Computer Vision: Improving Detection and Tracking for Occluded and Blurry Settings

by

Andrei Larsen

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2021 by the
Graduate Supervisory Committee:

Ronald Askin, Co-Chair
Jorge Sefair, Co-Chair
Yezhou Yang

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

Computer vision and tracking has become an area of great interest for many reasons, including self-driving cars, identification of vehicles and drivers on roads, and security camera monitoring, all of which are expanding in the modern digital era. When working with practical systems that are constrained in multiple ways, such as video quality or viewing angle, algorithms that work well theoretically can have a high error rate in practice. This thesis studies several ways in which that error can be minimized.

This thesis describes an application in a practical system. This project is to detect, track and count people entering different lanes at an airport security checkpoint, using CCTV videos as a primary source. This thesis improves an existing algorithm that is not optimized for this particular problem and has a high error rate when comparing the algorithm counts with the true volume of users.

The high error rate is caused by many people crowding into security lanes at the same time. The camera from which footage was captured is located at a poor angle, and thus many of the people occlude each other and cause the existing algorithm to miss people. One solution is to count only heads; since heads are smaller than a full body, they will occlude less, and in addition, since the camera is angled from above, the heads in back will appear higher and will not be occluded by people in front. One of the primary improvements to the algorithm is to combine both person detections and head detections to improve the accuracy.

The proposed algorithm also improves the accuracy of detections. The existing algorithm used the COCO training dataset, which works well in scenarios where people are visible and not occluded. However, the available video quality in this project was not

very good, with people often blocking each other from the camera's view. Thus, a different training set was needed that could detect people even in poor-quality frames and with occlusion. The new training set is the first algorithmic improvement, and although occasionally performing worse, corrected the error by 7.25% on average.

This thesis is dedicated to my parents, for supporting and encouraging me in every way through college and for instilling in me all my values and inspiring me by their example.

ACKNOWLEDGMENTS

I would like to thank Dr. Ronald Askin and Dr. Jorge Sefair for their continuous support throughout this project. Their insight and advice were invaluable in writing this thesis.

I would also like to thank Dr. Yezhou Yang for serving on my graduate committee and for providing his feedback and expertise.

Finally, I would like to thank the Capstone students who assisted with this project.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1: Introduction

Computer vision is an area of growing interest and increasing research. Machine learning in general, and specifically computer vision, is often used to solve problems that would otherwise be very difficult to unravel. In practice, the accuracy of a computer vision algorithm will vary depending on factors such as video quality (Aqqa *et al.* (2019)) and training data availability (Brodley *et al.* (1999)). This thesis describes a situation that encountered such challenges along with several ways in which measurement errors can be minimized.

1.1 Project Overview

This is a thesis dealing with a practical system; thus, context is needed to describe the exact system that is being used. This project is part of a larger project, in which the DHS's Center for Accelerating Operational Efficiency, housed at Arizona State University, has partnered with the Transportation Security Administration (TSA) at Phoenix Sky Harbor International Airport to improve the performance of security checkpoints (SSCPs). Thus, the overall project combines different areas including statistics, operations research, and computer science. In order to improve the SSCP performance, an accurate count is needed of how many people enter the checkpoint at any given time. To this end, this project uses computer vision to detect, track, and count people, using security camera footage from Terminal 4A in Sky Harbor as the video source. Specifically, the goal is for the algorithm to provide counts for separate lanes in

5-minute intervals. Originally, this task was assigned to a Capstone team in Computer Systems Engineering at ASU (of which I was a member) to complete as their senior-year project. The Capstone team completed the project using a state-of-the-art detection algorithm called YOLO, but the algorithm was not tested extensively and had a high error rate. This paper focuses on improving this algorithm, both in the training phase and with detections, tracking and counting. The initial algorithm is called baseline for future reference.

1.2 Overview of Previous Work

The baseline algorithm created by the Capstone team uses You Only Look Once (YOLO) (Redmon *et al.,* 2016) to detect people and the Simple Online and Realtime Tracking (SORT) (Bewley *et al.*, 2017) tracking algorithm to track them as they enter in and out of the security lanes. Specifically, it uses YOLO to identify people and marks them with bounding boxes. A bounding box is a set of coordinates for a rectangle that contains a person; an example of a bounding box can be seen in Figure 1. SORT then tracks those people into the security lanes using those bounding boxes. The baseline algorithm is built on Python 3.6 and uses the Google TensorFlow library, which is a popular machine learning library for training and inference. The baseline algorithm is implemented based on an open-source GitHub repository (Zhang, *GitHub*) that already has the YOLO-v3 algorithm implemented using TensorFlow 2. Another reason that GitHub repository was chosen is that it comes with models pre-trained on the Common

Objects in Context (COCO) dataset (*COCO – Common Objects in Context)*, which is a state-of-the-art dataset with over 80 classes.

Using the knowledge of where people are in the video frame, the baseline algorithm tracks them into and out of security lanes using a virtual boundary created at the entrance into each lane. Figure 1 demonstrates the virtual boundaries, which are marked as black lines. Two people are detected by the algorithm in Figure 1; the person who is in the precheck lane (on the left) is marked with a red bounding box, while the person who is not in any lane is marked with a white bounding box.

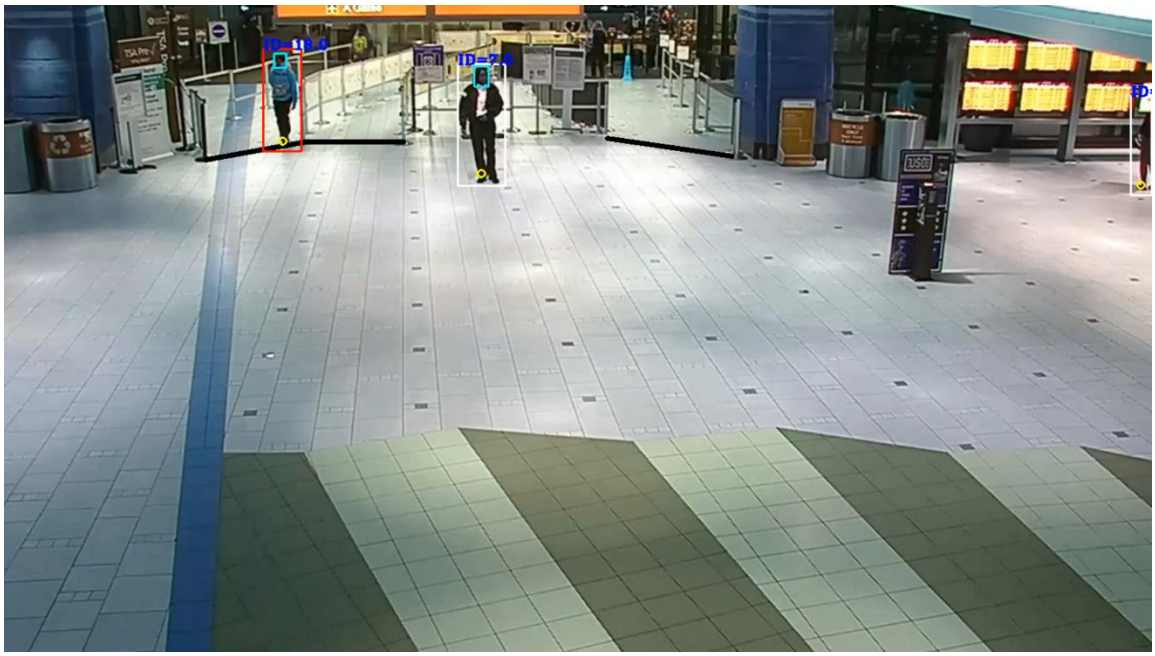

Figure 1. Line Markings and Bounding Boxes.

In theory, the algorithm can also account for people walking out of the lanes, by subtracting when anyone crossed the boundary in the reverse direction. It should be noted that any time a person is walking out of the lanes it throws the algorithm off because the tracking algorithm has difficulty tracking people walking in reverse directions right next

to each other. There is a very large error associated with the baseline algorithm, specifically, it always undercounts with an error between 20% and 40%. To fix this, an offset of ~27% was applied to the result of the baseline algorithm to help with the average error. However, the variation of the error was still very high.

There are several things that impact the performance of the baseline algorithm. By far the biggest factors are video quality and viewing angle. The video provided by the CCTV system is jumpy and blurry, which primarily affects tracking and detection of people, respectively. In addition, the security camera from which footage was taken is located at a mostly horizontal angle, which means that people often block each other from the camera's view (this is known as occlusion and will be discussed more later). An example of the mostly horizontal angle can be seen in Figure 2. In the figure, the individuals who are standing separately are identified separately; but in the regular boarding lane (on the right), the people block each other from the camera's view because of the mostly horizontal viewing angle. The baseline algorithm does not address these problems.

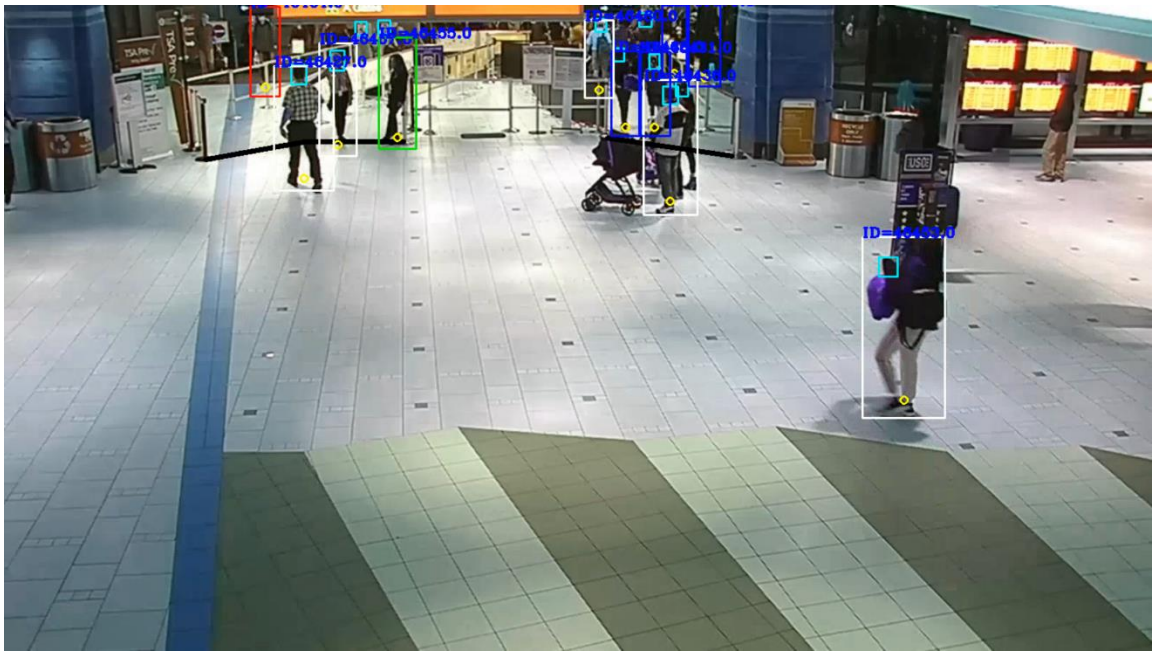

Figure 2. Illustration of the Horizontal Viewing Angle Problem.

1.3 Technical Literature

When attempting to improve the algorithm, it is important to consider not only the factors that impacted its performance, such as video quality, but also the underlying algorithms. Given the proliferation of algorithms for computer vision, it is possible that different algorithms could have better results than the ones used in the baseline algorithm.

Kim *et al.* (2019) indicates that while Region-Based Convolutional Neural Networks (RCNNs) are by far the most accurate, the full version of YOLO-v3 provides pretty accurate results in a reasonable amount of time. Of course, while this thesis seeks to make the overarching algorithm as accurate as possible, it is important to consider that it cannot take too long to run, both because of the limited time available for testing and because of the possibility that it will be used in near real-time, if deployed. Kim *et al.*

(2020) suggests that YOLO-v4 provides the best real-time results with 93% accuracy. After this research, it was decided to keep using YOLO, but to update the program to use YOLO-v4 instead of YOLO-v3 to make use of the latest developments in the algorithm.

As reported in Bewley *et al.* (2017), the SORT tracking algorithm uses rather simple tracking techniques, but its accuracy is still comparable to other trackers. One benefit of SORT's simple tracking techniques is that it can perform at speeds up to 20x faster than other state-of-the-art trackers, which means it can be used in real-time, while the speed of the most accurate trackers is too slow for real-time. Thus, the decision was made to continue using SORT. It should be noted that there is not much other literature on comparison of the accuracy of SORT vs. other trackers, for example the trackers provided by OpenCV (*Introduction to OpenCV Tracker*). The decision was made to still use SORT without testing any other trackers, since there was no clear benefit to switching tracking algorithms.

The baseline algorithm uses the COCO dataset. This dataset is a state-of-the-art dataset with 80 classes, including people as well as different animals and furniture. The proposed algorithm uses a different dataset, for two reasons – because the COCO dataset does not have a class for heads and because the COCO dataset does not contain very many occluded people or people in poor quality frames. These reasons are discussed in greater detail in Section 1.4. Some compatibility issues arose when trying to train using the interface provided in the cloned GitHub repository, and after some research, it became clear that the author of the YOLO algorithm implemented his own algorithm, with support provided for training. This implementation is called Darknet (Redmon 2016). Thus, it seemed to make sense to switch to Darknet, the implementation that was

created by the author of YOLO himself. However, Darknet is not based on the TensorFlow library, and given that TensorFlow is very widely used, it seemed probable that the new implementation without TensorFlow might be slower. Although there is not much literature on comparing the two, Kromann (2018) in a blog published on LinkedIn did an in-depth comparison between Darknet and Darkflow (which is Darknet with TensorFlow, essentially what the baseline algorithm uses). The results show that Darknet was up to 2x faster than Darkflow; in addition, Darkflow is implemented with YOLO-v2, even though YOLO-v4 is already available. Given these factors, namely, ease of training, speed, and YOLO version, the decision was made to switch to Darknet.

