## 2. Human Following Task

### 2.1. Detection Selection

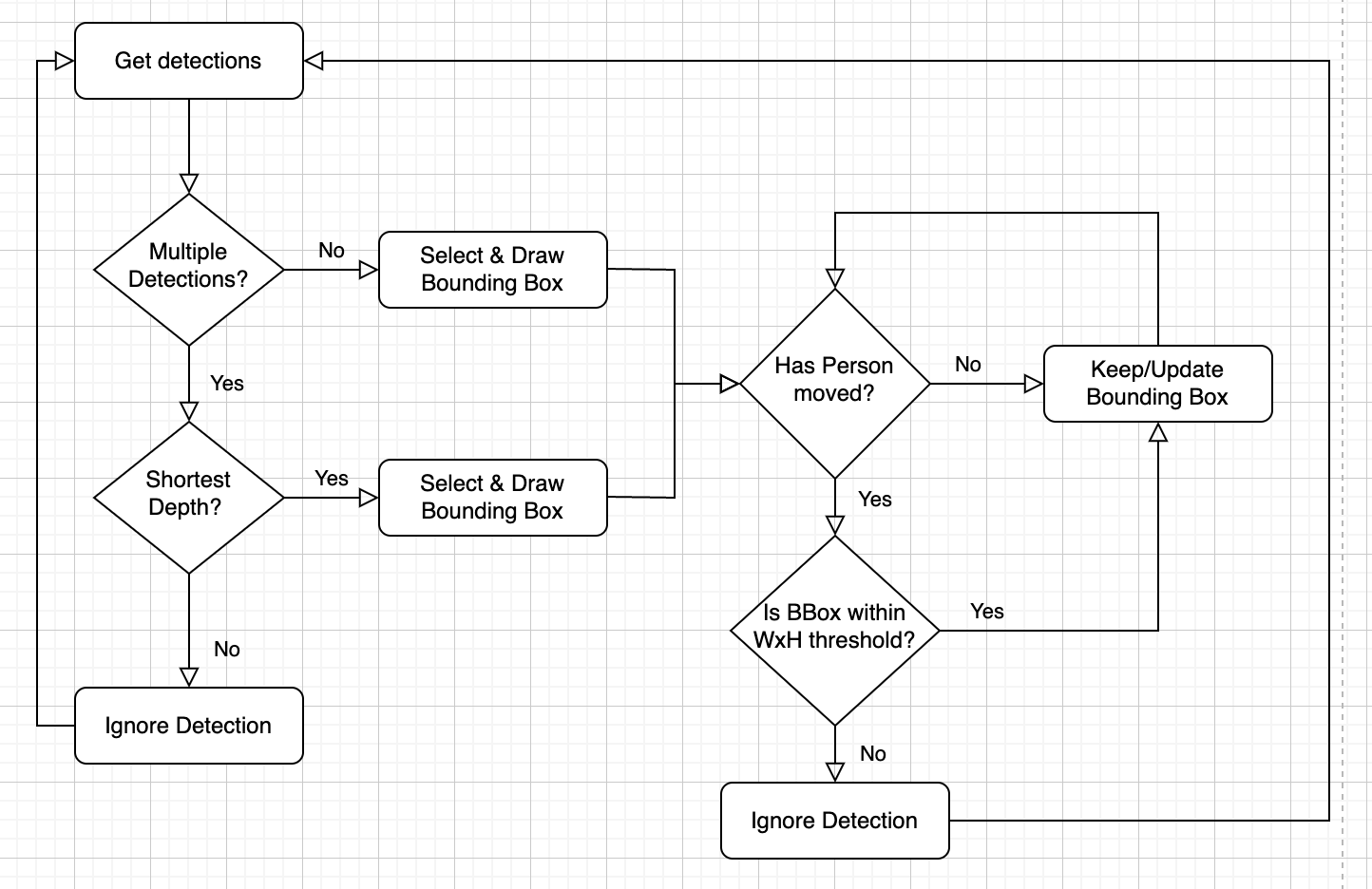
#### 2.1.1. Design

Each Human ‘target’ is defined using the pre-trained model as discussed later, the result of these definitions are captured in an image using a bounding box. This bounding box is crucial for the design and implementation for the given task, as it relays information that the target has moved resulting in the robot to respond.

