Pedestrian Detection, Tracking, and Behaviour Analysis

Team COMP9517 Prodigies

Nan Du
Department of Comp Sci & Eng
UNSW
Sydney, Australia
z5245818@ad.unsw.edu.au

Shu Wang
Department of Comp Sci & Eng
UNSW
Sydney, Australia
z5211077@ad.unsw.edu.au

Zhitong Chen
Department of Arts, Design & Architecture
UNSW
Sydney, Australia
z5300114@ad.unsw.edu.au

Yunseok Jang
Department of Comp Sci & Eng
UNSW
Sydney, Australia
z5286005@ad.unsw.edu.au

Chenming Yuan
Department of Comp Sci & Eng
UNSW
Sydney, Australia
z5355419@ad.unsw.edu.au

I. Introduction

As technology develops, there is a high demand for using computer vision all around the fields. One of the most widely used parts of it is object detection. There are significant number of projects and studies that have been researched.

In this project, numeral tasks will be implemented: pedestrian detection and tracking, pedestrian counting, and pedestrians' behaviour analysing, such as group analysing and walking direction analysing. Object detection and tracking can be accomplished by using machine and deep learning algorithms such as HOG (Histogram of Oriented Gradients) + SVM (Support Vector Machines) and YOLO (You Only Look Once). Those algorithms can be combined with SORT (Simple Online and Realtime Tracking) or DeepSORT (an extension to SORT) to improve tracking performance. Other tasks might be implemented by writing custom functions to manipulating results of pedestrian detection and tracking.

The dataset used in the project is from Segmenting and Tracking Every Pixel (STEP) benchmark. Two folders of images with annotations are provided for training. In addition, two folders of unlabelled photos are given for testing. In this project, the first test set will be mainly focused on testing and demonstrating purpose.

II. Literature Review

A. *YOLO and YOLOv5*

YOLO (You Only Look Once) is an object detection algorithm implemented using deep learning methods that divides images into a grid system, and each cell in the grid is responsible for detecting objects within itself. YOLO is one of the most famous object detection algorithms due to its speed and accuracy. [1]

YOLOv5 is pretrained on the COCO dataset, so it can recognise specific objects in 80 different classes, like ("person", "car", "bicycle", "boat", "bird", etc). There is an argument called *--classes*, which let users to choose which classes of objects will be detected in the input source. [2] There are different models of different number of hidden layers, different number of hidden nodes in each layer, and parameters in YOLOv5. Since YOLOv5 is popular, so there are many pretrained weights to do specific object detection tasks. [2]

B. *SORT, DeepSORT, and StrongSORT*

StrongSORT is a tracking algorithm upgraded from DeepSORT, which is a classic tracker based on Kalman Filter, which estimates an object's velocity and direction in the next frame and stable the tracking line, and Hungarian Algorithm, which does multiple calculations to find the best matching [3]. The traditional way of tracking objects is the SORT algorithm. The problem with SORT is that SORT creates too many identity switches when the object is blocked. Hence, we drove our attention towards DeepSORT and StrongSORT. The successors of SORT enrich the Kalman Filter with a deep learning component. Here is the comparison between SORT and DeepSORT [4].

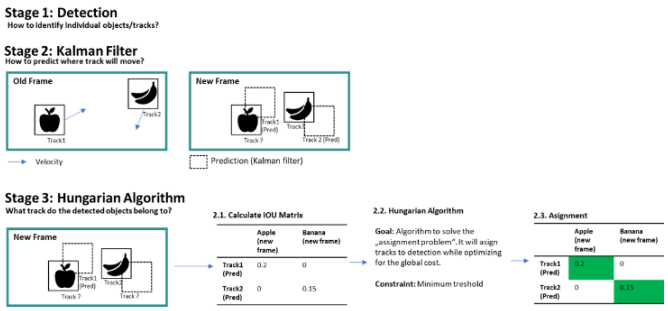


Figure 1 SORT

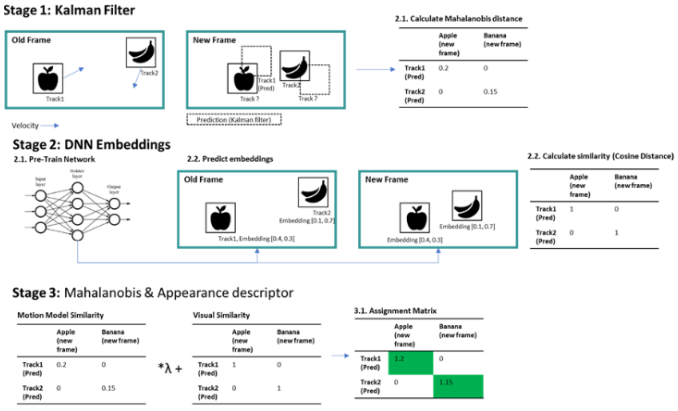


Figure 2 DeepSORT

StrongSORT is an improved version of DeepSORT.