The greatest cause of error in the baseline algorithm is caused by crowds of people, because people in crowds occlude each other and the algorithm cannot count them accurately. There is some literature that provides various methods for dealing with crowded scenarios in computer vision; however, current methods that are designed for crowd flow estimation, such as the one described by Behera *et al.* (2020), will not work for this problem, since they are designed to estimate overall crowd flow for large crowds, while the scenario for this project deals with relatively small crowds in which the people must be tracked and counted individually, and the overall crowd flow is not as important. There are also methods for density estimation and approximate crowd counting available, such as the one described by Sindagi (2017), but the algorithm presented in this thesis must focus on the tracking component, since passengers only get counted when tracked into a security lane. The overall crowd size, while important to know because of its impact on the error, is not actually used directly to get counts. Thus, current crowd counting techniques do not work well for this project.

There are also algorithms for counting/tracking objects in surveillance videos. For example, Kim *et al.* (2011) proposes an algorithm that uses background modeling to detect moving objects. The algorithm proposes several improvements, including used colored (RGB) images instead of grayscale images, which decreases processing time by removing the requirement of processing the images from color to black-and-white; since using RBG images results in increased sensitivity to even small light changes, the algorithm uses a sensitivity factor and ends up getting results better than standard greyscale algorithms. To track the objects through frames, the algorithm groups any moving objects and attempts to predict the groups' trajectories. It updates the predictions as the groups' actual motion is detected. Some aspects of this algorithm could be beneficial to the algorithm proposed by this thesis, but the specific task of the proposed algorithm is to count individuals as they enter security lanes; thus, tracking people in groups would not be helpful. Overall, the algorithm proposed by Kim *et al.* (2011) appears to be geared toward detecting and tracking objects through security cameras in order to replace the need for humans watching the camera feed; it does not work for this exact project.

Another algorithm for tracking and counting objects in surveillance video is proposed by Prakash *et al.* (2014). It differs from the previous one by using Extended Kalman Filters instead of background modeling for detecting moving objects. However, it is also geared toward replacing the need for a human overseer, specifically by identifying motion of "groups", whether it be one object or multiple; while the paper does mention that counting the objects could be a final step in the process, it does not propose

a method for doing so. Since the task of the algorithm proposed in this thesis is to count people, an algorithm that tracks group motion is not particularly helpful.

A method of counting objects (presumably people) in crowds is proposed by Rabaud *et al.* (2006). The algorithm uses an optimized version of the KLT tracker, which is a tracker that uses feature extraction to identify objects across frames. The algorithm uses spatial and temporal constraints to smooth the trajectories. A benefit of this algorithm is that it always considers the possibility that one object is really two objects, one of which is occluded by the other. Thus, it can get fairly accurate counts. However, it is designed for very crowded environments; the environment of this project can get somewhat crowded, but it is unusual for the entire frame to fill up with moving people. In addition, the purpose of the algorithm proposed by this thesis is to count people entering the security lanes, while the algorithm proposed by Rabaud *et al.* proposes a method for counting people generally. In addition, the use of the KLT tracker, which can most likely perform better than the SORT tracker when dealing with occlusions, is not necessary. As explained in Section 2.3, the proposed algorithm only needs to track a person across 2 frames: before and after s/he enters the security lane. Thus, the SORT tracker is enough. In conclusion, the algorithm proposed by Rabaud *et al.* could be used to estimate crowd size, which could be used to estimate the error, but it does not solve the problem this thesis attempts to answer.

1.4 Improvements to the Algorithm

This section describes the problems encountered in the baseline algorithm, improvements made to the algorithm, and the thought processes that lead to the improvements. It will also mention some modifications that were attempted but did not improve the algorithm.

One of the major problems with the baseline algorithm is that it does not account for the camera position. In the CCTV videos available, the camera does not have a very good angle, and therefore people often occlude each other from the camera's view. One of the first things that comes to mind when encountered with the problem of occlusion is to use a dataset that is trained for occluded objects. For example, a good car dataset would include full cars and the front and rear parts of a car, so that if a car were sticking out behind a building it could still be identified as a car by the trained model. Another option is whether a different object can be detected that would not occlude as much. In this case, since the algorithm is trying to count people, it is possible that detecting heads instead of full bodies would have less occlusion for two reasons: first, because heads are smaller and are thus less likely to get in each other's way; and second, because the camera has a slight upward angle over the people, so heads in back are likely to be visible to the camera even if the bodies are occluded. An example of this is shown in Figure 2 in Section 1.2, where several heads are detected in the crowd of people in the regular lane.

The CCTV videos available are of poor quality. This is not an algorithmic problem specifically but is more of a problem with the data provided to the baseline algorithm. The videos were originally provided on DVDs with a proprietary embedded

player, which meant that to get the videos for processing they had to be screen-recorded while playing, which only served to make the quality worse. Moreover, the video occasionally lags and skips frames. The choppy quality of the video does not affect detection, but it adversely affects tracking. While this is not a huge issue, it certainly contributes to the error. Poor quality videos and pictures used for detection can be a major problem for computer vision algorithms (Marciniak *et al.*, 2015). The reasons for this are complex, and there are ways to use filters and smoothing to partially overcome the issues of poor-quality frames. Another possibility is to get a new training set, one that is trained to better detect people in low-quality images.

There are several ways to deal with occlusions. The usual way is to look at the appearance of the detected object to see if any parts of it appear different from before; this would indicate an occlusion. A large body of research has come out detailing conceptual ways to deal with occlusion; Pan *et al.* (2007) utilizes spatiotemporal information around the object, while other methods involve including the occluding object into the modeling, in an attempt to find patterns in the most common occluding offenders (Pepik *et al.*, 2013). However, these approaches do not work in the problem environment because meaningful spatial/spatiotemporal information would be difficult to obtain given the poor video quality and constant full occlusion of people in crowds. Including the occluding object into the model would not really change anything, because people are the occluding object as well as the occluded object. Thus, another method to counter the occlusion problem had to be used.

The problems described above lead to the decision of using detection of heads to improve tracking and counting, specifically whenever it got crowded. However, the

baseline algorithm, which uses the COCO dataset, is not trained to detect heads. This is another reason to use a different training dataset. An ideal new dataset would include lots of people in low-quality frames, lots of occluded people, and annotations for heads. The CrowdHuman dataset (Shao *et al.*, 2018) is a perfect fit for this, because it includes full annotations of people, partial annotations of occluded people, heads, and people in image backgrounds and small corners of images (which correspond to low-quality areas).

The use of the CrowdHuman dataset instead of the generic COCO dataset improved the performance of the baseline algorithm. Even without using any head detections, the algorithm's total counting error over a 2-hour period dropped to between 0.5% - 17%, while the baseline algorithm with COCO model had errors between 0.7% - 56%. The head detections provide further improvements.

The simplest way to improve the algorithm is to use the total headcount to estimate the crowdedness of the frame. This should work because in theory, the more crowded it is, the more heads will be present. In the same setting, more occlusion will occur, and hence a higher error will be present.

The second method of utilizing the headcount to minimize the error is to identify any events when there is more than one head identified inside of a person's bounding-box. The most likely cause of such an event is when two people are walking so close to each other that they appear as one person to the model; however, since the two heads would still be distinguishable, the person could be counted as two. Practically, this works well for small crowds, but when it gets too crowded there are so many people and heads in the same area that the algorithm ends up overcounting. Thus, for larger crowds a third solution is needed.

The third method is similar to the first method, except that instead of utilizing the overall number of heads in the frame it will count heads in specific areas of the frame where crowding might occur, specifically around the entry points to the security lanes. This can serve two purposes: the headcount can be used by the algorithm to determine an approximate offset for the error, and the headcount can determine if the crowd is small enough for the second method to be used.

The proposed algorithm uses a combination of the second and third approaches, depending on the crowd volume. The first approach of estimating total crowdedness is too general and no meaningful correlation was found between the total number of heads and the error.

Chapter 2: Technical Components

This chapter lists and explains all the implementation details including the development environment and the libraries used. This is for the purpose of replicability and to assist anyone who seeks to improve or test the proposed algorithm. This chapter also includes some pseudocode. The actual code implementation is available upon request.

2.1 Development Environment

The proposed algorithm is implemented in Python 3.6 and uses the Darknet shared python library. In order to facilitate replication of this project for any who desire to do so, detailed information about the system on which this project was developed and tested is provided. The project was developed and tested on an MSI GE63 Raider with an NVIDIA RTX 2080 graphics card. The operating system is Linux Mint 19, and the NVIDIA driver version is 440. The CUDA version is 10.2. FFmpeg is used to write output video files for demos and debugging. OpenCV 4.5.1, compiled from source, is used to decode the input video and to display real-time video output. The training data is available from the CrowdHuman dataset. Additional information about specific Python libraries, Makefiles used to compile Darknet, and scripts used to compile OpenCV are available upon request.

2.2 Algorithm Implementation

The proposed algorithm is based on the theoretical details described in Chapter 1 and it requires the prior training of the model. The goal for this algorithm is to provide an accurate people count for 5-minute intervals. The focus of development and testing is to count people entering the regular boarding and precheck lanes. The clear lane has a low number of passengers compared to the other lanes (between 4% - 8% of all passengers based on the manual counts used in testing), thus it is not very important for improving performance at checkpoints; in addition, there is almost always a staff member standing in front of the clear lane, which greatly increases the error to an unusable amount.

Despite any improvements, there is an inherent error in the raw count provided by the algorithm. Using the principle of augmenting the count whenever two heads are detected inside of one person bounding-box is overcorrecting, because there are often times when it gets too crowded. In order to average out the error, the algorithm samples the result every 30 seconds, and then sums the last 10 samples every 5 minutes. The algorithm first uses headcount to determine the crowd size in proximity to the security lane entrance, and then uses the crowd size to "weigh" the augmentation. This is best demonstrated in pseudocode, which is shown below. Code snippets 1 and 2 are used to explain the variables in code snippet 3, which is the one that illustrates the augmentation procedure. For sake of brevity, simple algebraic components of the code are not shown.

Code Snippet 1 describes how the algorithm determines the extra count (augmentation) whenever multiple heads are detected inside 1 person's bounding box.

Code Snippet 1 Count the augmentation due to headcount
aug represents the augmentation due to headcount
headbox represents the position of the head bounding-box
personbox represents the position of the person bounding-box
headlist represents a list of all heads in the frame
personlist represents a list of all people in the frame

```
for personbox in personlist
    if personbox is entering lane then
        heads = 0
        for headbox in headlist
            if headbox is in personbox then
                heads += 1
        if heads > 1
            aug += heads - 1
```

Code Snippet 2 describes how the algorithm determines the size of a crowd and how it stores that information in an array for later use.

Code Snippet 2 Determines the crowd size next to a security lane entrance
headbox represents the coordinates of the head bounding-box
rect represents the position of a rectangle; everything inside that rectangle is considered "in proximity" to the lane
headlist represents a list of all heads in the frame
crowdsize is an array that represents the number of times the crowd in front of a security lane reached a certain number of people. For example, if there was 11 frames in which 9 people were in proximity to the security lane entrance, then `crowdsize[9] = 11`

```
heads = 0
for headbox in headlist
    if headbox is in rect then
        heads += 1
for i = 0 to 19
    if heads == i then
        crowdsize[i] += 1
```

Code Snippet 3 describes how the augmentation from Code Snippet 1 and the crowd size array from Code Snippet 2 are combined via experimentally determined weights to get an accurate count.

Code Snippet 3 Demonstrate crowd-size headcount weighting
aug represents the augmentation calculated above
val represents the number of people counted crossing into the security lane
crowdsize represents the crowd size in front of a security lane. See code snippet 2
res represents the algorithm's final output for how many people were counted in the last 30 seconds

```
aug = aug if aug < val else val
if crowdsize[12] > 0 then
    res = 0
else if crowdsize[6] > 0 then
    res = crowdsize[7] * aug * 0.1
    res += crowdsize[8] * aug * 0.2
    res += crowdsize[9] * aug * 0.3
    res += crowdsize[10] * aug * 0.4
    res += crowdsize[11] * aug * 0.5
    res /= sum(crowdsize[6:11])
else
    res = aug * 0.5
res += val
```

The first two code snippets are run every frame, while the third code snippet is run every 30 seconds. The first line of code snippet 3 makes sure that the augmentation is not greater than the actual number of people counted entering the security lane; leaving it out results in greater variation in the error. The first "if" statement (on the second line) is used whenever a crowd of 12 people was detected. In such crowded environments, the augmentation is prone to be inaccurate and it is best not to use it. This was determined experimentally. Since it has been the case that crowds cause undercounting, it seems counterintuitive that when it gets more crowded the augmentation is unneeded. However,

it is likely that the jittery frames and the small changes in bounding box locations combine to cause the tracker to count people crossing the lane boundary multiple times, which actually fixes the error. The “else if” statement is used whenever a crowd of medium size is detected. Under such circumstances, the weighted augmentation works best. The “else” statement is used for non-crowded situations. In this case, the augmentation is still too high but when halved is fairly accurate. The last line adds the actual number of people counted.

It should be mentioned that these crowd size cutoffs of 12 and 6, as well as the weights from 0.1 to 0.5, and the final weight of 0.5, were determined experimentally, as explained below. These values work extremely well for the regular boarding lane, and different values are used for the precheck lane. It will likely be necessary to recalibrate these numbers if this algorithm is applied to a different project environment. However, the code has support for printing out all of the information at 30-second intervals, which would allow the user to run quick statistical analysis to determine the best weights. In the case of the weights used in this project, they were tweaked experimentally, by first starting with equal weights and then changing the weights, first in increasing, then decreasing order. Using an increasing order of weights provided good results.

As can be seen in Figure 3, without headcount augmentation, the algorithm’s result is always just below the correct value, and the headcount was supposed to result only in a small increase, enough to correct that small error.

