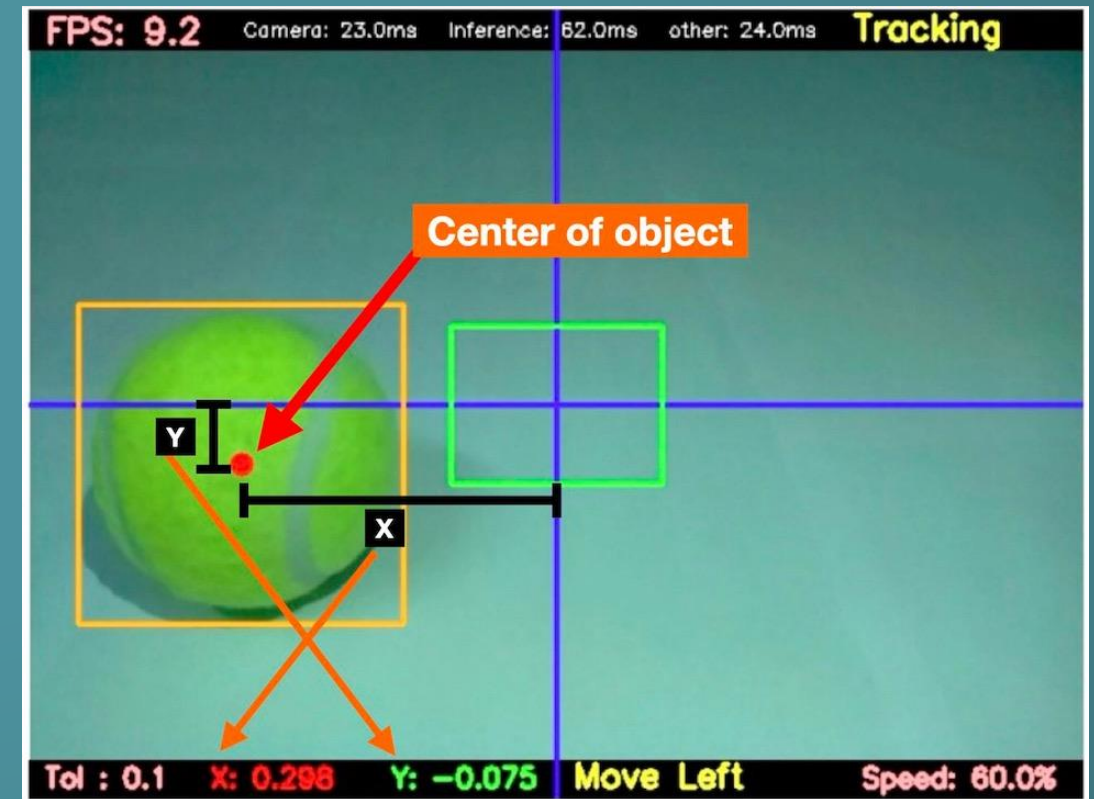
- Tolerance zone is the area around the center of the frame within which the center of the object must fall in order to stop the tracking. As long as the red dot (center of the object) is outside the tolerance zone, the robot continues to move and track the object. The tolerance value determines the size of the tolerance zone.



