# Queue Management System using Image Processing

## Executive Summary

“7 people is the queue tipping point – any longer and shoppers won’t join the line.” (Box Technologies, 2015)

Queuing in stores in the modern era is still a concern as shoppers had to wait mindlessly in line for their turn. As such, supermarket, fashion stores, DIY stores – having the longest queue length reported by Box Technologies, are the best target in the market.

Currently in the market, there are a range of queue management system which could be costly and rigid in design. Most queue management system are stereotypical as most provide ticket and add in the queue. While it is a good system, it does not work well in a free-flowing environment such as supermarket where people can change queue.

The proposed solution is to reduce the waiting time of user by directing them to the shortest queue or alert the lengthy. This could drastically improve shopper satisfaction as they can use their time wisely instead of queuing for a long period of time.

The development period would take a month as the foundation of the solution is quite simple however it would require live-trial to adjust the algorithm and environment for better accuracy. The finished solution could be integrated with other product or act as a standalone solution.

The advantage of the proposed solution is not requiring high-end technologies. Since it can be substituted with low specification technologies, it reduces cost. Furthermore, it is designed to be malleable – the foundation to suit the general need of a queue management system; and as time goes on, improvement or upgrade can be made specifically to needs.

## Background

Box Technologies’ report in 2015 shows that 20.2 minutes was wasted on an average shopping trip while queuing. The main problem with queuing is the probability to spark tension between partner (19%) and other customers (27%). Few complaints among the shoppers are “barging in, saving places for others and couples making public displays of affection”. As a result, it can affect the shopping experience and thus reduce the chances of revisiting the store.

As store owners, it is important to reduce these negative encounters to increase shopper’s satisfaction to return next time. As such, the opportunity to implement a queue management system using camera would mitigate these problems. Currently there is a lack of these solution in the market and since cameras are already installed for security, it can be easily integrated.

### Solution

The solution to these problems is queue management through a camera. Cameras can be used to detect the length of the queue through different methods: depth sensor (high-tech) or basic webcam or security camera with basic to advance algorithms with low-tech camera. Each method has different price due to camera.

#### Motion Detection – 3/5

The simplest solution is to implement a motion detector where a static image of the ‘vacant’ queue against the livestream of the current queue. In this case, changes due to objects – environment would not affect, will change the status of the queue to ‘occupied’. As the queue builds up, it will change the queue status to ‘full’. The concept is taken from Adrian Rosebrock’s “Basic motion detection and tracking with Python and OpenCV”. The advantage of this solution is the use of low-specification technologies: Raspberry PI and camera. By following the tutorial, a queue management can be implemented through motion detection (due to background subtraction).

There are many ways to implement motion detection, tracking and analysis in OpenCV:

1. “An improved adaptive background mixture model for real-time tracking with shadow detection” (<http://www.ee.surrey.ac.uk/CVSSP/Publications/papers/KaewTraKulPong-AVBS01.pdf>) by KaewTraKulPong et al.
2. “Improved adaptive Gaussian mixture model for background subtraction” (<http://www.zoranz.net/Publications/zivkovic2004ICPR.pdf>) and “Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction” (<http://www.zoranz.net/Publications/zivkovicPRL2006.pdf>) by Zivkovic.

Both these methods are categorised as a form of Gaussian Mixture Model-based foreground and background segmentation.

Another type is Bayesian (probability) based foreground and background segmentation by Godbehere et al: “Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation”

(<http://goldberg.berkeley.edu/pubs/acc-2012-visual-tracking-final.pdf>)

**Important note! All the mentioned methods can discern motion from shadow and small lighting changes.**

2 important factors while implementing motion detection are:

1. Fixed mounted camera
2. Controlled lighting conditions.

While it is a powerful solution, it is also heavy computation. In the tutorial, Rosebrock kept it simple thus the solution can be run on Raspberry Pi and a camera.