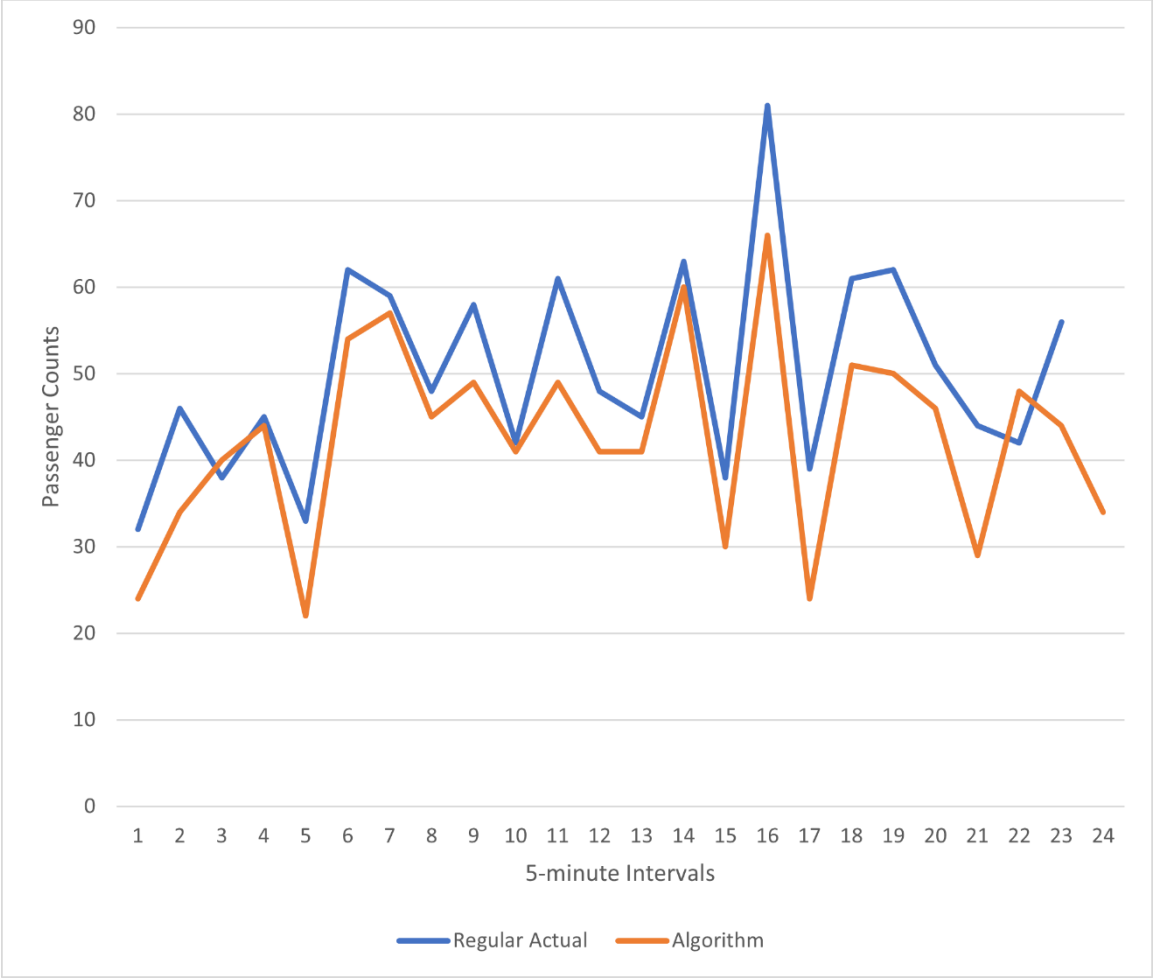

Figure 3. No Headcount Augmentation.

Using the crowd size to weight the headcount augmentation was not originally planned as part of the algorithm. However, when simply adding the headcount augmentation, the algorithm overcounted by a lot; one noticeable thing is that it overcounted more when the number of passengers went up, as seen in Figure 4.

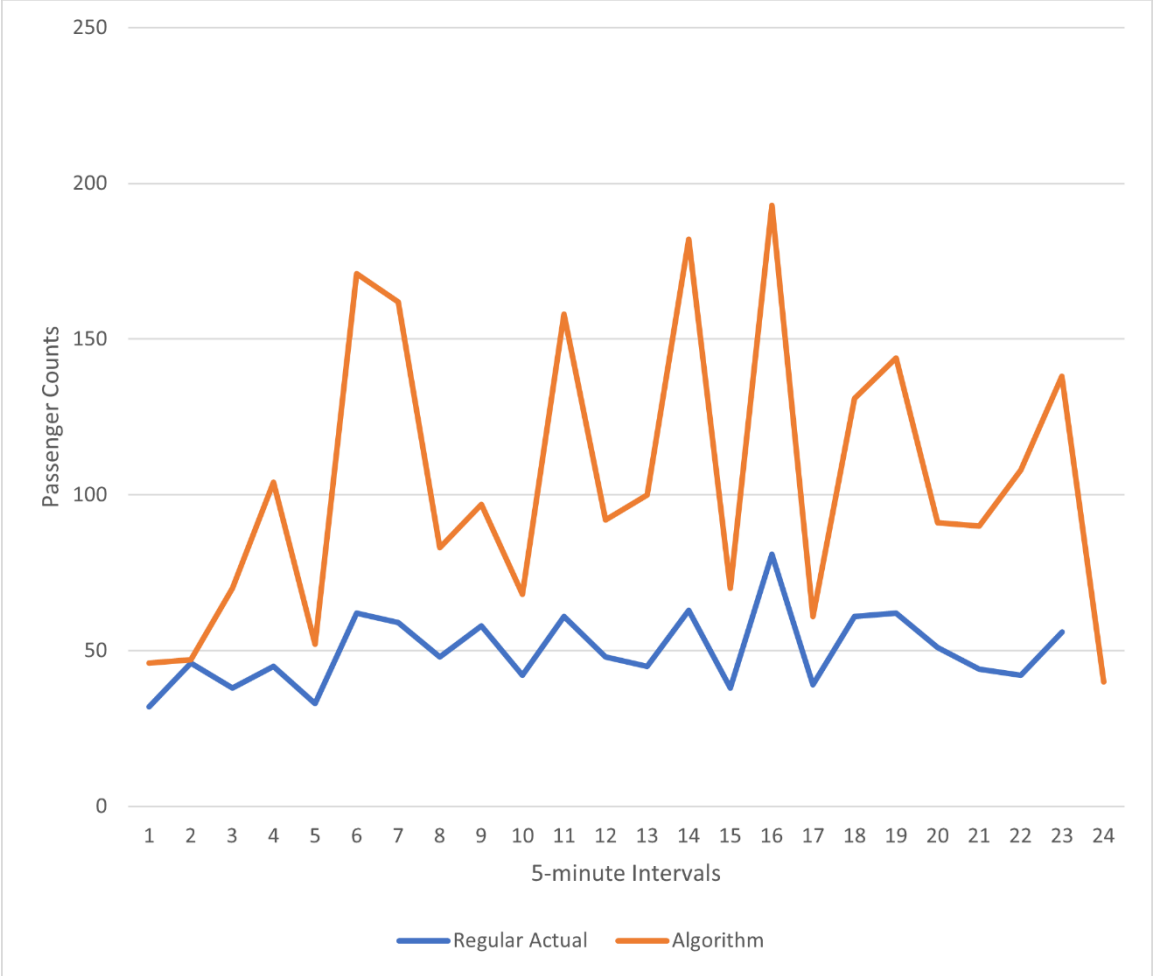

Figure 4. Adding Raw Headcount Augmentation.

The simplest fix to the huge overcount is to make sure the augmentation is not allowed to be higher than the actual count. This fix is shown in Figure 5.

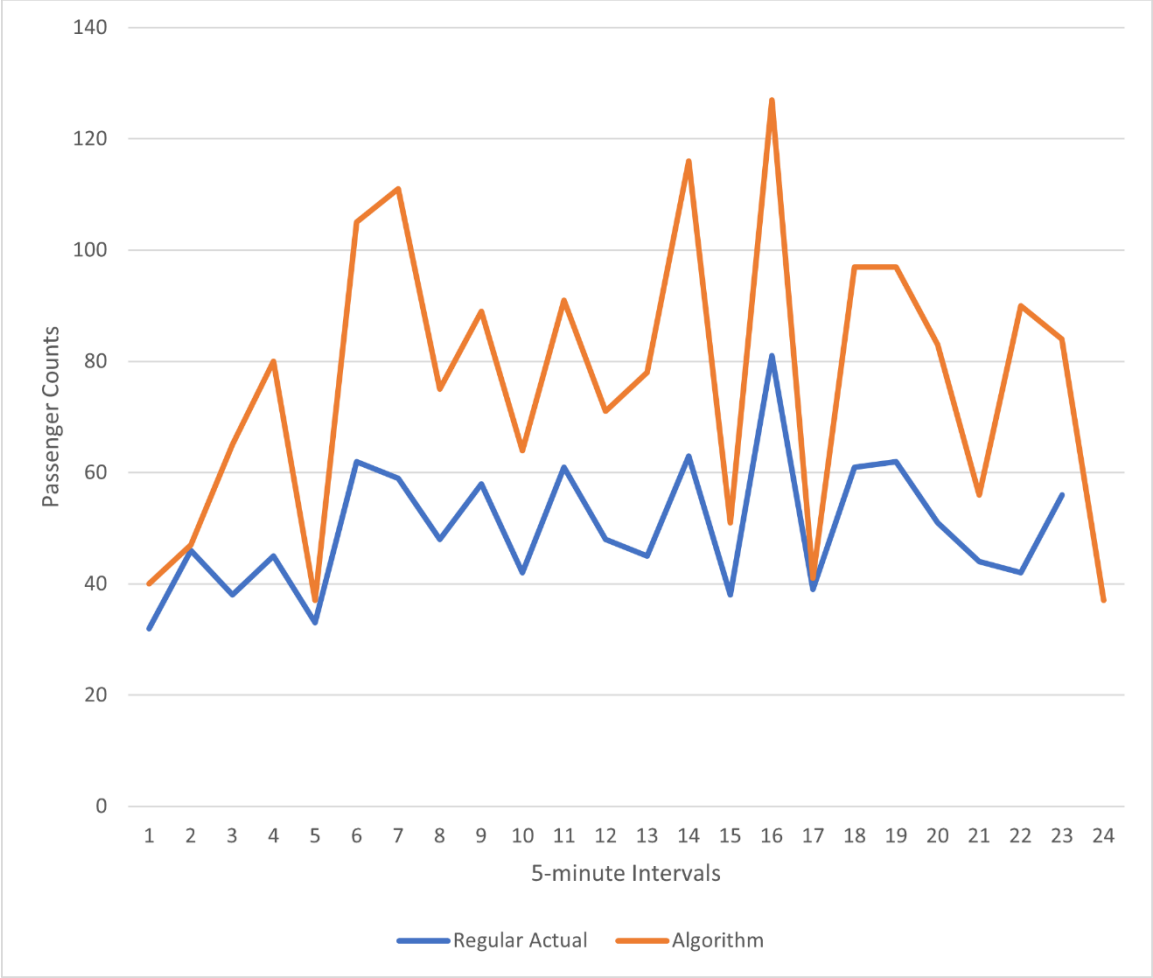

Figure 5. Adding Headcount Augmentation Maxed at Count.

Cleary, the algorithm still always overcounts. This shows that the headcount augmentation is too much. One simple fix is to add only a fraction of the augmentation; adding ½ seems like a simple fix, which is shown in Figure 6.

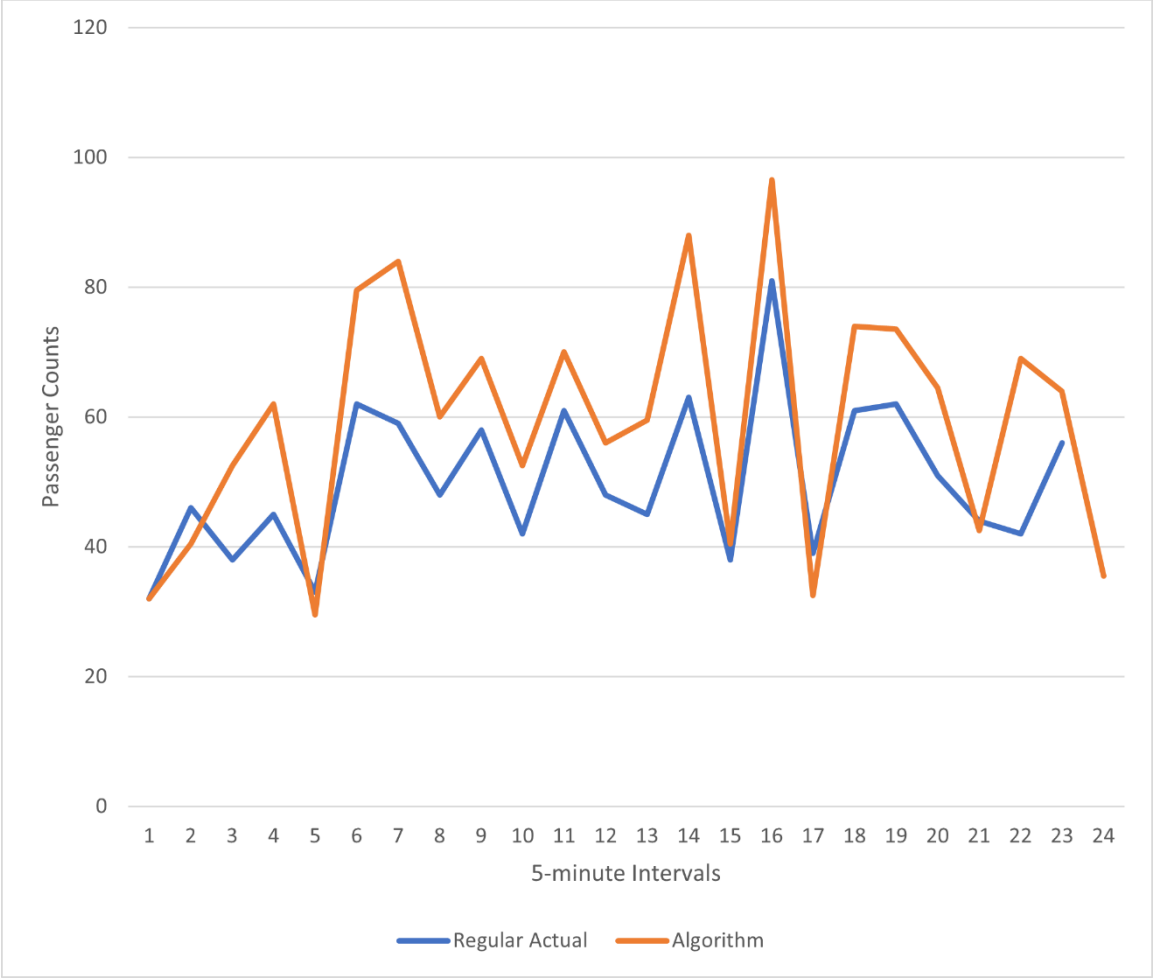

Figure 6. Adding Headcount Augmentation Maxed at Count and Divided by 2.