By implementing YOLOv5 and StrongSORT, pedestrians can be tracked with minimum interference from other objects like tree branches.

C. *Bounding Box*

After recognising the pedestrian, a rectangular bounding box is drawn according to the object's centre point. An efficient way of doing this is using the OpenCV's *rectangle* function [5].

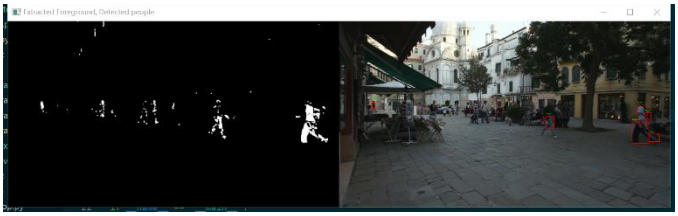
D. *setMouseCallBack*

Detecting mouse action has been used to crop a certain region out of the video. The setMouseCallBack waits for a mouse call and runs a function that's given as a parameter. [6]

III. Methods

A. *Methods Tried but not Used*

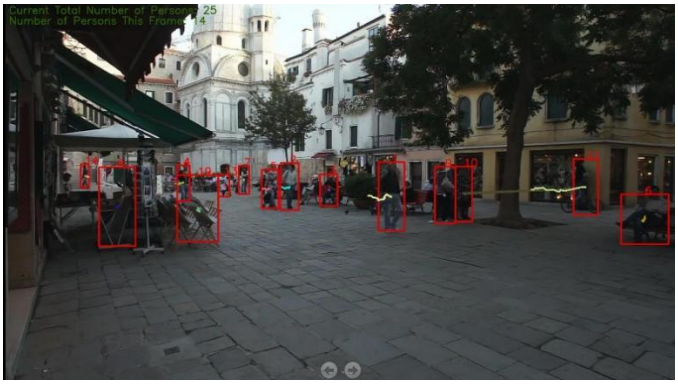
For task 1, the final model contains YOLOv5 for pedestrian detecting. At first, the attempt was to implement our detection algorithm; Zhitong suggested doing a background subtraction to filter the pedestrians out, but the result was unsatisfactory. Here is the result we got after implementing the algorithm.



The algorithm pictured the outline of all the moving objects. The consequence was that we did detect objects, but with no restraint of pedestrians, and boxes were drawn to parts of the pedestrian instead of the whole person. The second problem might be solved by changing the codes of drawing boxes, but this method is not worth the time and attention due to its low accuracy in detecting pedestrians.

Also, another choice was using HOG and SVM for object detection. However, since there are time consuming processes such as sliding window that must check all segments of the windows in the images. It is not compatible with real time video. [7] On the other hand, using deep learning can shorten the overall process up to significant number of time faster than processing each frame. Therefore, the final decision of using YOLOv5 has been made.

Then we start to try YOLOv5 for detecting pedestrians. YOLOv5 provides pre-trained weights, and we could also choose to train the model ourselves. Since in the dataset for this project, two videos were provided for training, so we labelled images in the training set ourselves and tried to train the model with the labelled images [8]; however, the results were not satisfying. Various errors occurred, including recognising chairs and benches as pedestrians like the output image below.



We reckoned that insufficient training videos led to this outcome. Then we tried some pre-trained weights in YOLOv5 like *yolov5s.pt* and *yolov5x.pt*, but both have the problem of

losing pedestrians far from the camera and misidentifying backpacks and bicycles as persons. Hence to solve this issue, we tried weights recommended for person detection tasks by the author of the model we used [9]. As instructed, *crowdhuman_yolov5m.pt*'s weight was utilised to improve the performance in detecting pedestrians [10]. The table below illustrates the differences in the numbers of pedestrians detected in the first ten frames of the first test video by three different weights we mentioned before.

Training Set / Pedestrians/Frame	YOLO5s	Provided Training Set	Crowdman YOLO5m
1	5	13	15
2	6	16	16
3	5	14	16
4	6	15	16
5	6	14	16
6	6	13	16
7	6	12	17
8	5	13	16
9	5	14	16
10	5	16	16

According to the output image and the table above, compared to pre-trained weights like *yolov5s.pt*, *crowdhuman_yolov5m.pt* could recognise people almost three times more than *yolov5s.pt*; compared to weights we trained by training sets, *crowdhuman_yolov5m.pt* produced fewer misidentification errors. Therefore, *crowdhuman_yolov5m.pt* is finally used as weights for YOLOv5 in this project.