The main problem regarding target detection is the process of selecting the initial target. A solution to this issue would be to implement the robot’s depth camera. This method includes choosing the ‘target’ that has the shortest (average depth inside the bounding box).…

After this problem, the next issue is determining when the detected ‘target’ has moved either to the left, right, towards or away from the robot. As this information will allow the robot to move in its required direction. This could firstly be done by defining a threshold which determines whether the next detection is the same target or another Human. This threshold would take into account the width, height and depth of the bounding box. Then examine whether the same target has moved by calculating whether the new detection is within the stored ‘target’ threshold.

The below Flow Chart describes the process of how each detection is defined and the results are given by drawing/ignoring or updating the bounding box for use in the movement section of the robot.



#### 2.1.2. Functions

As touched upon above, each detection is defined using the pre-trained convolutional neural network model MobileNet. MobileNet is used because of its low latency processing which is perfect for use in an external device such as a robot.

The pretrained model takes an image as input along with an integer label which represents the type of object that the model should detect. In this case, a person would have the label ‘1’, and after processing would return a dictionary of arrays with both; positional axis of the detected person and the confidence that the detection is a person. These positional axis are then translated using the height and the width of the whole image for an easier representation when designing and implementing bounding boxes for each detection.

Initially each bounding box is drawn using the cv2 library’s draw() function but for the specified problem, we need only the selected target’s bounding box for further processing. However, during some testing, the most common error occurs when the bounding boxes are trying to be defined when there are no initial detections. By employing an initial boolean flag that triggers when a detection is found was the fix to this issue, and when this flag is true the main loop proceeds to be executed.

Using the bounding box as discussed above, most of the information inside the bounding box is not usable, i.e. either there is too much leeway on empty space (area that isn’t human). Hence, by resizing the bounding box to be more central around the human, gives us the opportunity to implement the average depth value of the “zoomed in” boundaries.

The resizing of the bounding box was done geometrically by scaling each bounding box coordinate by a factor of 0.5 (50% decrease). This can be seen in the following lines;

**for** det **in** matching\_detections:

new\_bbox **=** det['bbox']

**for** i **in** range (len(new\_bbox)):

new\_bbox[i] **=** 0.5**\***new\_bbox[i]

new\_bbox[0] **=** new\_bbox[0] **+** 0.5**\***(new\_bbox[0] **+** new\_bbox[2])

new\_bbox[1] **=** new\_bbox[1] **+** 0.5**\***(new\_bbox[1] **+** new\_bbox[3])

Further on from this scaling in the same “for det in matching\_detections” loop, includes a conditional. This conditional compares each other detection with any new detections “new\_bbox” to see whether the bounding box is within the width and height threshold (WxH Threshold). As discussed above, these lines of code stop any other people who are walking past to be the selected ‘target’.

After the scaled bounding box and selected target are defined. Using the depth camera’s depth values allowed us to determine how far away each detection was. Which is then used for processing the robot’s forward and backward movement.

#### 2.1.3. Testing

As seen from the image below, the bounding box is scaled to a smaller square which covers all of the human. And the debug messages accompanying it shows the average depth of the bounding box and the next action of the robot when the bounding box (human) moves either to the left side or to the right.