As can be seen in Figure 6, the low points of the graph are now mostly on track; sometimes the algorithm counts a little more than the correct value, and sometimes a little less. In the medium range, the counts are still too high, and in the high range, the counts are even higher. Looking back to Figure 3, the algorithm count is always low, but the high values are pretty close to the correct count. This led to the idea of using the crowd count to determine how much of an augmentation to use; if it's not crowded, use ½ of the augmentation, as demonstrated in Figure 6; if it's really crowded, don't use any augmentation at all; and if it's medium-crowded, use a different metric. After a couple more such tests, the correct metrics for crowd sizes were determined: if the maximum

headcount in one frame is less than 6, it's not crowded; if it's greater than 12, it's very crowded. After several attempts to find a good fraction that was between 0 and ½, it became clear that no one number would provide accurate results in the medium-crowded scenarios. Thus, the idea of using the crowd size to weight the headcount augmentation arose, as shown in Code Snippet 3 . At first, identical weights were used across the 5 crowd size metrics (between 7 heads and 11 heads, inclusive). Figure 7 shows the result.

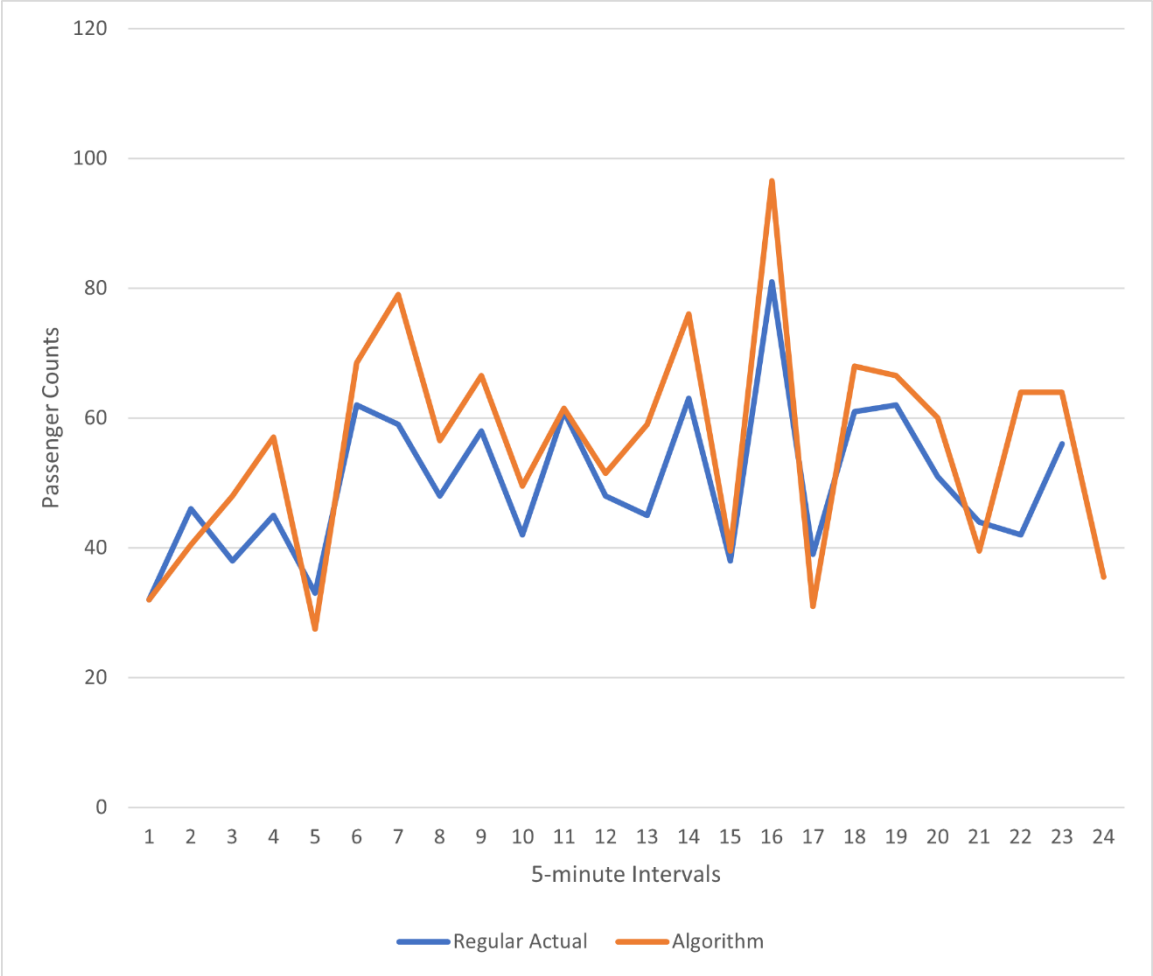

Figure 7. Headcount Augmentation with Equal Weights.

In the figure, it appears that the high points are still spiking, which lead to an attempt at using weights that decreased as the crowd size increased. This is shown in Figure 8.

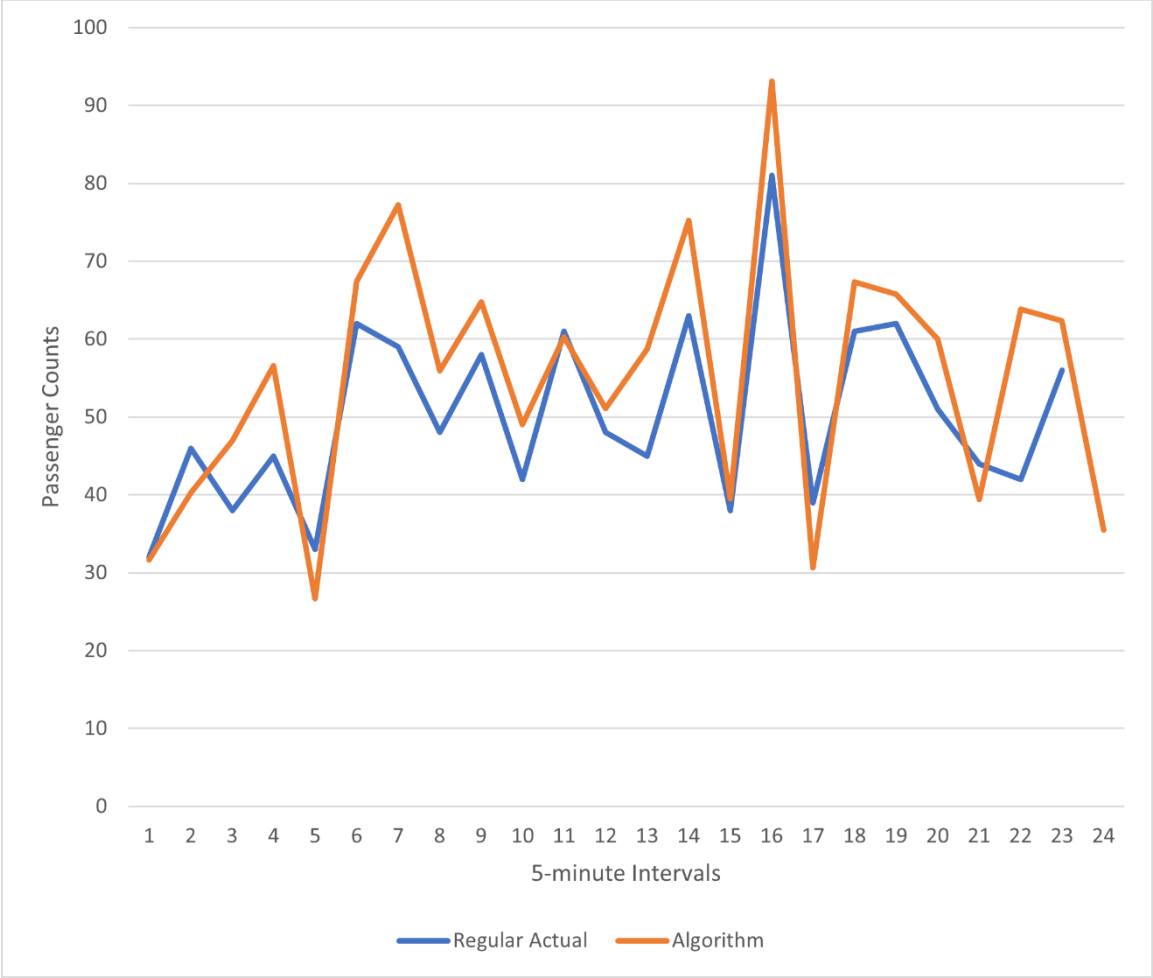

Figure 8. Headcount Augmentation with Decreasing Weights.

If anything, putting the weights in decreasing order made things worse, since the spikes on the high points are even more pronounced. Naturally, the increasing order of weights was used next. This is shown in Figure 9.

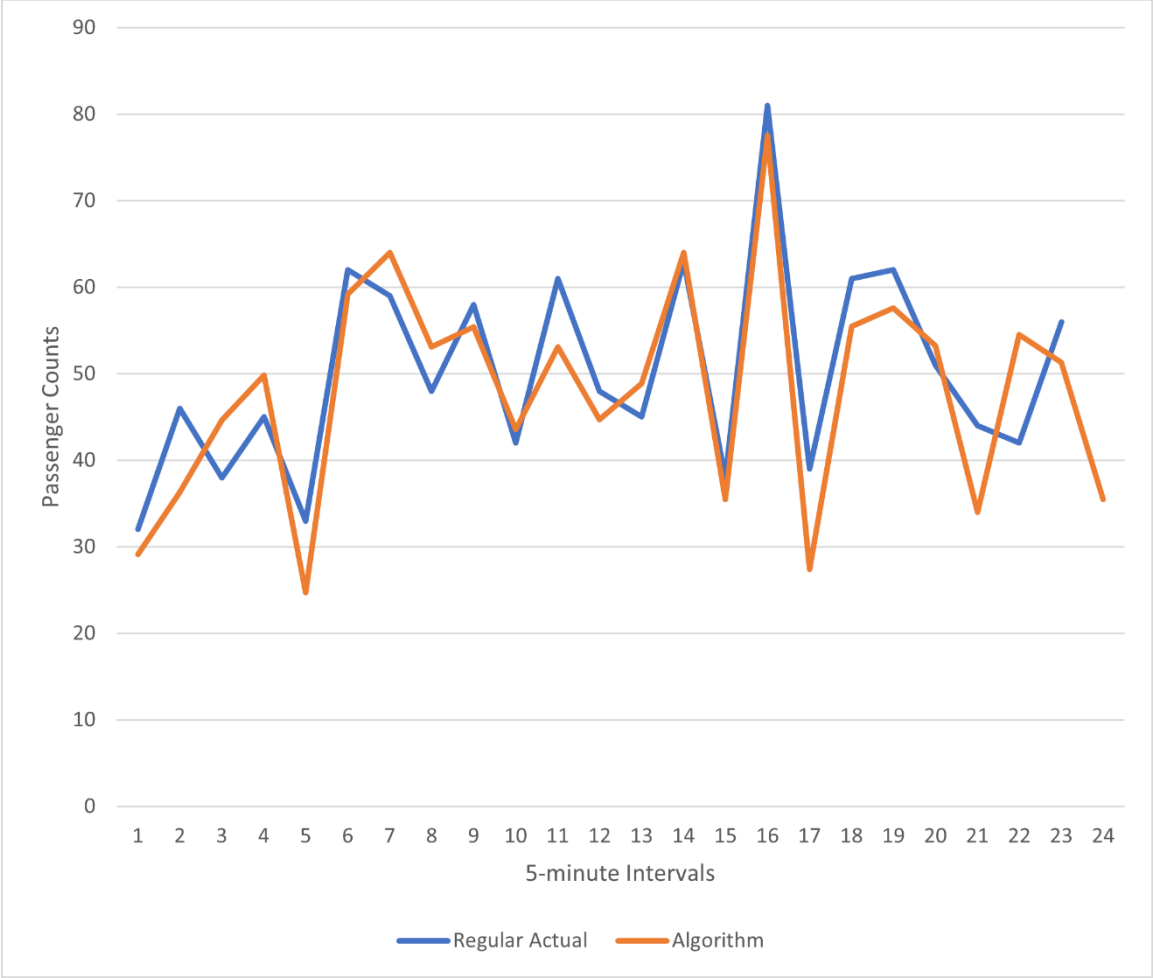

Figure 9. Headcount Augmentation with Increasing Weights.

As can be seen, this is still not perfect. However, it is more or less centered around the actual count, which means that in the long run, the error will be very small. Note that in testing for good ways to use the augmentation and creating the graphs shown in Figures 3 – 9, the actual data points were used to measure total error, while the graphs were used to visualize where the problem causing the error may lie. In the case of Figure 9, the testing showed that the algorithm miscounted by 2 people when considering the entire video. Such a low error demonstrated that the weights used in Figure 9 were best. This was confirmed when testing the other videos; the final results and comparisons are shown in Chapter 3.

2.3 Algorithm Details

This section describes the proposed algorithm in detail, with the goal not of justifying it but of explaining it. This includes explaining the functioning of the YOLO and SORT algorithms.

The proposed algorithm begins by processing the video frames by passing them through the darknet library, which is an implementation of the YOLO algorithm. The YOLO algorithm is built using Convolutional Neural Networks (CNNs). CNNs are a type of neural network that is commonly used for computer vision and image processing. As explained by O'Shea *et al.* (2015), this is because the design of CNNs is partially based on real vision; as explained by Albawi *et al.* (2017), the primary difference between CNNs and fully connected neural networks is that in CNNs, every neuron in a hidden layer only gets inputs from a subset of the neurons in the previous layer, which ultimately gets it inputs from a small portion of the image. Thus, the image is essentially processed in pieces, although simultaneously and with continuous overlap of all parts of the image. CNNs also typically have one or several fully connected layers. Every layer, whether fully connected or not, has parameters that must be trained in the training phase of the model; the fully connected layers typically have the most parameters because they have the most links between neurons.

The frame is first resized to fit the CNN, and then the frame (specifically, its pixels) is processed through the network. As explained in the YOLO paper (Redmon *et al.,* 2016), the convolutional layers of the CNN are responsible for extracting features from the frame while the fully connected layers produce the bounding box coordinates

and the probability of correct detection. One major advantage of YOLO as opposed to other detectors is that it produces all of the outputs in one step, which greatly reduces the complexity of the pipeline and increases the speed at which detections can be performed.

Once a list of detections has been returned from the darknet library, the proposed algorithm loops through the detections of heads to determine the crowdedness in front of each lane. It then updates the crowd-size array, which is used later in weighting the final count. This process is shown in Code Snippet 2 in Section 2.2.