YOLOv5 saves x-coordinates and y-coordinates of centres of the bounding boxes and widths and heights of the bounding boxes of detected objects in each frame into a txt file, so we could draw boxes for each frame using these txt files. Therefore, we tried to implement pedestrian tracking using these output txt files. However, since the simple tracking method could not do the re-identification task, it will give the pedestrian a new id once the pedestrian was lost in frames before. This problem makes us detect about 500 unique pedestrians in the test 1 video, which is almost 20 times of the number of unique pedestrians who appeared in this video. Then we searched for re-identification methods and found an open-source model called Yolov5+StrongSORT+OSNet implements pedestrians detection using YOLOv5, which is the detection method we used and implements pedestrian tracking and re-identification by StrongSORT. This open-source model perfectly implemented the pedestrian detection and tracking task we wanted in our project. Hence, we decided to use this model and implement our tasks based on the output of this model.

B. Adjusting Parameters of the Model

There are three main parameters in this model: weight of YOLOv5, ReID model for StrongSORT, and parameters for StrongSORT in *strong_sort.yaml*.

We have decided to use *crowdhuman_yolov5m.pt* for YOLOv5 in the last part. For the ReID model, we made a trade-off between model inference speed and accuracy and chose the *osnet_x0_25_market1501.pt*.

There are eight parameters in the *strong_sort.yaml* as shown in the table below:

Parameter	**Default Value**	**Description**
ECC	True	Activate camera motion compensation.
MC_LAMBDA	0.995	Matching with both appearance (1 - MC_LAMBDA) and motion cost
EMA_ALPHA	0.9	Updates appearance state in an exponential moving average manner.
MAX_DIST	0.2	The matching threshold. Samples with larger distance are considered an invalid match.
MAX_IOU_DISTANCE	0.7	Gating threshold. Associations with costs higher than this value are disregarded.
MAX_AGE	30	Maximum number of missed misses before a track is deleted.
N_INIT	3	Number of frames that a track remains in the initialisation phase.
NN_BUDGET	100	Maximum size of the appearance descriptors gallery.

We tried different values of MAX_DIST and MAX_IOU_DISTANCE. A smaller value of MAX_DIST will make the model cut more trajectories because it cuts the trajectory when the distance of centres in 2 frames exceeds the MAX_DIST value. Since we write a function to fill gaps ourselves, we decide to use the default value of MAX_DIST instead of smaller values. MAX_IOU_DISTANCE is like the similarity threshold of the model because IOU here is Intersection over Union. Since the model will identify two different pedestrians as the same pedestrian, we tried a larger MAX_IOU_DISTANCE value to solve this problem. However, it increases the total number of unique pedestrians in the video because a larger MAX_IOU_DISTANCE value makes the model detect one pedestrian as several different pedestrians. However, the problem of identifying different pedestrians as the same pedestrians still happens, so we decide not to change the MAX_IOU_DISTANCE value as well.

Therefore, we used default parameters in *strong_sort.yaml* for the model and decided to improve the performance by post-processing.

C. Implementation of Task 1

Each identical pedestrian will have a bounding box with a unique colour. To accomplish this task, we first used *np.random* to generate three random integers between *0* and *255* (the BGR parameter ranges from 0 to 255) to represent the three channels of BGR, and then stored them in a dictionary. When a new pedestrian appears and needs a colour, we will use the pedestrian's id as key to find a matching value in the colour dictionary. In this case, there is a very low probability that the colours are not unique. After obtaining the colour value, we passed it along with other parameters into the *rectangle* function in OpenCV to draw a bounding box.

After various iterations, StrongSORT was introduced to track objects.

The output file from YOLOv5 and StrongSORT has the following format, with the first six elements indicating frame

index (the index of frame starts from one), id of pedestrians, top left corner x coordinate of bounding box, top left corner y coordinate of bounding box, bounding box's width, and bounding box's height (in pixels).

```
2 1 761 457 51 134 -1 -1 -1 0
2 2 711 454 57 136 -1 -1 -1 0
```