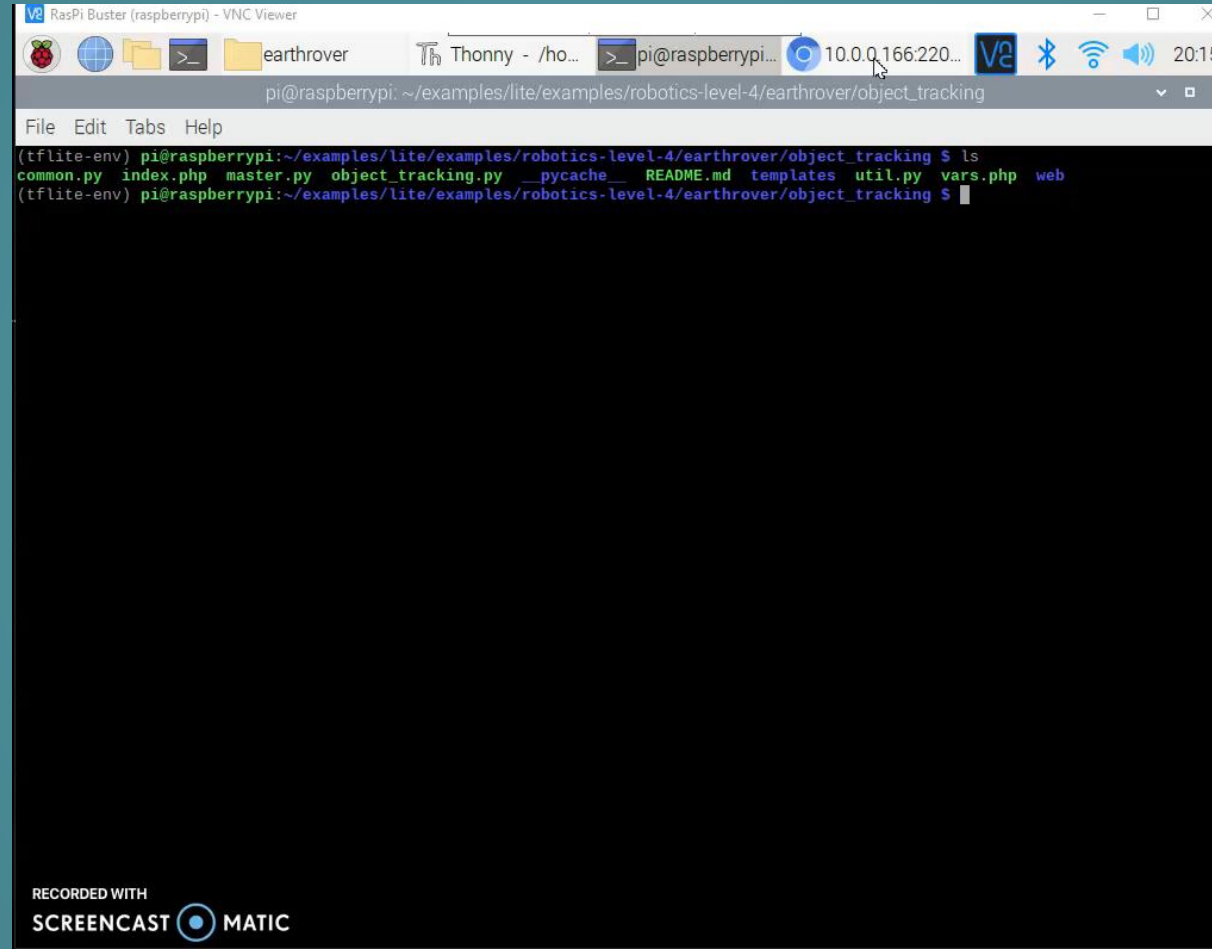


# CALCULATING THE DEVIATION


- The distance of center of the object from the center of the frame can be calculated along the horizontal and vertical axis. This distance is denoted as 'X' (horizontal deviation) and 'Y' (vertical deviation) on the frame. Both these values are compared with the tolerance value. If any of these values is more than the tolerance value, the robot continues to track.



# VIDEO EXAMPLE OF OBJECT TRACKING



```
(tflite-env) pi@raspberrypi:~/examples/lite/examples/robotics-level-4/earthrover/object_tracking $ ls
common.py  index.php  master.py  object_tracking.py  __pycache__  README.md  templates  util.py  vars.php  web
(tflite-env) pi@raspberrypi:~/examples/lite/examples/robotics-level-4/earthrover/object_tracking $
```

RECORDED WITH  
SCREENCAST  MATIC

# MOVEMENT

The aim of the object tracking algorithm is to move the robot to make both  $X$  and  $Y$  values less than the tolerance value.