Implementing this concept to a queue management system will have some limitation. First and major limitation is the classifying object. When object enters the screen, there is not a process to identify it as human, as such a random object change the status of the queue to occupied or full.

The lack of feature to detect human also limit the way to change the status of the queue. Since it is unable to detect human, the rules of 7 people or less could not apply to manage the queue efficiently. As such, measuring the percentage of space occupied (changes between static and current) is the only way to change the queue statue with some loss of accuracy.

Another major limitation is the setup and environment. The camera must be place directly above the queue look down. Few disadvantages of the position are listed below.

Disadvantages of fixing the camera on the front:

1. It could detect people outside of the queue (background) if there is no region of interest setup.
2. If region of interest is setup, portion of the people’s body (especially at the back of the queue – furthest away) might be in the queue while the rest is outside.
3. Object could easily overlap each other which prevent proper counting – the front blocking the back as it overlaps.
4. Lack of good segmentation which reduces the accuracy to count people.

Illustration below:

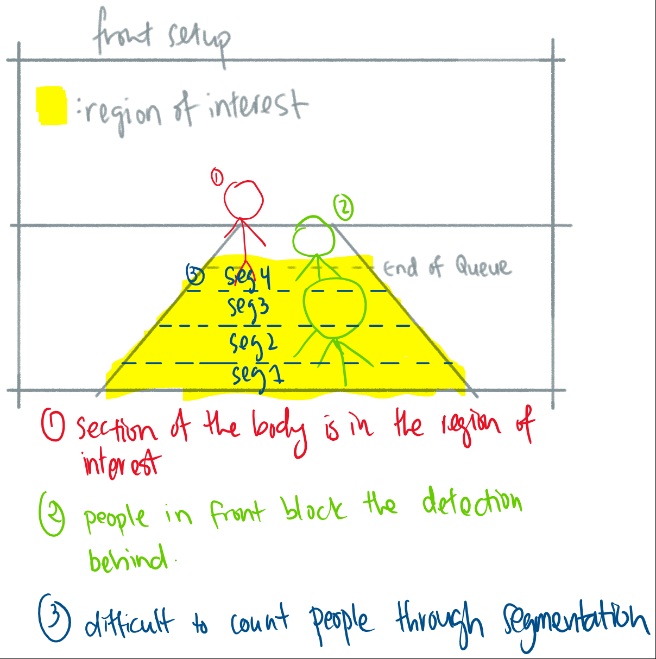


Figure 1 Front Setup Issues

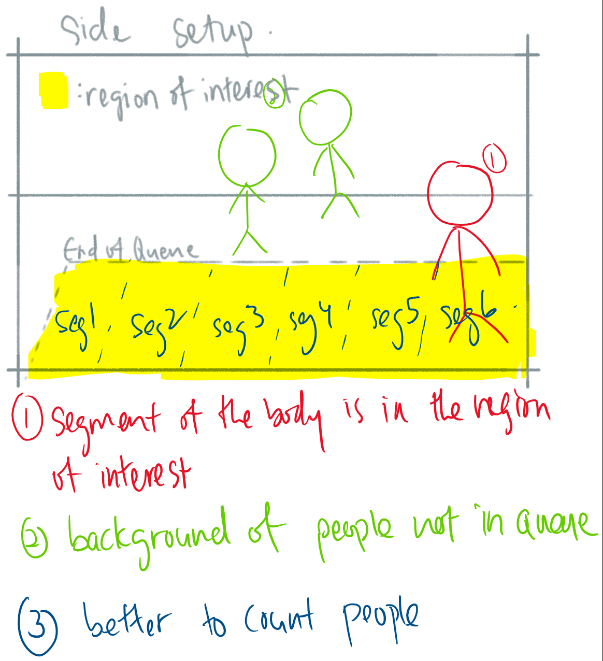


Figure 2 Side Setup Issues

Similarly, disadvantages of fixing to the side:

1. Detecting object outside of the queue – requires a region of interest.
2. Portion of the body outside the queue.

Compared the front and side view, side is better if properly setup. However, both front and side require to setup the environment to get accurate results.