The bounding box and trajectory were drawn according to the output file. We then created a 3D array *all_labels*, with *shape=(n_frames, n_person_this_frame, 5)*, where the *5* is defined as *[center_x, center_y, width, height, id]*. The problem is that we may lose track of some specific people, which can lead to miscounting pedestrian numbers. To fix this, Shu implemented a fill gap method. First, we created a three-dimensional matrix *id_tracks* (initialize it with 0), with *shape = (n_ids, n_frames, 4)*, where *4* is *[center_x, center_y, width, height]*. Then according to frame index and id, put matching records from *all_labels* into *id_tracks*. Then traversed *all_ labels* to find the number of frames that appeared for the first time and the number of frames that appeared for the last time of each id, and stored into the dictionaries *starts* and *ends*, with *id* as the key and frame index as the value. Then traversed *id_tracks* according to *id* starting from *starts[id]* to *ends[id]* until we find the continuous *[0,0,0,0]* (that is, *id_tracks[id][frame] = [0, 0, 0, 0]*) which indicates that this is a part of a gap. Then we stored the frame index, center_x, center_y, width, and height of the frame before and after the gap. Then *max_dist*, which is the maximum straight-line distance between centres of pedestrian with the same id in two consecutive frames, was set to 10. We used *math.hypot* to calculate the straight-line distance between two points. We chose this value because we have calculated that the moving distance between two frames was about five based on the moving distance of the pedestrian. However, if the value is too small, it may cut off some paths by mistake, while if it is too large, it may contain some wrong recognitions. Using the index of the frame after the gap and the index of the frame before the gap can obtain the size of the gap. If the distance between the two points before and after the gap is greater than (*gap_size*max_dist*), then we may recognise two different people as the same pedestrians. In this case, if we fill the gap, we will draw some boxes between the two people, which is an obvious error, so we chose to truncate it here by giving *center_ x, center_ y, width,* and *height* value -1 as a mark. If the distance is less than or equal to (*gap_size*max_dist*) then the gap is filled. *Width* became the average of *width* before and after the gap, so was the *height*. For *center_ x* and *center_y,* we used the last value minus the previous value then divided by gap_ size. Then the calculated value was filled into the position of the corresponding frame of the corresponding *id*, and the method fill gap was realised.

The last part of task 1 was drawing trajectories. For this part, first, we need to traverse *id_tracks* to check whether an *id* has matching elements of all *0* or all *-1*. For *0*, the tracking has not started or is already finished. For *-1*, it represents that the gap we chose was empty. For other cases, we used *rectangle* in OpenCV to draw the box of this pedestrian. Then we counted from the current frame to the previous frame and drew the line between two *centres* for both frames. We did this continuously until we reached *0* or *-1*.

D. Implementation of Task 2

For task 2, task 2.1 requested to display the total number of all unique pedestrians along with the output video. In the previous task, a list called *id_list* was used to record ids that have appeared. The length of this list represents the total number of pedestrians till the current frame. In the second subtask, the total number of pedestrians per frame is also needed to report. The method used for task 2.2 was keeping a counter named *n_person_this_frame* inside each loop for frames. After one loop ends, this counter was initialised to 0 and started to count for the next frame. We show the results of tasks 2.1 and 2.2 by using *putText* in OpenCV to print them in the top right corner of output frames.

The later subtasks were similar. In task 2.3, the user will choose a region over the video and draw a rectangle. There were two options. One is for the user to type in the coordinate of top-left, bottom-right corners when given the video size. The other is for the user to drag a mouse over a region of interest. The latter has been chosen over the first because it is more intuitive and user-friendly. Asking the user to type coordinate each time has a higher possibility of causing an error. Task 2.4 is built on top of task 2.3, which is to count the number of pedestrians in the ROI (region of interest, which is the rectangular region drawn by the user in task 2.3). In the first place, since it is quite similar to task 2.2, it was considered to use the same method with it (detect, create labels etc.). However, it is redundant for what it needs is only the label (coordinate of the pedestrians) and performing detection over the region repeatedly will slow the overall time complexity. Therefore, a chosen method is simply getting *id_tracks*, the label array created from previous steps, traversing through the data to check how many people's centre points are in the ROI. Rather than the bounding box, the centre point has been chosen to be used while counting the number. It made more sense because part of the bounding box being shown on the video does not always mean it can be seen as a human in the user's perspective (e.g., only a hand). On the other hand, if the centre point is in the region, it is more likely for the user to identify the object as a human since it will show at least a quarter of the object.