The horizontal deviation or  $X$  value can be reduced by moving the robot in Left or Right direction.

The vertical deviation or  $Y$  value can be reduced by moving the robot in front or back direction.

# MOVEMENT CONTINUED

Every time a new frame is processed, the X and Y values are calculated. Depending upon the location of the object in the frame, one of the following cases is handled by the code:-

$X \text{ and } Y < \text{TolValue}$

The robot does not move.

$X > \text{TolValue}$  and  $Y < \text{TolValue}$

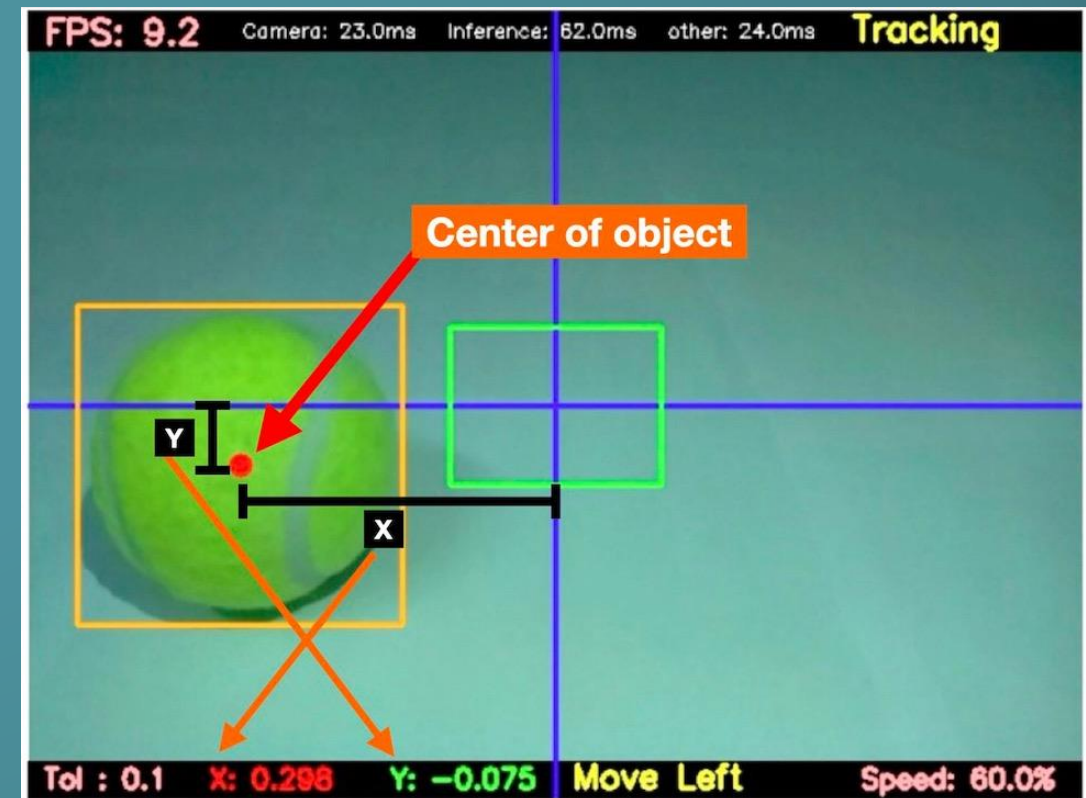
The robot moves left or right to minimize X.

$X < \text{TolValue}$  and  $Y > \text{TolValue}$

The robot moves forward or backward to minimize Y.

$X > \text{TolValue}$  and  $Y > \text{TolValue}$

The code further checks which one is greater among X and Y and moves the robot in that direction first.





# VIDEO OF FIRST TRY WITH MOVEMENT



# IMAGES SEEN BY CAMERA

In the last image, the camera sees the tennis ball, thus turns to the right in order to move towards it. However, after turning, the robot moves too fast and thus loses the view of the ball.

## First Image



## Last Image



THANK YOU