Then, the proposed algorithm loops through the detections of people and updates the SORT tracker with those detections. The basic function of the tracker is to match detections from one frame to the next (Bewley *et al.*, 2017). Unlike many other trackers, the SORT tracker does not use the internal data of a detection, such as facial features or clothing of a person, to match detections; instead, it uses only the bounding box position and size to track it through frames. As a result, the SORT tracker is not able to re-identify an object after it has lost track of it because of occlusion. The reasons for such simplicity is speed; as argued by the authors of SORT, introducing such complexities into the tracking framework would potentially limit its use in real time applications. While the inability to reidentify an object could prove troublesome for a computer vision algorithm, the proposed algorithm has a simple way of dealing with that, which is explained next.

The SORT tracker returns the coordinates of the bounding boxes (which are the same as the bounding boxes returned from darknet), but it also returns an ID number with each bounding box. The ID number represents which tracked object the bounding box is. The proposed algorithm keeps track of several lists, one of IDs that are in the precheck lane, one of IDs that are in the regular boarding lane, and one of IDs that are not in any

lane but are in the open area in front (see Figure 1, Section 1.2). If an ID was in the no-lane list, but the bounding box associated with that ID is now inside of a lane, the ID is reassigned to the correct list and the count of that lane is increased. This corresponds to a person entering the lane. Due to this method of counting, the tracking only needs to be accurate on the frame before and after a person has entered a security lane. Although occlusion occurs frequently in this project environment, it is unlikely for occlusion to occur during those two specific frames out of the hundreds of frames in which the person is begin tracked. Thus, the problem with SORT not recovering from occlusions is mitigated.

If a person is detected as crossing a lane boundary, the proposed algorithm loops through the detected heads to determine if there are more than one head detected inside that person's bounding box. If yes, the headcount augmentation is increased. This process is shown in Code Snippet 1 in Section 2.2.

Once the SORT tracker and the lists of IDs in each lane have been updated, the next frame is processed by darknet and the headcount augmentation, the crowd-size array and the SORT tracker are updated. Every 30 seconds (1800 frames at 60fps), the weighted headcount augmentation is added. The number of IDs in each list represents the number of people actually counted entering the lane. This value is increased by the headcount augmentation weighted by the crowd-size array. This process is shown in Code Snippet 3 in Section 2.2. After the summing and weighting is done and the value for that 30-second interval is saved, the headcount augmentation and crowd-size array are set to 0. The SORT tracker is not reset, because it needs to keep tracking people from the previous 30 seconds.

Every 5 minutes, the values from the last 10 30-second intervals are added up and returned as the output for that 5-minute interval. This concludes the detailed description of the proposed algorithm.

To help explain the algorithm, an example will be provided. This example is about a 10-minute video. Given the huge number of frames involved, the numbers counted in each frame cannot be shown; nevertheless, this example should help the reader to understand the algorithm.

The first frame of the video is processed. It is passed through darknet and SORT. Since this is the first frame, no people have been tracked yet. Since the headcount augmentation is calculated only when a person is tracked into a lane, the headcount augmentation is also 0. Suppose that there were 4 heads detected near the entrance to the regular lane. The 4^{th} element of the crowd size array is incremented from 0 to 1 (Code Snippet 2). Suppose further that one of the tracked people, with ID 1, is in the open area (not in any lane).

The next frame of the video is processed. Suppose that the person with ID 1 from the first frame is now inside of the regular lane. ID 1 is added to the regular lane list. Since the transition occurred during this frame, the algorithm checks to see how many heads are detected inside of person 1's bounding box. Suppose only 1 head is detected; the headcount augmentation will still be 0 (Code Snippet 1). Suppose this time there are 3 heads detected near the entrance to the regular lane, so the 3^{rd} element of the crowd size array is incremented to 1.

The third frame of the video is processed. Suppose that this time another person, with ID 2, is tracked into the regular lane. Again, since the transition occurred during this

frame, the algorithm checks to see how many heads are detected inside of person 2's bounding box. Suppose there are 3 heads detected. The headcount augmentation is incremented to 2 (because one head is expected to appear, only the extra heads are added to the augmentation). The crowd-size array is updated depending on the number of heads that are near the regular lane.

This continues for another 1797 frames (1800 frames total, or 30 seconds at 60fps). At the end of that time, suppose there are 5 IDs in the regular lane list (note that the actual ID number is irrelevant). Suppose further the headcount augmentation is 6. The crowd-size array is shown in the Table 1.

Table 1. Example of Crowd-size Array

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Value	933	128	264	305	170	0	0	0	0	0	0	0	0

The crowd-size array can be explained as follows: There were 933 frames in which 0 people were in the vicinity of the regular lane, 128 frames in which 1 person was in the vicinity of the regular lane, etc., and there were 170 frames in which 4 people were in the vicinity of the regular lane. In the last 30 seconds, the regular lane never got more crowded than 4 people.

The algorithm proceeds to run Code Snippet 3. The algorithm first checks if the augmentation is greater than the number of people counted entering the regular lane. Since there are 5 IDs (aka people) in the regular lane list, the algorithm checks if $6 > 5$. Since the augmentation cannot be greater than the actual number of people counted, the augmentation is limited to 5. As seen in the crowd-size array, the crowd size never

exceeded 4 people; thus, the “else” statement is executed in which the augmentation (5) is multiplied by 0.5. The final augmentation is 5*0.5, or 2.5. This number is implicitly truncated to 2 using integer conversion because counts of people cannot be fractional. The augmentation, 2, is then added to the actual count from the regular lane list, which is 5. Thus, the total count for the last 30-second interval is 5+2, or 7. This number is stored for future use.

All of the variables are then set to 0; namely, the headcount augmentation is set to 0, the elements of the crowd-size array are set to 0, and the regular lane list is emptied. Then, the algorithm processes the next 1800 frames, performing the exact same steps and math. After processing 10 1800-frame chunks, which is equivalent to 5 minutes of video at 60fps, the algorithm sums the numbers from those 10 chunks. Suppose, in this example, that the numbers were 7,5,9,2,3,9,8,3,1 and 0. The sum is 47. This sum is returned as the count for that 5-minute video segment.

The algorithm then proceeds to run the entire process over for the next 5-minute segment.

2.4 Unwanted Camera and Lane Movement

One issue that escaped notice at first but became clear during detailed testing is that the security camera is adjustable, and it occasionally moves. There are times when the camera moves to a completely different area and even zooms in; at such times it is, of course, impossible to count people entering the security lanes, and the algorithm does not employ any means to detect such unwanted movement.

There are also other times when the camera moves only slightly, so that the lanes are still visible but are no longer aligned as they were before. Another problem can also arise: the entrances into the security lanes are marked by line dividers, and they are sometimes moved by the security officers or staff. This also causes the lanes to be aligned differently. A visual example of this is demonstrated in Figures 10 and 11. The images both have black lines drawn to show where the lane coordinates are. In Figure 11 the lane coordinates are clearly off from what they are in Figure 10, because the line dividers were moved. A red circle was added to pinpoint the problem.

The precheck lane is on the left; the clear lane is directly to right of that; and the regular boarding lane is on the right. As can be seen, the precheck and clear lanes are right next to each other, which means that if the line dividers between those lanes are moved even slightly it can have a large impact on the error for those lane counts. Moving the lane dividers near the regular boarding lane does not have as much of a detrimental impact, because people are not likely to be in the vicinity of the regular boarding lane unless they are entering it. However, the algorithm can easily confuse people between the precheck and clear lanes if those lane dividers are moved.

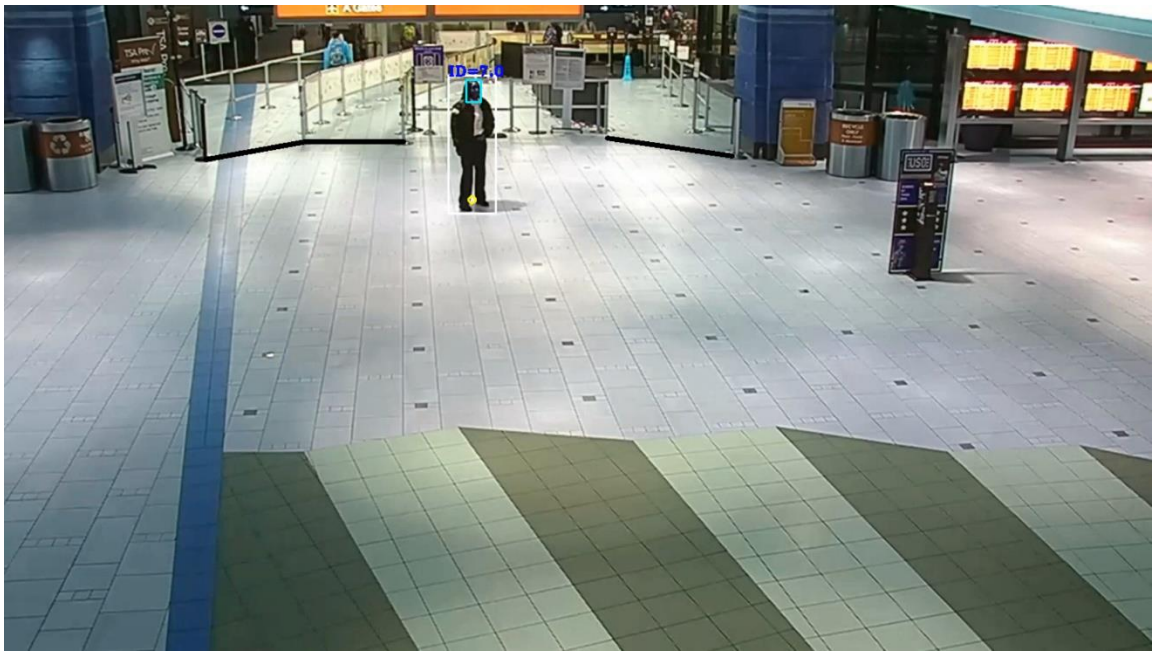

Figure 10. Correct Line Orientation

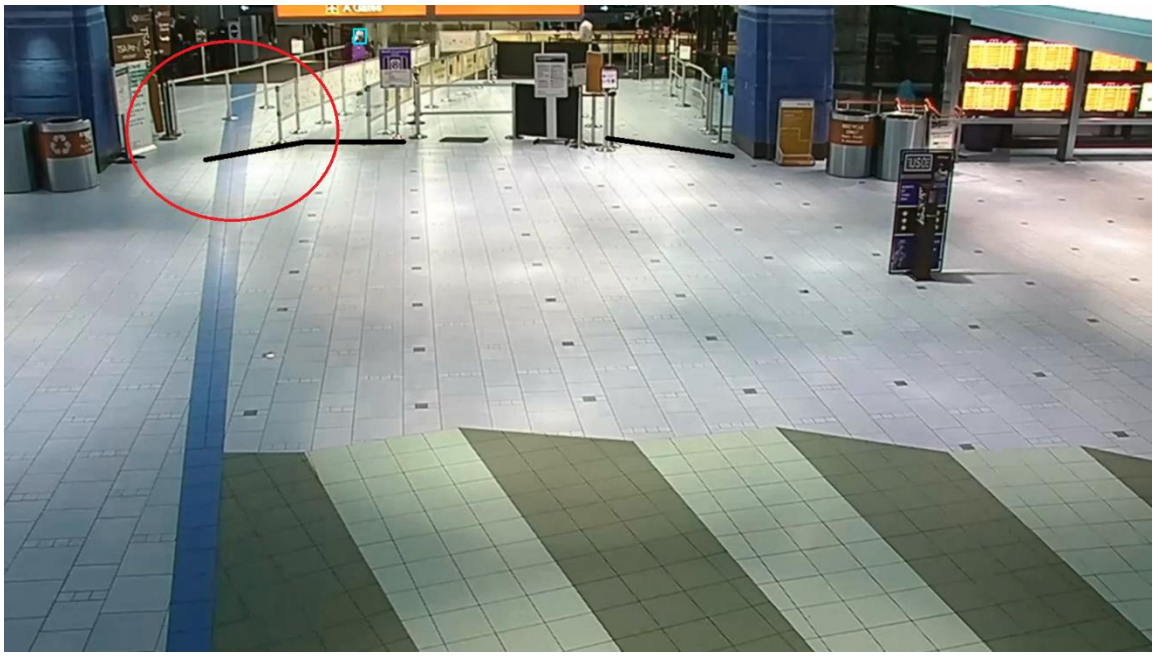

Figure 11. Incorrect Line Orientation

As discussed in Chapter 3, five videos were tested thoroughly, and to minimize the potential error caused by unwanted lane alignment changes, different lane coordinates were manually found and used for each video. However, the line dividers could have still been moved during the videos; this is not accounted for by the current algorithm. In theory, this could be fixed using background subtraction methods; there are many powerful frameworks that could tackle this task (e.g., see Bouwmans *et al.,* 2019); this is discussed more in the conclusion of this thesis.

Chapter 3: Results and Comparisons

This chapter provides comparisons between the baseline algorithm and the improved algorithm. Comparisons are also provided to test the benefits of introducing the CrowdHuman dataset, but before adding the headcount procedure. This essentially corresponds to a half-way improvement – the new training set is used, but the headcount is not used.

This algorithm was tested on 5 videos, each of which is 2 hours long. Counts are provided by the algorithm in 5-minute segments, for a total of 24 data points per video. The algorithm (all versions, baseline, CrowdHuman, and proposed) is tested against counts that were manually counted for each video, also in 5-minute intervals. For four of the five videos, data is available from the baseline algorithm; for all five videos, data is available for both the CrowdHuman dataset alone as well as the CrowdHuman with headcount (which is labeled “Final” in the graphs that follow).