E. Implementation of Task 3

For task 3, the overall approach for 3.1 is dynamic threshold. In order to determine how many persons will be classified as a group. The distance between people and the camera and the distance between people are taken into consideration. The distance between people and the camera will be determined by calculating the area of the bonding box. The bigger area of the bonding box shows that this person is closer to the camera. Distance between persons will be determined by the current focusing person's bonding box width. Take an example of two person inside one frame. Person A has an area of 150, and persons B has an area of 210. When we are focusing on these two people, the area threshold will be calculated by 0.31*210 if these two people have an

area difference that is smaller than the threshold. Then the program checks the distance difference. We always want to calculate the threshold area of two people by times the area threshold coefficient of 0.31 with a bigger area. Moreover, we want to calculate the area difference between two people by subtracting the bigger area from the smaller area. The following calculation and logic explanation code show the reason:

```
each_frame = [ppl1, ppl2, ppl3, ppl4,...,pplN]
# fisrt_people = ppl1
for first_people in each_frame:
    # second_people = ppl2
    for second_people in each_frame:
        if first_people == second_people:
            # Ignore the same person
            continue

        # area_first = 150
        area_first = first_people.width*first_people.height
        # area_scond = 210
        area_second = second_people.width*second_people.height
        #area_diff = 150-210 = -60
        area_diff = area_first-area_second
```

If the second person has a bigger area than the first person, the area difference is negative. In this case, the area threshold is invalid because the area difference will always be smaller than the area threshold. Therefore, we design our code always to use a bigger area to calculate the area threshold and subtract the bigger area from the small area to calculate the area difference.

As we already know, the area difference is 60 for person one. The area threshold is 0.31*150=46.5, which is smaller than the area difference. For person two, the area threshold is 0.31*210 = 65.1, which is bigger than the area difference. In this case, two people's area threshold stands against of deciding if these two people are grouped to avoid these conflicts and prevent a future issue. Therefore, we always use the bigger area to calculate the threshold.

The distance threshold tells the distance between two people. Suppose these two people satisfy the area difference checking. Then if two people's centre points distance is less than the current focusing person's width, the program classifies them as a group.

When the program focuses on two new persons, suppose any of them is within a group. Then, the program will add persons who do not belong to any group into the existing group if they pass the area and distance checking.

The exact process happens in each frame on every person.

For task 3.2, the group analysis happens every frame after assigning every possible person into groups: the program stores groups and people's centre points for future detection of group formation or destruction. So, for example, if two individuals are getting closer to each other over the frame, and at a specific frame, the distance between these two individuals is smaller than the group distance threshold, which is three times of person's width, and area checking is valid, the program will put txt ‘HELLO!' on both individuals to show that they might be forming a group in the future.

For people inside a group, if the person is moving away from the centre point of the group over time, the program will image this person is leaving the current group and put the text 'BYE!’ on every person inside this group.

Both dynamic threshold and interframe tracking are applied to this subtask to achieve the goal.

For task3.3, we need to specify an edge area for each image to detect if people inside the frame are leaving or entering the scene. Only people inside this area will be detected. Interframe tracking is applied because we need the moving trajectory of a person to determine if the person is entering or leaving.

By applying the interframe tracking, we can calculate the distance changes between people and the image's centre point. If the distance increases over a specific time, the people are leaving. The program will put 'CYALL' at the top of this person's bonding box to indicate that. On the contrary, if the distance decreased, the program would put the text 'CHEESE' on the top of its bonding box to show that this person is entering the screen. The program will update the *prev_ppl* list every five frames to reduce the noise and clearly predict people's moving directions.

In the later section, *experimental results*, all defects, and their improvement will be analysed more detailly.

IV. Experimental Results

A. *Result and Evaluation for Task 1 Result*

Task 1 implements pedestrian detection and tracking function in this project.

Since there are no labels for test images in the dataset, so we randomly choose 3 images in the test set, label and count pedestrians in these images ourselves and compare these labels to output images (after filling gaps) of task 1 to calculate the metrics manually to evaluate the performance of pedestrian detecting. Frame 282, 219, and 259 are chosen by generating a random number from 0 to 450, where 450 is the total number of frames for the test set. Calculated metrics are shown in the table below:

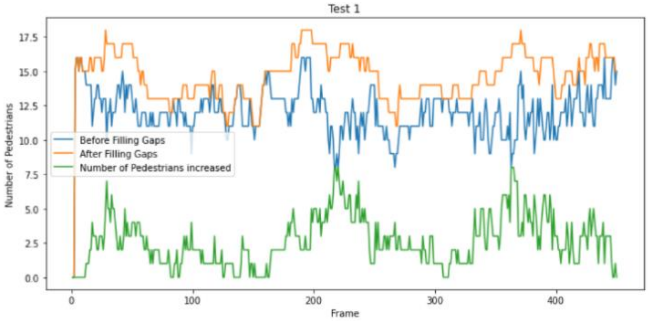
Frame	**No.282**	**No.219**	**No.259**
Number Counted	18	14	16
Number Detected	13	16	13
TP	13	13	13
FP	0	0	0
FN	6	3	4
TN	Not Applicable	Not Applicable	Not Applicable
Accuracy	72.22%	92.86%	81.25%
Precision	1.00	1.00	1.00

Recall	0.68	0.81	0.76
F1-score	0.81	0.90	0.86

TP (True positive) here is the number of objects that are pedestrians and were detected as pedestrians by the model as well; FP (False Positive) here is the number of objects that are not pedestrians but were detected as pedestrians by the model; FN (False Negative) here is the number of objects that are pedestrians but were not detected as pedestrians by the model; TN (True Negative) here is the number of objects that are not pedestrians and were not detected as pedestrians by the model, but this is not applicable in this project.