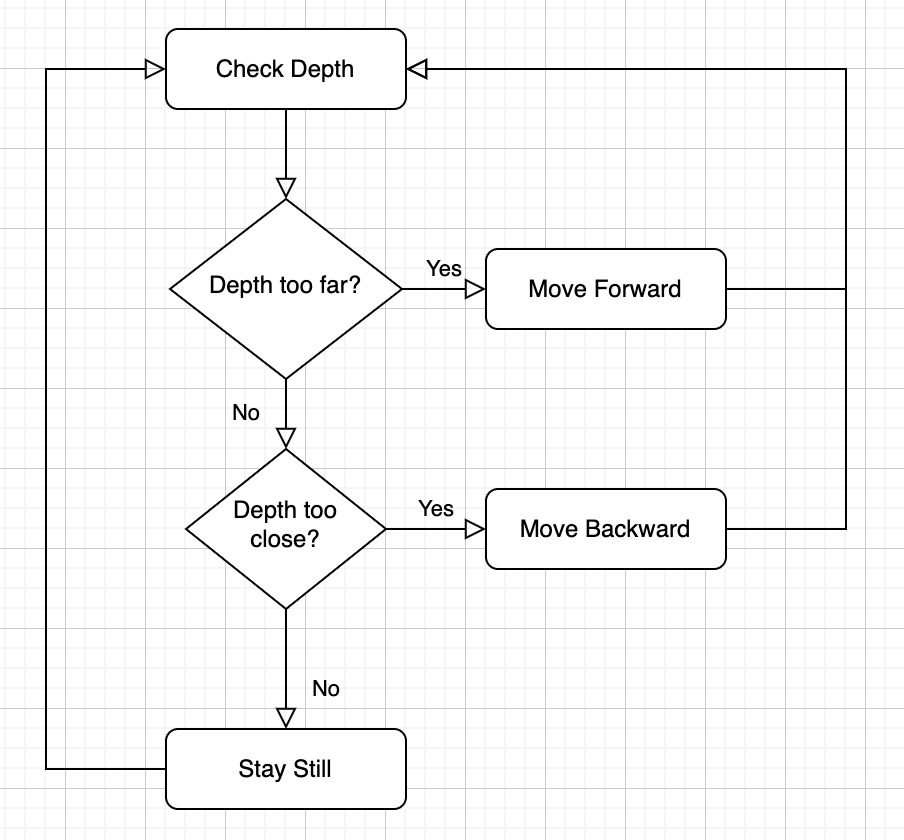
(Bounding Box & Debug Msg - Screenshots)

### 2.2. Robot Movement

#### 2.2.1. Design

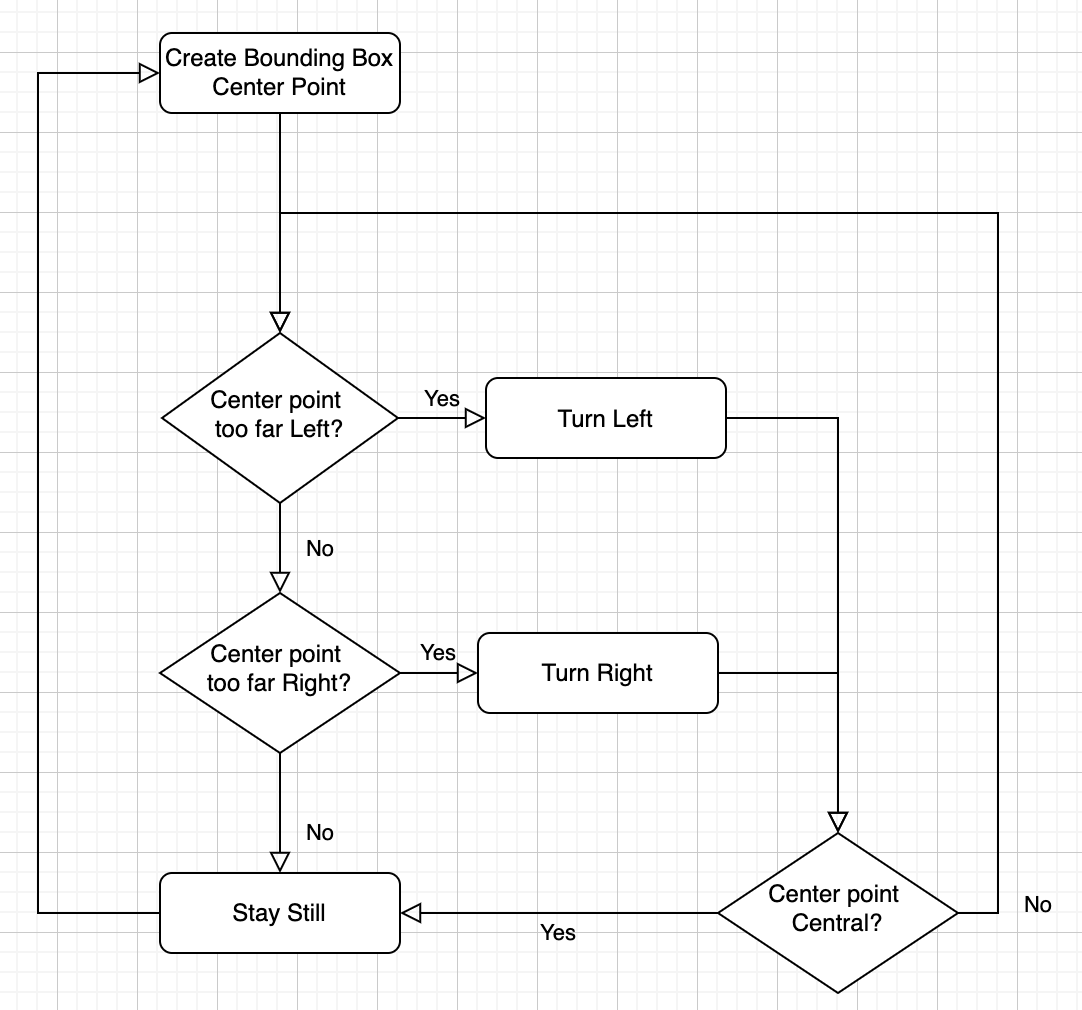
The specifications main priority regarding the robot’s movement is that the robot follows the target at a ‘safe distance’, due to the vagueness in this we had decided that the robot’s forward and backward movement would stabilise around a metre away from the target. Where, the robot being too close would result in its backward movement and vice versa.