3.1 Comparison of Results

The best result from the testing is seen on the regular boarding lane. Section 2.3 explains the most likely reason why the precheck lane has higher errors. Data for the clear lane is not used or shown, because it is extremely unreliable. The most likely cause of this is that Clear staff members often pace back and forth directly in front of the Clear lane to greet members and advertise the benefits of the Clear pass. This drastically increases the algorithm’s error. Figures 12 – 16 display the actual count and results from

the baseline algorithm, the CrowdHuman dataset change, and the proposed algorithm. The data are from the regular boarding lane; data from the precheck lane are shown later. Tables 2 – 6 report the total error and average absolute error per 5-minute interval of the corresponding counts. The formulas used to calculate these errors are from Excel and in order to show them here, they have been modified not be specific to any cells. Whenever "Counts" is used, a range of cells is implied; when "Count" is used, an individual cell is implied.

$$\text{Total Error} = \text{SUM}(\text{Manual Counts}) - \text{SUM}(\text{Algorithm Counts})$$

$$\text{Total Error\%} = \frac{\text{SUM}(\text{Manual Counts}) - \text{SUM}(\text{Algorithm Counts})}{\text{SUM}(\text{Manual Counts})} \times 100\%$$

$$\text{Average ABS Error} = \text{AVERAGE}(\text{ABS}(\text{Manual Count} - \text{Algorithm Count}))$$

$$\text{Average ABS Error\%} = \text{AVERAGE}(\frac{\text{ABS}(\text{Manual Count} - \text{Algorithm Count})}{\text{Manual Count}} \times 100\%)$$

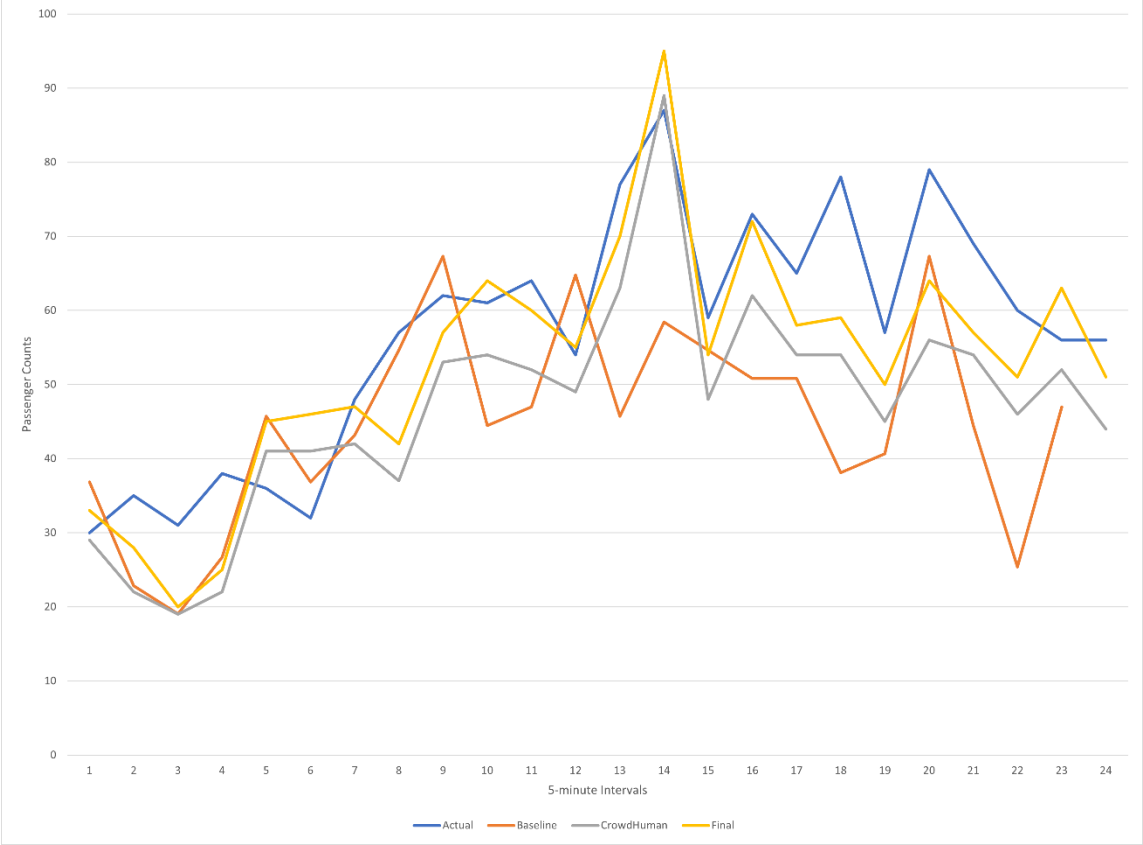

Figure 12. Count Comparison from Regular Lane, March 15, 2019 AM.

Table 2. Data for Regular Lane, March 15, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	275	21.06%	15.26	26.15%
CrowdHuman	236	17.30%	11.17	20.44%
Final	98	7.18%	7.83	15.26%

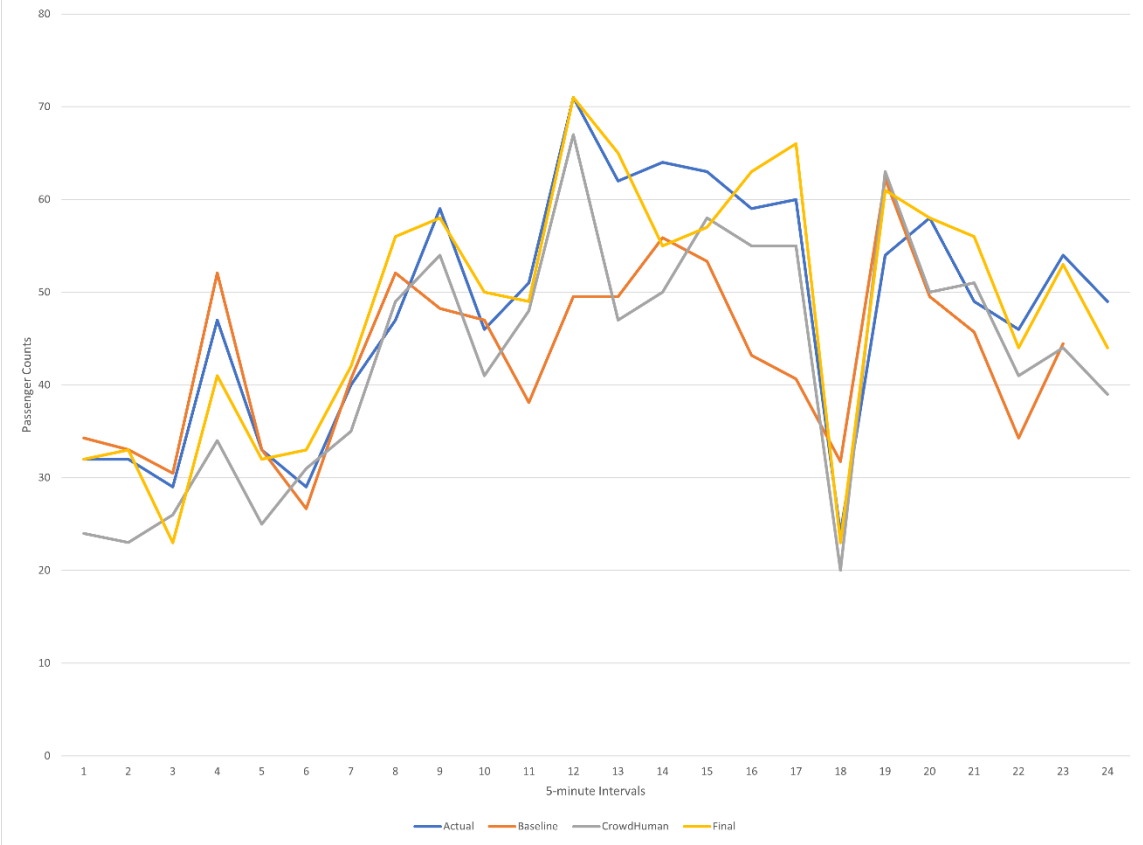

Figure 13. Count Comparison from Regular Lane, March 19, 2019 AM.

Table 3. Data for Regular Lane, March 19, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	113	10.21%	7.76	14.86%
CrowdHuman	128	11.05%	6.58	14.17%
Final	-7	-0.60%	3.63	7.70%

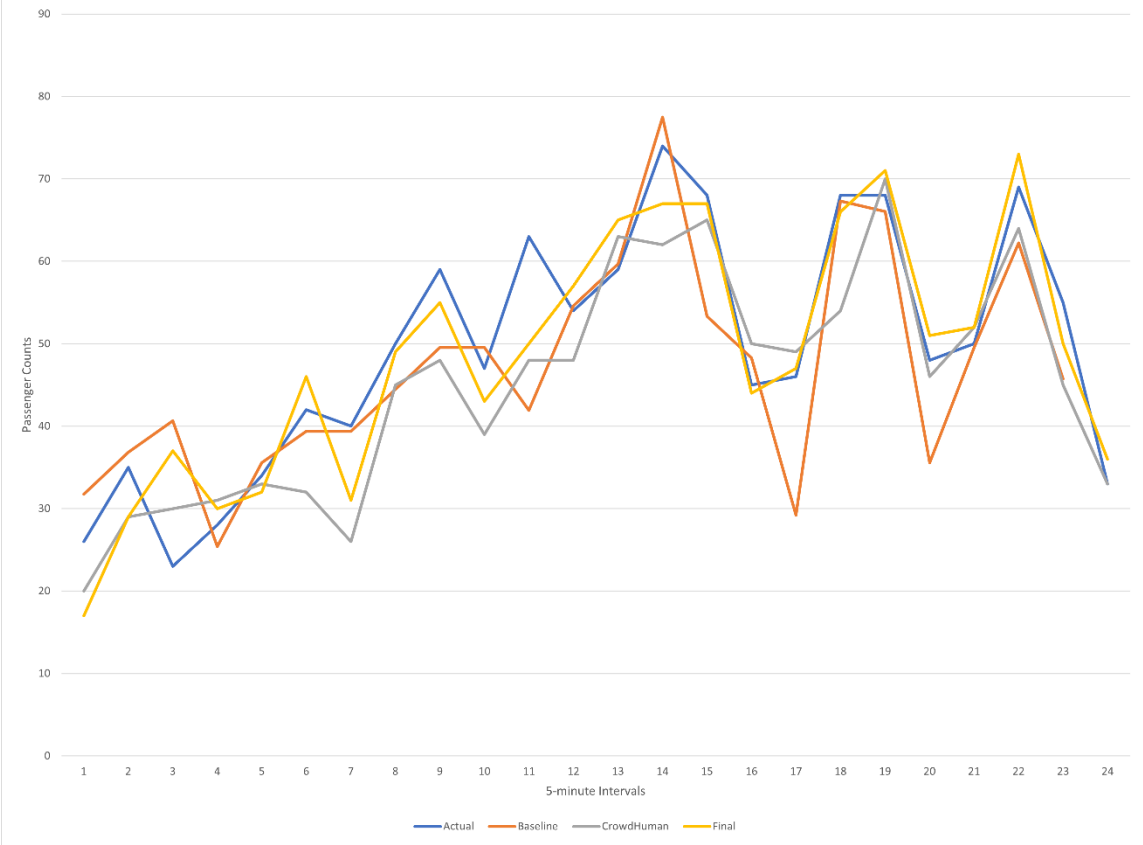

Figure 14. Count Comparison from Regular Lane, March 21, 2019 AM.

Table 4. Data for Regular Lane, March 21, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	67	5.88%	6.19	13.98%
CrowdHuman	102	8.61%	6.42	13.58%
Final	19	1.60%	4.54	11.18%

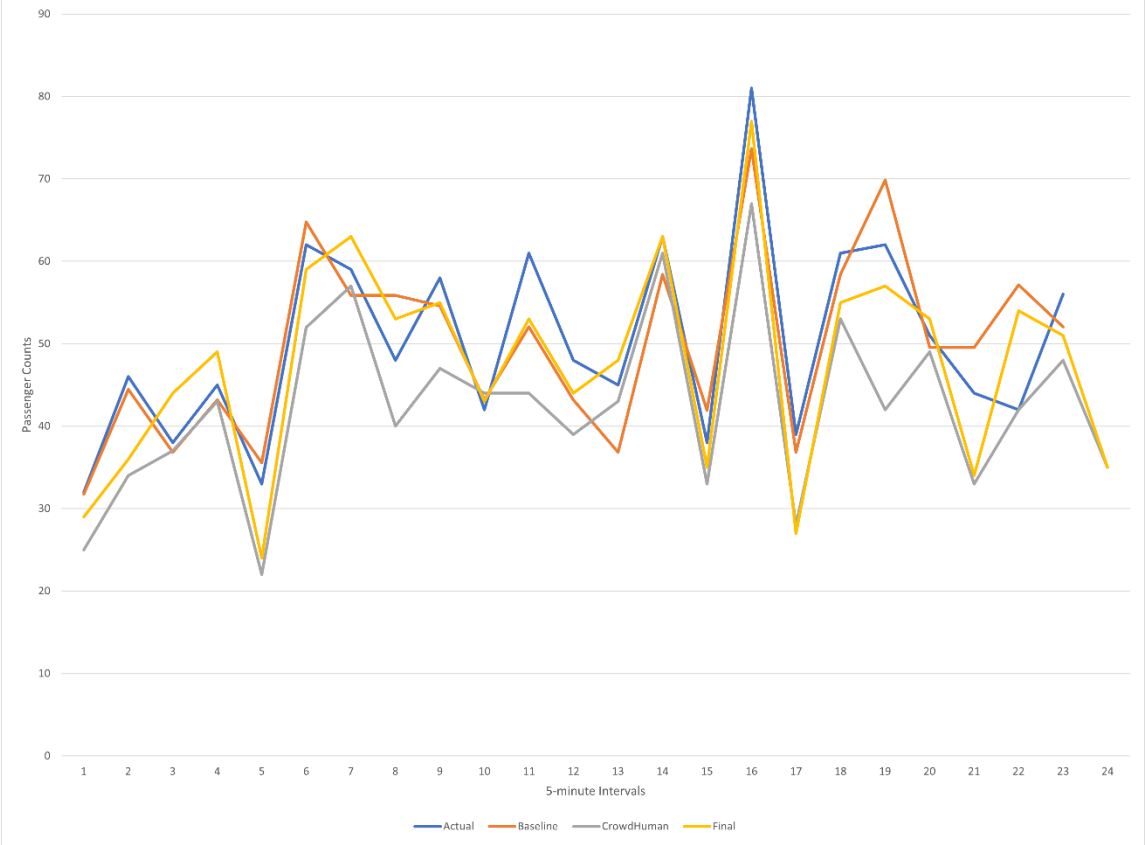

Figure 15. Count Comparison from Regular Lane, March 22, 2019 AM.