The best result is obtained from the top as:

1. There is no background noise to be detected.
2. The whole body can be detected if they are not standing on the edge of the queue.
3. No overlapping of objects.
4. Best segmentation – can be more precise (smaller boundary for people)

Environment limitations can be:

1. Lighting and shadow
2. Random Objects in region of interest.

The simplest solution to the lighting problem is limiting the use of windows and having a consistent light source. Another solution is to reset the static image manually or in an interval.

The random objects issue being detection could have two solutions: integrating human detection or notify people not to place objects in the queue. Whilst the latter solution is easier, it requires the co-operation of shoppers. The first solution is better; however, with a top camera, it might be unable to detect people accurately. As such, there are pros and cons in each solution.

Below shows some example of different human detection algorithm or methods.

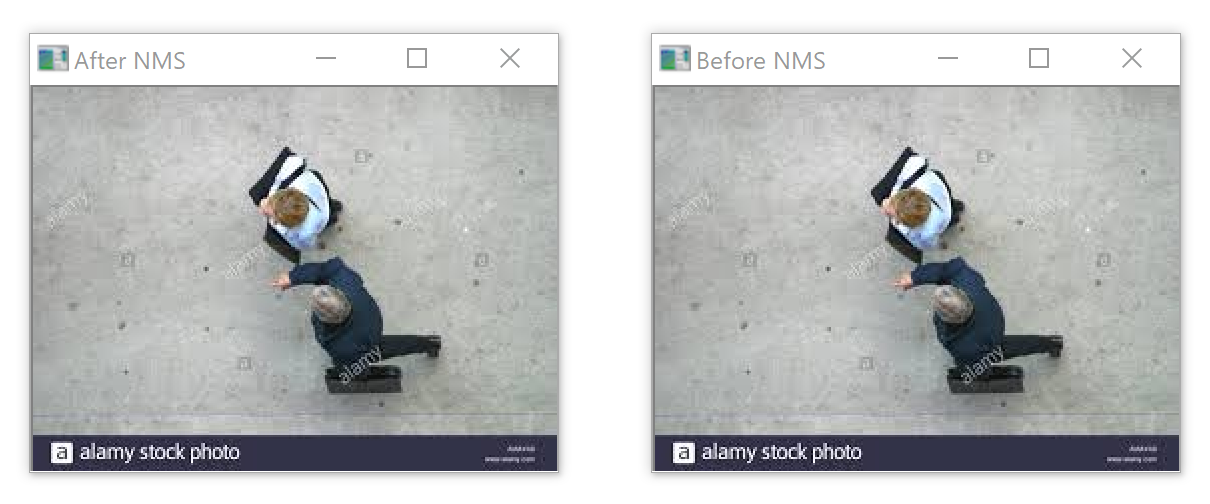


Figure 3 Pedestrian Detection

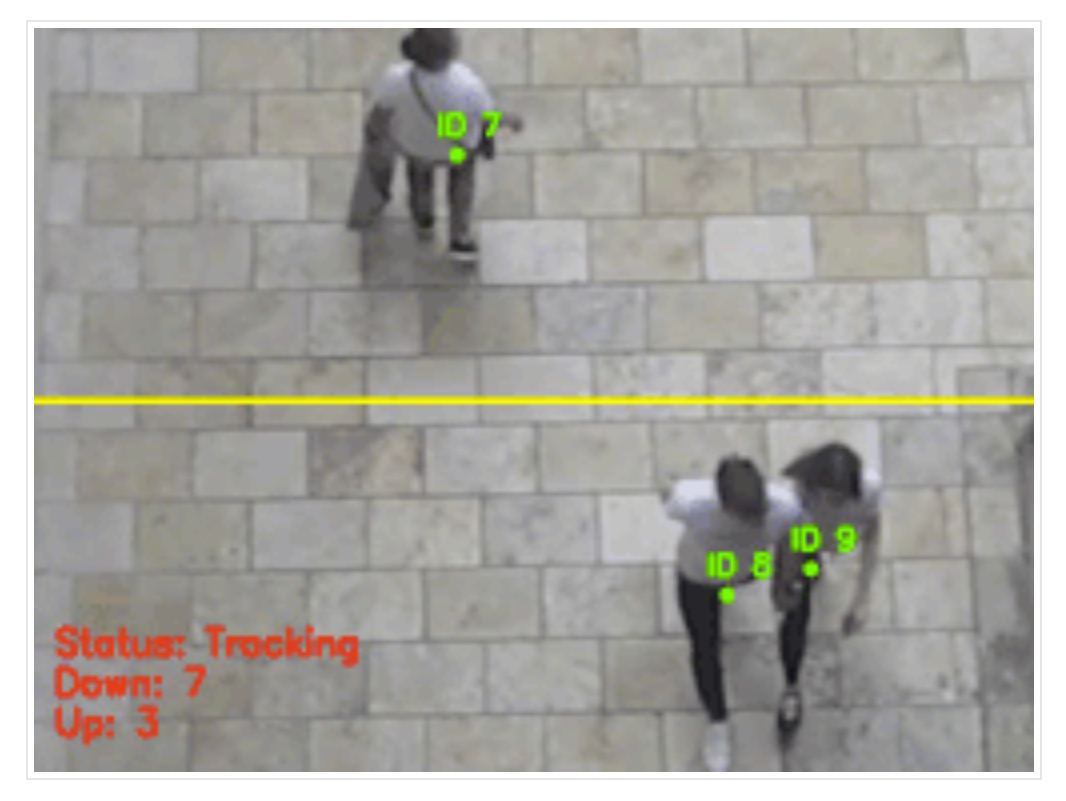


Figure 4 People Counter

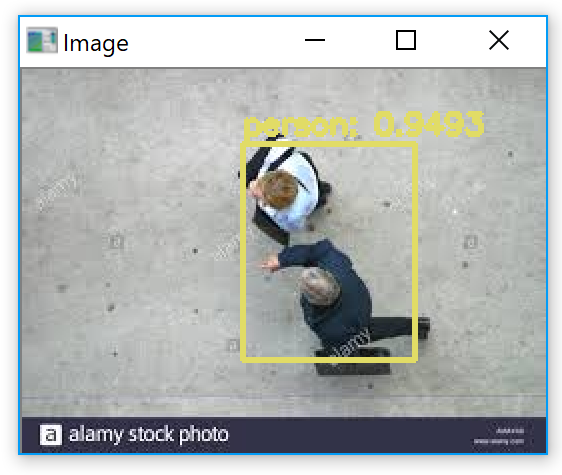


Figure 5 YOLO

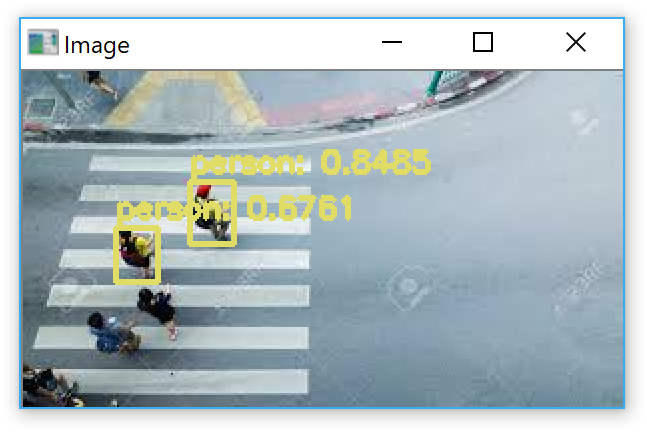


Figure 6 YOLO (Result 2)