Using the flowchart below, we can see how the robot processes whether the target is too close or too far. The starting box “Check Depth”, takes the centre point of the target and compares this value with the “Too Far” and “Too Close” variables which will be discussed later. Then as seen with both of the decision boxes below, if any of these are true the robot will move forward or backwards depending on whether it is too far or too close to the target respectively.



For the robot to turn, we decided that the best was to detect which direction the bounding box was moving based on the midpoint of the horizontal lines (Centre Point) of the bounding box, then turn the robot in the same direction. The below flow chart shows how the robot would decide which direction to turn based on the parameters set in the code.

Turning Flowchart:



The terms “Too Far Left” and “Too Far Right” are based on thresholds that are set to 30% of either side of the image. The reason for not using a 50/50 threshold is because then the robot would keep jittering left and right as it would be highly unlikely that the ‘target’ would be perfectly in the centre.

#### 2.2.2. Functions

Turning function:

**if** ((bbox[0] **+** bbox[2]) **\*** 0.5) **>** (width**\***0.7):

robot**.**right(0.5)

**elif** ((bbox[0] **+** bbox[2]) **\*** 0.5) **<** (width**\***0.3):

robot**.**left(0.5)

Forward/Backward Function:

**if** 690 < depth < 710:

robot**.**stop()

**elif** depth > 710:

robot**.**forward(0.4)

**elif** depth < 690:

robot.backward(0.4)

- Depth / RGB Camera (Forward/Backward)

- WxH Threshold (Left/Right)

#### 2.2.3. Testing

xxx

- What happens when it finds its target?

- What happens when the bounding box moves left/right?

- What happens when the target’s depth inc/dec?

- How does the robot respond to multiple targets?

### 2.3. Human Following Test

#### 2.3.1. Test variables

For this test, we have a screenshot of the bounding box’s behaviour when faced with multiple background detections.

(Screenshots/Pictures)

As seen above \*\*\*.

#### 2.3.2. Conclusion

In conclusion, the robot is able to follow its target at a “safe distance”. In accordance with this, the robot has also been tested when the target moved both to the side and backwards. This resulted in the robot turning to the human’s position followed by a forward movement of the robot. In an ideal world, the robot should curve its forward movement to show fluidity in its actions.

#### 2.3.3. Improvements

In order to improve this task, we would have liked to implement the earlier discussed function. This would be done using the robot.forward\_left(x) and robot.forward\_right(x) functions. In addition to this, we would have also liked to implement the speed of which the robot moves forward and backward. This would have been dependent on the robot’s distance away from the target.

- Slight Flickering in BBox?

- any more?