Table 5. Data for Regular Lane, March 22, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	8	0.73%	4.44	8.88%
CrowdHuman	171	14.82%	7.61	15.21%
Final	48	4.16%	5.30	11.58%

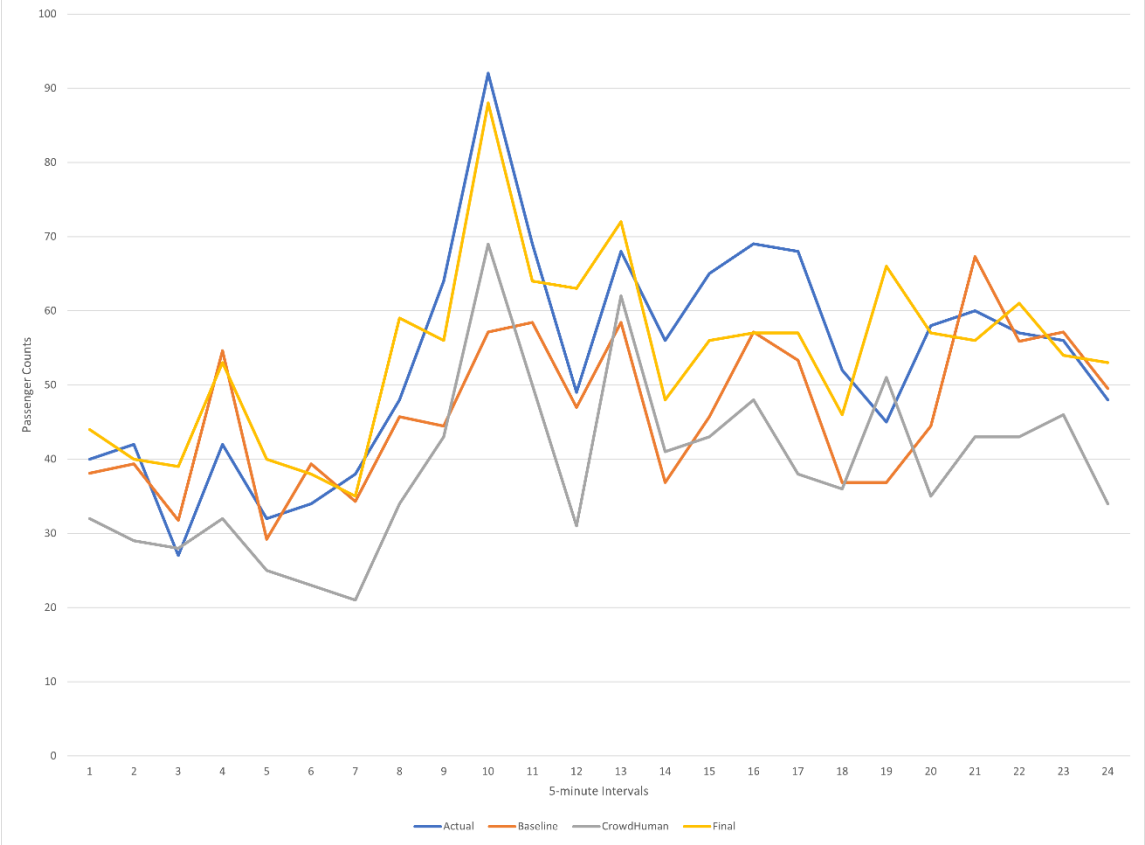

Figure 16. Count Comparison from Regular Lane, March 30, 2019 AM.

Table 6. Data for Regular Lane, March 30, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	161	13.13%	9.74	16.92%
CrowdHuman	342	26.74%	14.83	27.35%
Final	-23	-1.80%	7.21	15.03%

Figures 17 – 21 and Tables 7 – 11 report the algorithm performance for the precheck lane. Note that in some of the videos the CrowdHuman algorithm performs better than the final algorithm (in other words, the headcount augmentation makes the algorithm worse). This is most likely due to factors explained in Section 2.3.

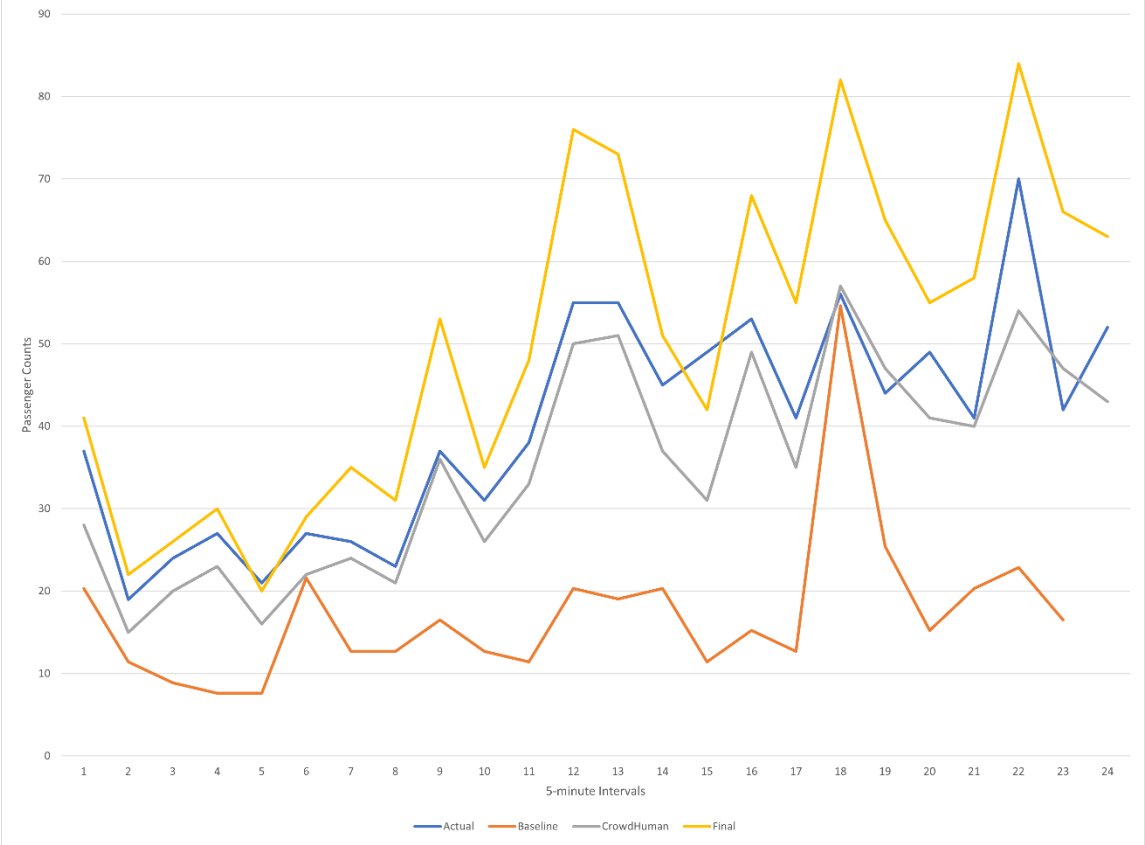

Figure 17. Count Comparison from Precheck Lane, March 15, 2019 AM.

Table 7. Data for Precheck Lane, March 15, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	512	56.32%	22.28	55.48%
CrowdHuman	116	12.06%	5.58	14.17%
Final	-246	-25.57%	10.92	25.72%

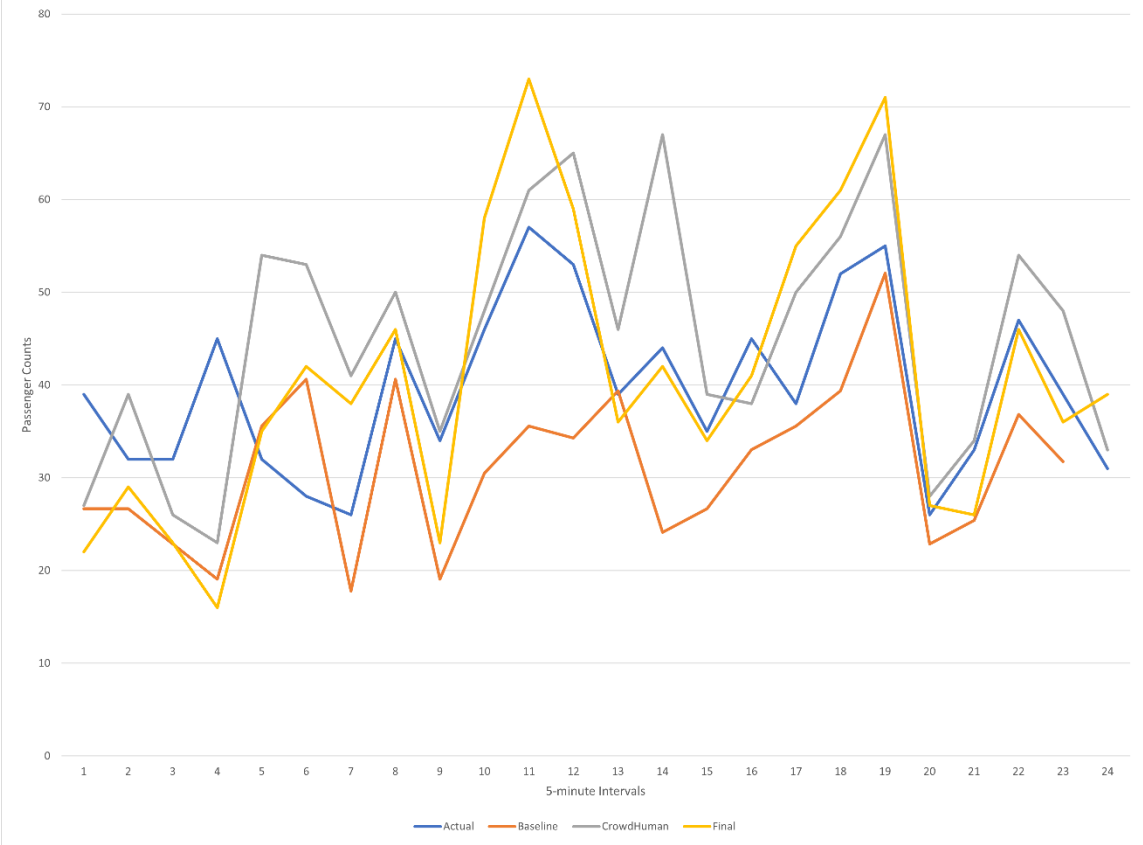

Figure 18. Count Comparison from Precheck Lane, March 19, 2019 AM.

Table 8. Data for Precheck Lane, March 19, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	205	22.31%	10.39	25.68%
CrowdHuman	-129	-13.54%	9.29	24.90%
Final	-25	-2.62%	8.54	21.96%

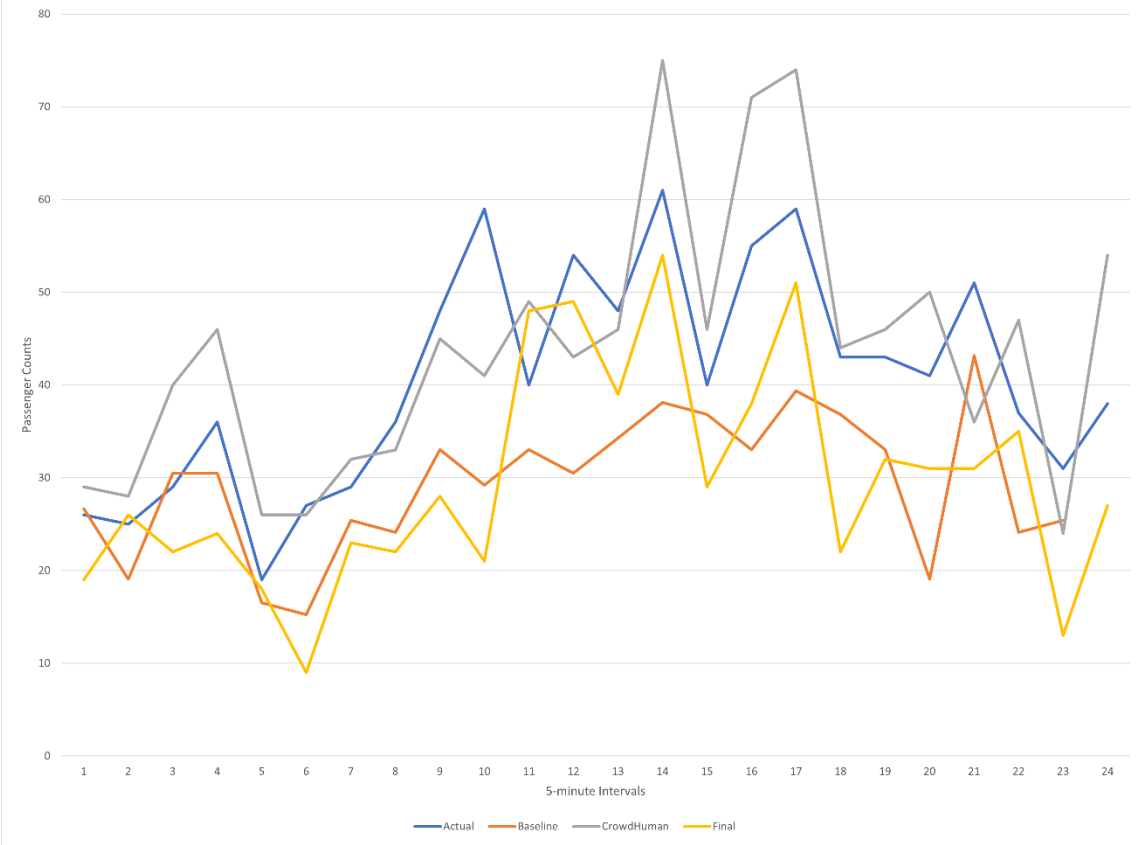

Figure 19. Count Comparison from Precheck Lane, March 21, 2019 AM.