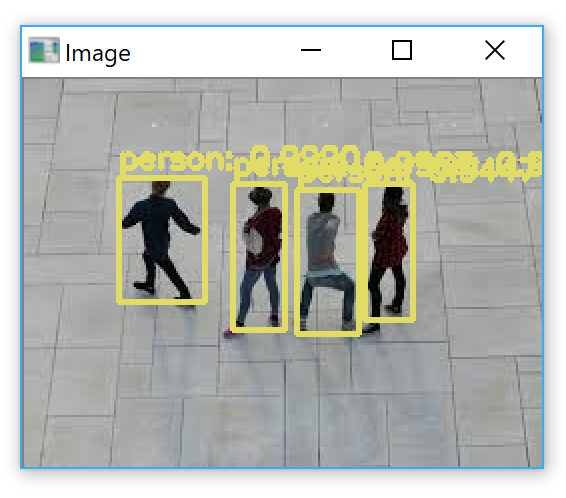


Figure 7 YOLO (Result 3)

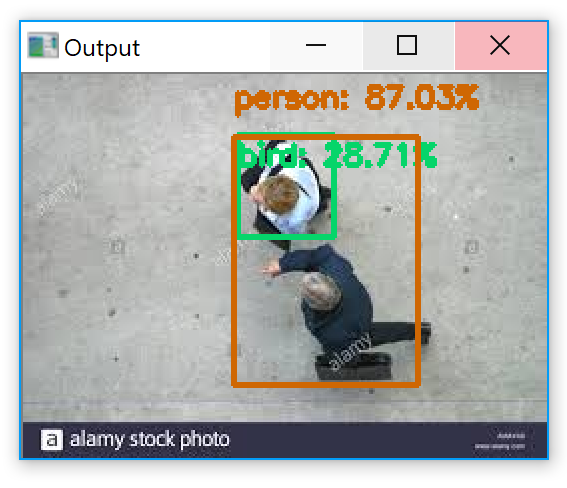


Figure 8 Object Detection with Deep Learning and OpenCV

As seen, the best result can be obtained with ‘YOLO’. However, ‘YOLO’ method is heavy computation which result in 2 to 4 seconds processing for one frame. As such, processing a video feed is not feasible. Since it is a queue management system, it might be feasible as the queue does not change every few seconds – meaning an interval of 5 second of running ‘YOLO’ could adjust the status of the queue.

Otherwise, ‘people counter’ – using the same method as ‘object detection with deep learning and openCV’ is feasible to be run all the time. Even so, in the ‘people counter’ the method is run every 30 seconds to redetect the objects – similarly to the concept of running every few second in ‘YOLO’. Although it has worse accuracy than ‘YOLO’, it is still feasible to be implemented.

#### Human Detection – 4/5

In motion detection, a few human detections are discussed. In this solution, motion detection is discarded, and human detection is used primarily. With this solution, it can easy detect the number of people in a queue and change the queue status accordingly. While it might be heavy computation – depending on the method used, it is the most accurate and feasible solution. This is due to the lack of limitation. The position of the camera is the major limitation as proper angle and field of vision is required to properly detect people. There is no restriction to the environment such as having random object in the queue – since it will not be classified as people. As discussed above, ‘YOLO’ would be ideal if processing power is strong for more accurate detection whereas ‘people counter’ is better if there is a lack of processing power.

The weakest human detection would be haar cascade as it can only detect certain angle of the face, as such a ceiling camera will not work accurately. This mean that the camera must place on the side and looking directly at the face – almost perfectly. The constraints made this solution unreliable and difficult to implement in a real environment.

##### Implementation

The pre-programmed solution is from Pyimagesearch by Adrian Rosebrock.

The ‘people counter’ (<https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/>) and ‘YOLO’ (<https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/>) can be tweaked to meet the requirements. The main change is setting a region of interest to detect the queue length in both programs base on the installation of the camera on the top of the ceiling – similarly to motion detection.