Since TN is not applicable here, so accuracy here is the percentage of pedestrians correctly detected as pedestrians, i.e. TN/# Counted. The calculations of Precision and Recall are not affected, so Precision = $\frac{TP}{TP+FP}$, and Recall = $\frac{TP}{TP+FN}$, and the F1-score is the adjusted Precision and Recall, which is $\frac{2*precision*recall}{precision+recall}$.

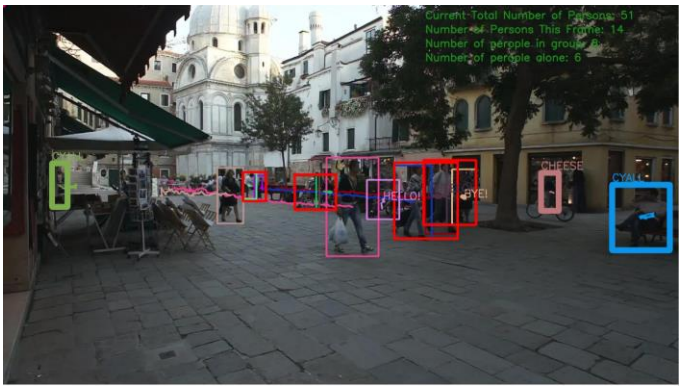
Since we implemented a gap-filling method when drawing trajectories, our evaluation for the pedestrian tracking part is mainly about the improvements made by our gap-filling method. The line chart below shows the number of pedestrians detected in each frame by the model as the blue line, the number of pedestrians in each frame after filling gaps as the orange line, and the number of pedestrians increased in each frame by the gap filling method as the green line.



From the line chart above, we can see that in most frames, the numbers of pedestrians are increased by the gap filling method, and the maximum number of pedestrians increased is 8, while the maximum number of pedestrians detected by the model in a single frame is only 16, so this is a vast improvement.

For task 2.3, by trial and error manually, performance (for this task, error) was evaluated. There were two errors. One is that the user can choose an invalid region (line and point). Another is user dragging other than from top-left to bottom-right. For task 2.4, evaluation has been done by manually checking frame by frame. There was no significant error, and it performed fine.

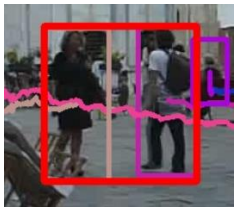
For task 3.1, The grouped people will be drawn with a red bonding box. The bonding boxes will surround all the people inside the group. And the information on grouped people and people walking alone will be shown at the top right of the image under the text for task 2.



For task 3.2, before the group formed, two people greeted each other by saying 'HELLO!' and when the group broke, saying 'BYE!'

Frame 219

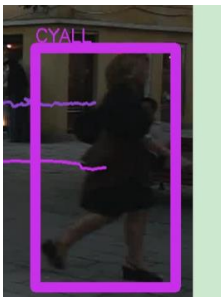
Frame 220



Frame 249

Frame 259

For task 3.3, as previously discussed inside the method section, determining the edge area and tracking the movement direction inside the edge area will help the program to determine if the person is entering or leaving the screen. In addition, to draw extra attention, the bonding box for these people is thicker than the bonding box for people in the middle of the scene.



The overall evaluation for task three is decided to delay updating the previous position lists for people and groups. The

reason for that is to reduce noise for people standing still. The following table shows the outcome of adding the delay update method. Moreover, we will focus on a person sitting at the edge of the image throughout the whole time. Notice that he did not enter or leave the screen. Without delay means, the previous position list is equal to the position list one frame before the current frame.

Person	With five frames delay	With no delay	Expected
Time the assigned label with entering/leaving the screen changed	101	184	0

As the table shows, with delay updating the previous list, the variation of the person's status reduces. This number will go down continuously by further increasing the delaying frame, however, to not affect other moving people's detection. Therefore, we choose to update the list every five frames.

The other evaluation for task 3 is to modify the area/distance threshold coefficient. This is important because it directly affects the number of groups shown inside one frame. We verified those parameters to obtain a better result. As the scene is a 3-dimensional area, but the image is 2-dimensional. Area and distance threshold is essential to identifying the distance between two persons. The big threshold will cause too many groups formed, which is not valid; too small threshold will make the program ignore some groups which should be present in the image.