Table 9. Data for Precheck Lane, March 21, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	260	27.76%	11.50	26.00%
CrowdHuman	-76	-7.80%	8.17	19.88%
Final	264	27.08%	11.75	28.66%

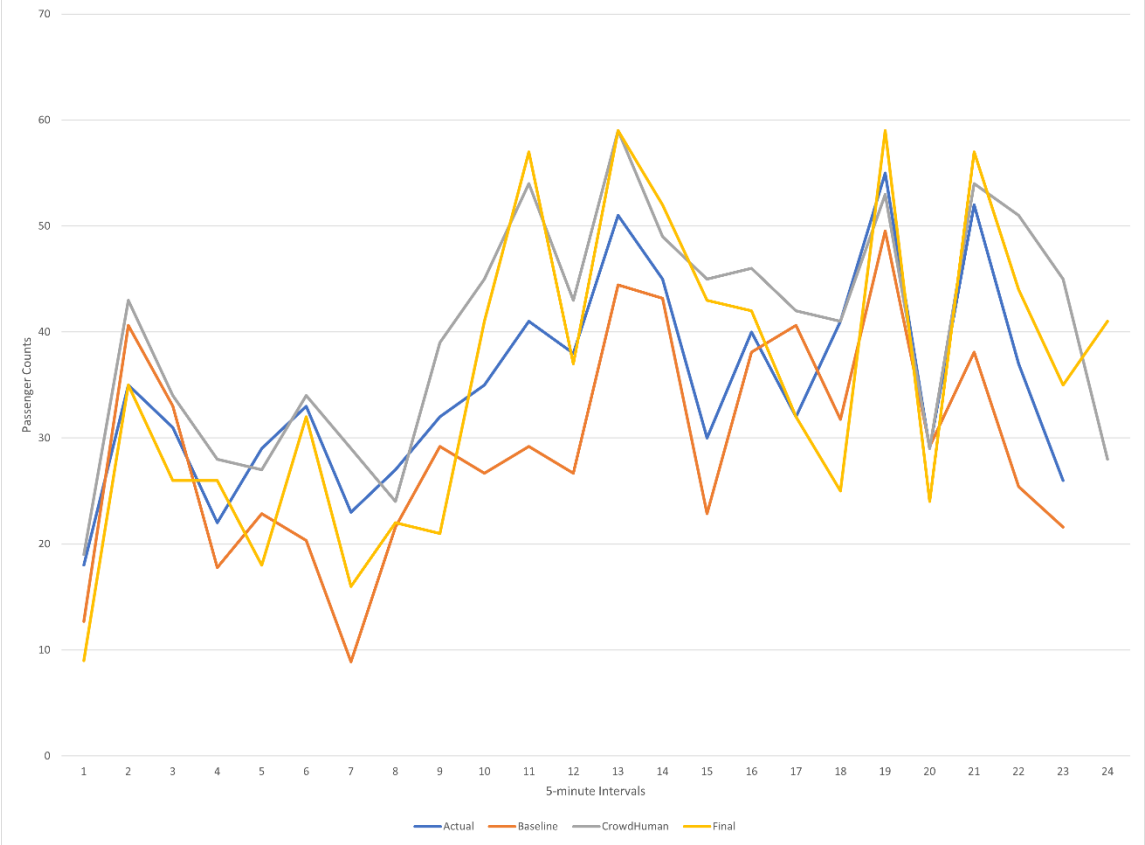

Figure 20. Count Comparison from Precheck Lane, March 22, 2019 AM.

Table 10. Data for Precheck Lane, March 22, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	127	15.91%	6.98	21.05%
CrowdHuman	-131	-16.33%	6.30	19.44%
Final	-10	-1.25%	6.61	20.59%

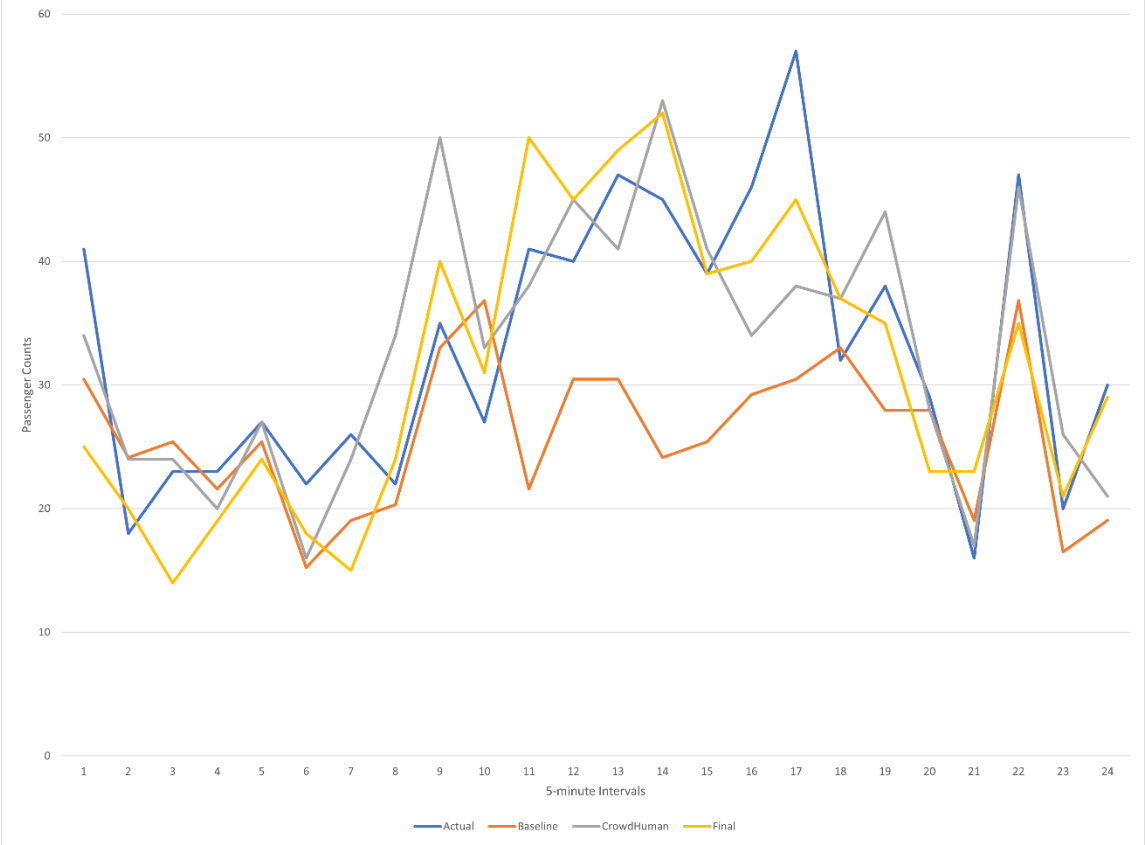

Figure 21. Count Comparison from Precheck Lane, March 30, 2019 AM.

Table 11. Data for Precheck Lane, March 30, 2019 AM.

	Total Error	Total Error %	Average ABS Error per 5-min Interval	Average ABS Error per 5-min Interval %
Baseline	156	20.56%	8.75	23.97%
CrowdHuman	-4	-0.51%	5.92	18.36%
Final	38	4.80%	5.67	17.78%

As demonstrated in the figures and tables above, the final algorithm, which incorporates both the CrowdHuman dataset and the headcount augmentation, almost always outperforms the others, especially in the regular boarding lane; only once (in the March 22 AM video) was the baseline algorithm better than the final algorithm. The

results from the precheck lane are not as great for the final algorithm; they suggest that the CrowdHuman and final algorithm are about equally as accurate, since 2 of the videos are counted better by the final algorithm, and 3 are counted better by the CrowdHuman algorithm. Note that in the precheck lane, they both always outperform the baseline algorithm. Also note that what is referred to as the "CrowdHuman" algorithm is the same as the final algorithm without headcount augmentations and is also the same as the baseline algorithm with a different training dataset.

Table 12 provides a summary of the differences between the 3 algorithms, including both the algorithm details and their results. The "Average Absolute Total Error" takes the average of the absolute values of the Total Error from Tables 2-5 and 10 for the Regular Lane and Tables 6-9 and 11 for the Precheck Lane. The "Average of Average Error" takes the average of the Average ABS Error from the same tables.

Table 12. Summary of Algorithm Differences.

	Baseline	CrowdHuman	Final
Dataset Used	COCO	CrowdHuman	CrowdHuman
Classes Used	Person	Person	Person, Head
Uses Headcount and Crowd Size?	No	No	Yes
Average Absolute Total Error in Regular Lane	10.20%	15.70%	3.07%
Average Absolute Total Error in Precheck Lane	28.57%	10.05%	12.26%
Average of Average Error per 5-minute Interval in Regular Lane	16.16%	18.15%	12.15%
Average of Average Error per 5-minute Interval in Precheck Lane	30.44%	19.35%	22.94%

Chapter 4: Conclusion

Computer vision algorithms often rely on perfect input to produce the right output. When it comes to blurry, jumpy, or occluded videos, however, state-of-the-art algorithms can have a high error. It is important to note that for fixing such problems, it is likely that no one solution will work for all blurry or choppy videos. Sometimes, it is feasible to filter the videos to create a sharper image and smoother video which will result in more accurate detections and tracking. An alternative solution is to use a different training dataset. Yet a third option is to utilize other objects, in the case of this project, heads, to somehow modify the algorithm to count more accurately. The exact modification will depend on the problem at hand. In the case of this algorithm, using the number of heads to determine how crowded a setting is, and also using the number of heads to estimate how many people were missed while counting, proved to be a good solution for the regular boarding lane. For the precheck lane, this solution does not work as consistently, although it still improves a lot on the baseline algorithm. This is most likely due to the fact that the precheck and clear lanes are right next to each other, and whenever a TSA officer or airport staff moves the line dividers, that causes the algorithm to think people are in a different lane than they really are.

There are several things that could be improved in this algorithm. One way to mitigate the error associated with a moving camera is to use background subtraction to update the lane coordinates whenever the camera moves. Background subtraction can also be used to identify and adjust whenever line dividers are moved. This will especially help mitigate the error in the precheck lane. Another possible improvement to this

algorithm would be to test the OpenCV tracking algorithms, as well as other state-of-the-art trackers, against the SORT algorithm used in this project. While the count from the Clear lane does not contribute highly to the total number of passengers using the checkpoint, a third improvement could focus on improving the code that counts people in the Clear lane. This improvement would involve improving the backwards-tracking code (counting whenever a person walks out of a lane), as well as identifying TSA officers or airport who pace back and forth in front of the Clear lane (or any other lane) in order to exclude them from the count.

Finally, an improvement that is not directly related to this algorithm, but could be run in parallel, is to develop an algorithm to count the amount and types of luggage and other large objects, such as strollers, that enter the checkpoint. This could also help TSA improve their staffing and lane availability.

REFERENCES

Albawi, S., Mohammed, T. A. and Al-Azawi, S. "Understanding of a Convolutional Neural Network," *2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.*

Aqqa, M., Mantini, P. and Shah, S. K. "Understanding How Video Quality Affects Object Detection Algorithms." *VISIGRAPP 5, 96-104 (2019).*

Behera, S., Dogra, D. P., Bandyopadhyay, M. K. and Roy, P. P. "Estimation of Linear Motion in Dense Crowd Videos using Langevin Model". *Expert Systems with Applications 150, 2020*.

Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. "Simple Online and Realtime Tracking". *arXiv:1602.00763v2 (2017).*

Bouwmans, T., Javed, S., Sultana, M. and Jung., S. K. "Deep Neural Network Concepts for Background Subtraction: A Systematic Review and Comparative Evaluation". *Neural Networks 117, 8-66 (2019)*. https://doi.org/10.1016/j.neunet.2019.04.024.

Brodley, C. E., and Friedl, M. A. "Identifying mislabeled training data." *Journal of Artificial Intelligence Research 11, 131-167 (1999).*

COCO – Common Objects in Context. https://cocodataset.org/#home (accessed April 7, 2021).

Introduction to OpenCV Tracker. OpenCV. https://docs.opencv.org/3.4/d2/d0a/tutorial_introduction_to_tracker.html (accessed April 5, 2021).

Kim, C. E., Oghaz, M. M. D., Fajtl, J., Argyriou, V. and Remagnino, P. "A Comparison of Embedded Deep Learning Methods for Person Detection". *arXiv:1812.03451v2 (2019).*

Kim, J., Sung, J. and Park, S. "Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition". *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia),* Seoul, Korea (South), 2020, pp. 1-4, doi: 10.1109/ICCE-Asia49877.2020.9277040.

Kim, J. S., Yeom, D. H. and Joo, Y. H. "Fast and Robust Algorithm of Tracking Multiple Moving Objects for Intelligent Video Surveillance Systems," *IEEE Transactions on Consumer Electronics, vol. 57, no. 3, pp. 1165-1170, August 2011, doi: 10.1109/TCE.2011.6018870.*

Kromann, R. “Comparison of YOLO in Darknet versus YOLO in Darkflow”. *LinkedIn (2018).* https://www.linkedin.com/pulse/comparison-yolo-darknet-versus-darkflow-rasmus-kromann/ (accessed April 5, 2021).

Marciniak, T., Chmielewska, A., Weychan, R, Parzych, M. and Dabrowski, A. “Influence of Low Resolution of Images on Reliability of Face Detection and Recognition”. *Multimed Tools Appl 74, 4329-4349 (2015).* https://doi.org/10.1007/s11042-013-1568-8.

O'Shea, K. and Nash, R. “An Introduction to Convolutional Neural Networks”. *arXiv:1511.08458 (2015).*

Pan, J. and Hu, B. "Robust Occlusion Handling in Object Tracking." *2007 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-8. IEEE, 2007.*

Pepik, B., Stark, M., Gehler, P. and Schiele, B. “Occlusion Patters for Object Class Detection”. *Computer Vision Foundation. 2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, USA, 2013, pp. 3286-3293, doi: 10.1109/CVPR.2013.422.

Prakash, U. M. and Thamaraiselvi, V. G. "Detecting and Tracking of Multiple Moving Objects for Intelligent Video Surveillance Systems," *Second International Conference on Current Trends In Engineering and Technology - ICCTET 2014, Coimbatore, India, 2014, pp. 253-257, doi: 10.1109/ICCTET.2014.6966297.*

Rabaud, V. and Belongie, S. "Counting Crowded Moving Objects," *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), New York, NY, USA, 2006, pp. 705-711, doi: 10.1109/CVPR.2006.92.*

Redmon, J. “Darknet: Open Source Neural Networks in C”. https://pjreddie.com/darknet (accessed April 5, 2021).

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.*

Shao, S., Zhao, Z., Li, B., Xiao, T., Yu, G., Zhang, X. and Sun, J. “CrowdHuman: A Benchmark for Detecting Human in a Crowd”. *arXiv:1805.00123v1 (2018).*

Sindagi, V. A. and Patel, V. M. "CNN-based Cascaded Multi-task Learning of High-level Prior and Density Estimation for Crowd Counting." *14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 1-6. IEEE, 2017.*

Zhang, Z. “YoloV3 Implemented in TensorFlow 2.0”. *GitHub.* https://github.com/zzh8829/yolov3-tf2 (accessed April 5, 2021).