#### Depth Detection – 2/5

Another unique solution is using a depth sensor to detect the height of the object. Similarly, like the motion detection solution, the camera must be mounted on the top to detect the height of the object. There are a few issues with detection:

1. Random Object
2. Height further away from the camera

Random object can be detected if its height is tall enough, which mean eliminate the possibility of random object in region of interest either by:

1. Human detection.
2. Environment setup restriction.

And if human detection is used, it would be best to use that solution instead of adding a depth sensor detection.

The main problem with using depth sensor is the distance away from the centre of the image as illustrated below.

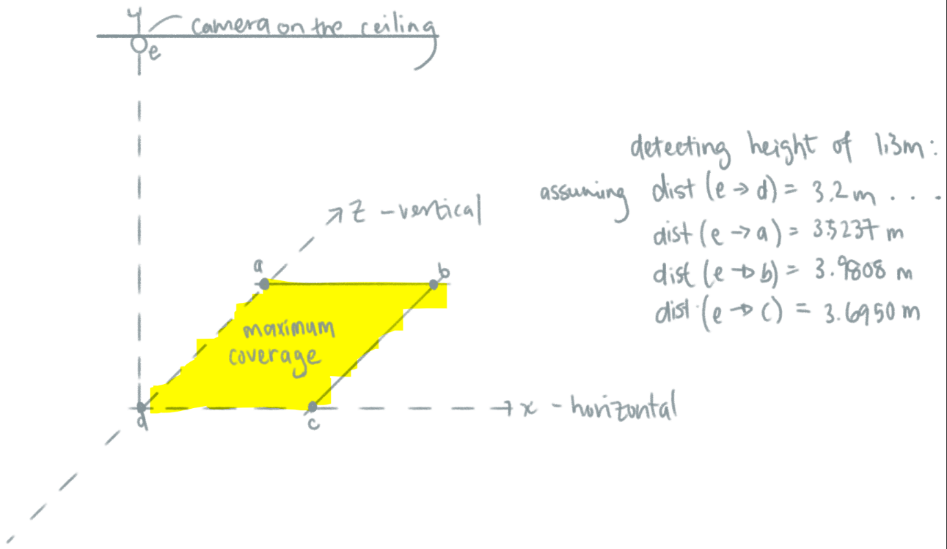


Figure 9 Distance from camera to different coordinate

The distance at the edge of the region of interest is different in length while detecting a height of 1.3 metres. As seen, the vertical at ~3.5 metres, horizontal at ~3.8 metres and diagonal at ~4 metres. This calculation is based on the Astra Orbbec camera with the specification of 49.5° vertical, 60° horizontal and 73° diagonal, field of vision.

The two solutions to solve this issue are:

1. Auto adjust threshold based on the coordinate of the object.
2. Reduce the region of interest

The first solution will be harder to implement as a relationship has to be drawn up to auto adjust the height. Furthermore, it will be more demanding on the processing as more data and detail is required to accurately detect. Even if it is done, it does not give a huge benefit as random objects are still be detected.

The second solution will be the easiest and straightforward solution in determining people of a certain height but at a reduced coverage. Below illustrate the reduce coverage calculation:

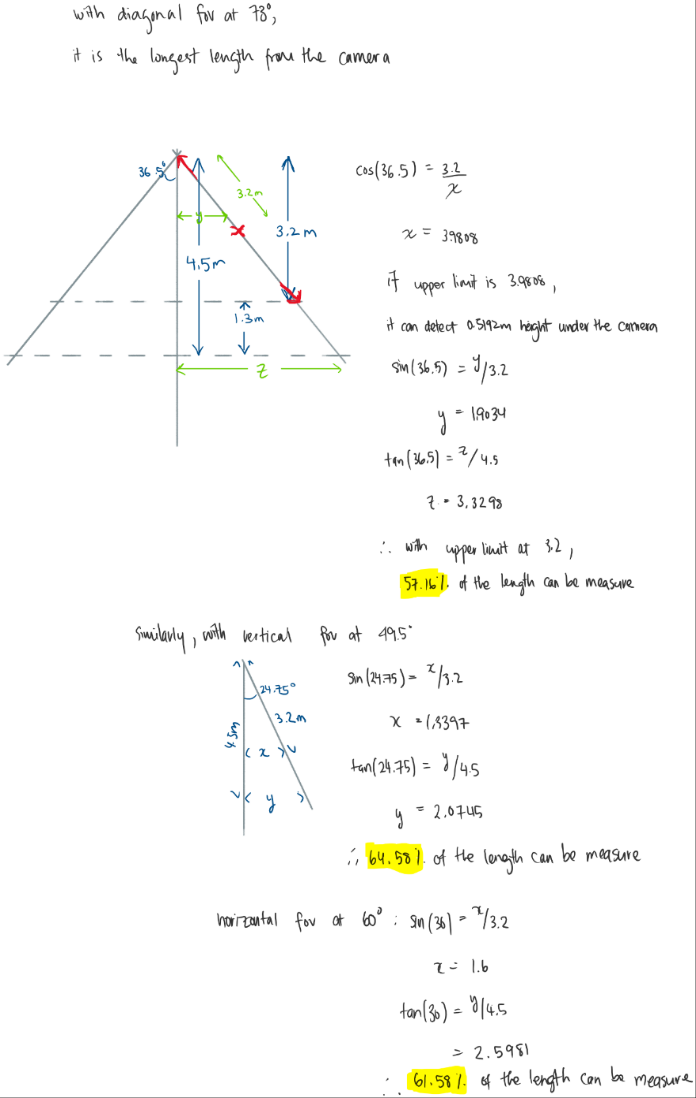
****

Figure 10 Length & Width of Restricted Queue

As shown, with a reduce coverage to detect 1.3 metres height, it can detect ~64.58% (~±1.34 metres) of the length vertically and 61.58% (±1.6 metres) of the length horizontally from the midpoint of the image. As such, the limited coverage is still applicable.

##### Basic Implementation

The depth sensor camera will be installed on the ceiling of 4.5 metres, looking directly down. With the threshold of 1.3 metres height, the region of interest is restricted as illustrated.

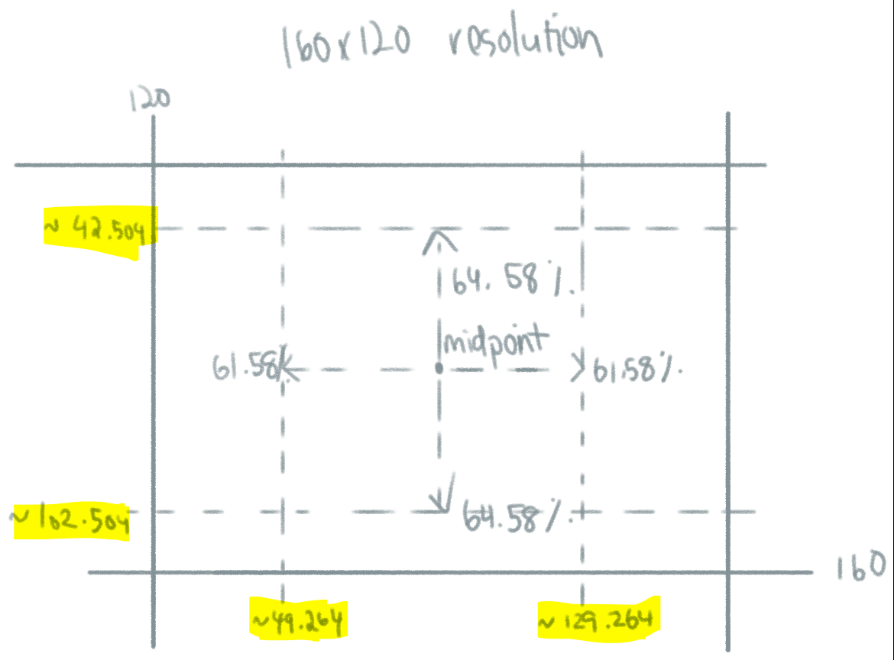


Figure 11 Region of Interest Calculation

Based on the illustration, vertically/rows of pixel will start from the 42nd to 103rd pixel, while horizontally/columns of pixel will start from the 49th to 130th pixel; however, the horizontal can be reduced further as a width of a person will not be 1.6 metres.