V. Discussion

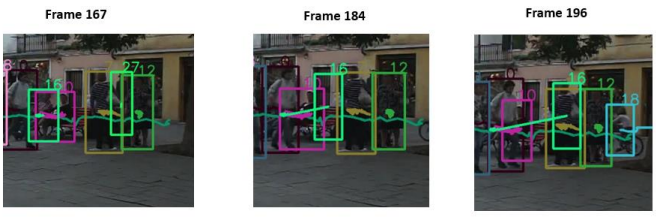
As we could see in the Experimental Result part, the average accuracy of our model reached 76.13%, and the average f1-score reached 0.823. The accuracy and f1-score are not very high but acceptable. It did not reach a very high accuracy because this test video had too many obstacles. There are pedestrians sitting on the bench and most part of these pedestrians are covered by the back of bench. Therefore, it is reasonable that these pedestrians could not be detected by the model. In the second video, there are fewer obstacles in the scene, so almost all pedestrians are detected successfully, even if some are very far from the camera. The image below shows the pedestrians detected in the second video after filling gaps.

There are still three problems in Task 1. The first one is that not all pedestrians in each frame are detected. This might be improved by labelling all pedestrians in images of the training set, and use these labelled images to train the current weights, *crowdhuman_yolov5.pt*.

The second one is that the program may recognise two different pedestrians as one person, like in the image below:

And it recognises one person as two or more pedestrians because there are about 36 unique pedestrians in this test video, but 52 unique pedestrians were detected at the end of this video (the result of task 2.1 in the last frame). To solve this problem, trying different Re-identification models could be helpful [11]. By learning a person's features and his or her matching similarity metric concurrently, the model can eventually minimise the mismatch of individuals.

Another problem is with the fill gap algorithm. It filled gaps that should not be filled, which means the two pedestrians are not the same person, but the model identifies them as the same pedestrians, and our gap filling method did not find that this is the gap that should not be filled. For example, in the image below, two different pedestrians in frames 167 and 196 were identified as the same pedestrians, and boxes were drawn between them when filling gaps.



This problem is also related to the re-identification model, but might be improved by setting more restrictions when deciding whether to fill the gap.

Task 2.3 had two errors. Those can be avoided by adding an extra code dealing with the situation. When given an invalid region, it can be ignored and print the error message asking for re-selecting the region. When the user drags from bottom-right to top-left, the reference points can be corrected as if it has been from top-left to bottom-right by swapping the coordinate. The method has partially failed and had some errors because possible user behaviour has not been fully considered. Therefore, flexible code to deal with the situation can resolve the errors.

Task 3 is highly correlated with the performance of task 1. The first problem for current implementation is our program's lack of ability to determine if the people are sitting or standing. During the test data set, the person sitting should not be grouped with people standing.

The second problem is when running group analyses. If one person is currently having multiple group action happens. The printed text overlay with each other, which makes the information easy to lose and hard to distinguish. This happens because the current implementation combines these two functionalities as one function, which means, each person under the situation will have to be put text multiple times at the same position.

The third problem is unable to detect people leaving or entering the screen in the middle of the image. The reason for that is by defining the Edge area, we restrict the person who inside that area to be detected as potentially leaving or entering the screen. This method applies for this test data set but might not be applicable for future data sets.

VI. Conclusion

For task 1, all goals are finished with reasonable accuracy. Most of the pedestrians are recognised and have bounding boxes with unique colours, with matching coloured tracking lines of centre points are drawn for each pedestrian. Most of pedestrians have a trajectory from the position that this pedestrian first appeared to the position that this pedestrian left the scene from, especially for the second test video. However, there are still problems like what we stated in the Discussion part. Therefore, future improvements could be made to the pedestrian detection part by trying to train the weights by labelling images in more detail based on the current weights we used, which is *crowdhuman_yolov5m.pt*; and future improvements could be made to the pedestrian detection part by trying different re-identification models and setting more restrictions before applying gap filling method to gaps.