**Note that this is for 160x120 resolution!**

Further implementation of human detection is not required as it can be a standalone solution without depth detection.

Without human detection, there are two methods of changing queue status:

1. Measuring the percentage occupied.
2. Number of blobs.

In both cases, random objects must be removed from the region of interest or there will be inaccuracy. While ‘number if blobs’ can possibly count people – it is still not accurate enough compared to human detection, but better compared to ‘measuring the percentage occupied’.

The basic flow of the control is to:

1. Map the depth of the image
2. Check the percentage occupied or number of blobs.
   1. If threshold exceeds, send an alert message or send a vacant message.

### Final Solution and Environment Setup

The final solution will consist of using human detection (YOLO or deep learning). Depending on the hardware and condition, ‘YOLO’ would be prefer as it is more accurate and less restriction on the setup; however, at the cost of speed. ‘Deep learning’ is faster but is less accurate than ‘YOLO’. As such, running ‘YOLO’ detection can be in an interval to change the queue status. Assuming tracking is not required, ‘YOLO’ is the best otherwise ‘deep learning’ people counter.

Assuming the environment is tracking one queue, the camera can be placed directly on top looking directly down (with a slight deviation for precaution) as illustrated (not to actual size):

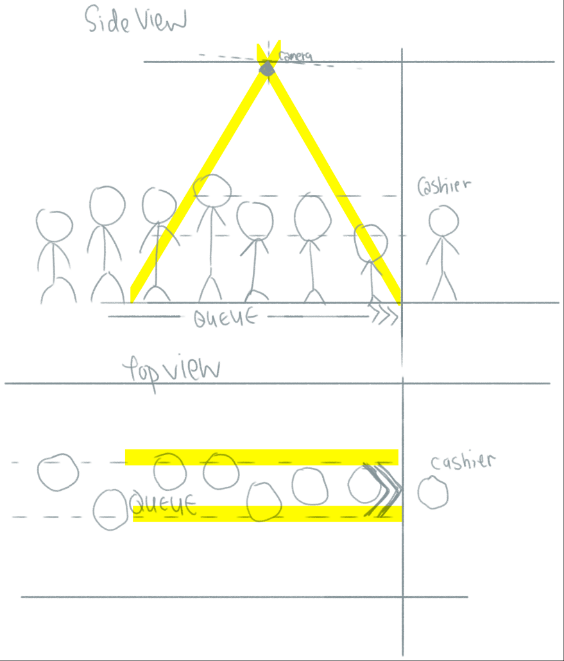


Figure 12 Possible Setup of Solution

### Resources

<https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/>

<https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/>

<https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/>

<https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/>

<https://boxtechnologies.com/wp-content/uploads/2018/03/box_how_long_does_it_take_to_lose_a_customer.pdf>

### Extra Information

3 Primary Deep Learning Object Detectors

1. R-CNN & variants (R-CNN, Fast R-CNN, Faster R-CNN)
   1. A two-stage detector thus it's slow while not a complete end-to-end object detector
   2. Very accurate
2. One-stage detector - less accurate but significantly faster
   1. Single Shot Detector (SSDs)
   2. YOLO

Other human detectors

1. Haar Cascades
2. HOG + Linear SVM

Popular network architectures compatible with OpenCV 3.3

1. GoogleLeNet
2. AlexNet
3. SqueezeNet
4. VGGNet
5. ResNet

Framework

1. Caffe
   1. prototype = network architecture (aka layer configuration)
   2. model = pretrained/training of the prototype (aka sending input)
2. TensorFlow
3. Torch/PyTorch