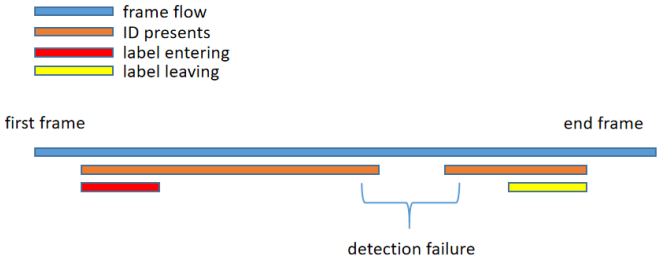
Task 2 was relatively easy, and we completed it satisfactorily. The total number of pedestrians is shown in the top left corner of the output video, with a unique-coloured binding box showing for each person. Users can choose to 'draw' a rectangle. In the rectangle, the same approach as task 2.1 is used to count the number of pedestrians. For tasks 2.3 and 2.4, they partially worked with having some errors which was improved as elaborated in Discussion. It would be recommended to work on a more optimised method. In terms of functionality, the current method works decently. However, it sometimes takes a few seconds to show up in the region, which might be a problem if an instant response is needed.

Current task 3 allows forming group of people, showing information when forming/breaking the group, and detecting people entering/leaving the screen at the edge area of the screen.

One possible improvement for the problem described inside discussion section is to modify the distance threshold. Nevertheless, this is risky because it may cause a further problem which the program is hard to group people who are both standing. Another solution is to add another threshold called 'ratio' threshold, which calculates the person's bonding box ratio from its width and height. Only the ratio difference smaller than the threshold will be grouped.

For future improvement for second problem of task 3, we can split the group analyses function into two and run them separately. Furthermore, put forming and breaking group information separately at different places. For example, breaking group information could be written under the group's bonding boxes instead of the person’s.

For third problem. We can take this part of the implementation into the pre-processing section by fixing that after reading and storing all movement information after reading the text file. We can determine every unique ID's first and last appearing frame. Doing so can determine people leaving or entering the screen before reading the images. The following indication shows the overall idea.



This method detects the people leaving or entering the screen without determining the edge area. Because in the future, we might encounter data set that contains people leaving or entering the screen in the middle of the image. The current algorithm would not handle this situation successfully.

The overall aspect of teamwork for this project is decent. We use GitLab to separate the work branch in order to keep track the version of our implementation. Group meeting happens regularly throughout the whole project. Most of group members contributes great effort towards this assessment. However, we do encounter schedule issue and some of our teammates did not corporate with others. We might increase the time of meeting in the future to track everyone's working more regularly to avoid this issue. The team COMP9517_Prodigies would like to thank all the team members who contribute to this project and each and every staff who provides help and clarification to our team members.

REFERENCES

[1] J. Redmon, "YOLOv5 Documentation", github.io. https://docs.ultralytics.com/, (accessed Aug. 5. 2022).

[2] G. Jocher, "ultralytics/yolov5", github.com. https://github.com/ultralytics/yolov5, (accessed Aug. 5, 2022)

[3] Y. Du, Y. Song, B. Yang, Y. Zhao, "*StrongSORT: Make DeepSORT Great Again*", arxiv.org. https://arxiv.org/pdf/2202.13514.pdf, (accessed Aug. 2, 2022)

[4] D. Pleus, "*Object Tracking - SORT and DeepSort*", LinkedIn.com. https://www.linkedin.com/pulse/object-tracking-sort-deepsort-daniel-pleus/, (Accessed Aug. 2, 2022)

[5] OpenCV, "Drawing Functions in OpenCV", opencv.org. https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html, (Accessed Aug. 4, 2022)

[6] A. Rosebrock, "Capturing mouse click events with Python and OpenCV", pyimagesearch. https://pyimagesearch.com/2015/03/09/capturing-mouse-click-events-with-python-and-opencv/, (accessed Jul. 21, 2022)

[7] K. Sakmann, "Vehicle tracking using a support vector machine vs. YOLO", Medium.com https://medium.com/@ksakmann/vehicle-detection-and-tracking-using-hog-features-svm-vs-yolo-73e1ccb35866 (accessed Aug. 5, 2022).

[8] G. Jocher, "Train Custom Data", github.com. https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data, (accessed Aug. 1, 2022)

[9] M. Brostrom, "*Yolov5 + StrongSORT with OSNet",* GitHub. https://github.com/mikel-brostrom/Yolov5_StrongSORT_OSNet, (accessed Aug. 1, 2022)

[10] M. Brostrom, "*Yolov5 + StrongSORT with OSNet",* GitHub. https://drive.google.com/file/d/1gglIwqxaH2iTvy6lZlXuAcMpd_U0GCUb/view, (accessed Aug. 1, 2022)

[11] E. Ahmed, M. Jones, T. Marks, "*An Improved Deep Learning Architecture for Person Re-Identification"*,openaccess.thecvf.com. https://openaccess.thecvf.com/content_cvpr_2015/papers/Ahmed_An_Improved_Deep_2015_CVPR_paper.pdf, (Accessed Aug. 3, 2022)