Master's degree in Computer Science,

curriculum in Security Engineering

a. Y. 2020/21

Exam of Urban Security

Social Distancing Detection

Project documentation

Team members:

Giovanni Forleo – g.forleo3@studenti.uniba.it

Paolo Gasparro – p.gasparro4@studenti.uniba.it

Roberto Colella – r.colella17@studenti.uniba.it

Giuseppe Piccininni - g.piccininni5@studenti.uniba.it

Index

1. Introduction

The goal of this document is to describe and introduce the project developed for the exam of Urban Security.

For this project it was proposed a model of social distancing detection using computer vision to detect people and distance between them, inside a video or through a set video camera using specific dataset to test the reliability of the system.

While studying the state of the art and developing the system, some needed constraints appear to allow the correct performance of the model. In the sequent section the state of art about crowd and people detection and distance computation between people is discussed. In section 3 a description about the dataset chosen and specific characteristics of it is exposed. In section 4 it will be presented the flow of the system, how it works and which existing methods and models the team applied to obtain reliable results. In section 5 results will show with computed videos on the chosen datasets, then conclusion is drawn, and possible future work is discussed.

2. State of the art

Crowd and people detection is a computer vision task which uses computer vision algorithms. Most of the existing work for detection/identification of people, groups of people, or even for the estimation of body parts (in a high-level analysis) have been focused on noncrowded situations. First, a crowd is something beyond a simple sum of individuals. The crowd can assume different and complex behaviors as those expected by their individuals. The behavior of crowds is widely understood to have collective characteristics that can be described in general terms. In a crowded situation, it is difficult to segment and track accurately everyone, mostly due to severe occlusions. In fact, when high-density video sequences are employed, the accuracy of traditional methods for object tracking tends to decrease as the density of people increases. Crowds can be characterized considering:

- image space domain
- sociological domain
- level of services
- computer graphics domain. [1]

Use cases of crowd analysis:

- Shopping Malls: Monitoring of High-Traffic Areas: people counters are used in Shopping centers for measuring the number of visitors in each area. People counters also assist in measuring the areas where people tend to congregate, the areas where people tend to gather are often charged higher rent.
- Museum and libraries: Non-profit organizations often use visitor counts as evidence when applying for grants or other financial aid, when planning for seasonal staffing, or other strategic operational decisions. In cases where tickets are not sold, such as in museums and libraries, counting is either automated or staff keep a log of how many clients use different services.

- Stadiums: crowd management: People counters are used to measure the traffic flows of events; traffic patterns are used to improve traffic flow, particularly when large crowds are entering and exiting the stadium.
- Smart office buildings (Fire Management): In the case of fire, people counters can be used to approximate the number of people inside the building. [2]

Crowd and people detection may concern the detection of distance between people in crowd, in open spaces or inside a building to improve the computation of social distancing, crucial due to the COVID-19 pandemic situation. This situation helps the researchers in developing new solutions for crowd management and detection of distance between people in crowd. To date, there are no established methods for determining the distance between two dynamic objects in a video stream created by using a single camera. All that people can achieve so far is measuring the distance of an object of known size where the distance between the camera and the object is also be known at first spot, often considered as a reference. However, there is no way we can tell these two parameters from a video containing people who continuously change their positions. [3]

Several methods exist in the literature and fall under two main categories: object detectors and pose estimation techniques. The former identifies objects by bounding a box around them, while the latter detects the human joints and connects them resulting in pose estimates. [4] On the one hand, object detectors, such as YOLO models, are more general-purpose than pose estimation techniques, but their utility for identifying human subjects may require pruning and/or retraining. In addition, they do not offer further information about the detected objects and their bounding boxes can be over-sized or incomplete. On the other hand, pose estimators are specialized models; hence, they are more suitable to detect people in a scene. [5]

Talking about YOLO is a deep learning method that aims to detect objects that unite the components of the detection object into a single neural network in the entire image. The system of the YOLO method itself, namely YOLO, divides the input image into regions or grids measuring S x S. If the center of an object falls into a grid cell, it is the grid cell that is responsible for detecting the object. Each grid cell predicts the bounding box and confidence. The value of this confidence represents how confident a model is in the box containing the object and how accurate the box is being predicted. Each bounding box consists of 5 predictions: x, y, w, h and confidence. The x and y values represent the midpoint coordinates relative to the grid cell boundaries. At the same time, the values of w and h represent the width and height relative to the whole image. YOLO gets the class confidence value specifically for each bounding box by multiplying the class probability value with the confidence value from the bounding box. This value indicates the probability class that appears in the bounding box and shows how accurately the bounding box predicts an object. YOLO-v5 is the latest version of the current YOLO development, based on YOLO-v4. The Processing Speed of the YOLO-v5 is drastically increased, with the fastest speed reaching 140 Frames per Second (FPS). YOLO-v5 is small, even 90% smaller than YOLO-v4, making it possible that YOLO-v5 can be deployed to an embedded device. Furthermore, a higher level of accuracy and a better ability to recognize small objects. [6]

YOLO is able to automatically detect some elements inside a frame of a video but can’t compute and detect distance between two bounding boxes. In [6] researchers try to define a way to detect distance of each human. It will be done using the Euclidian distance equation using 2 points. Euclidean distance is the calculation of the distance from two points in Euclidean space to study the relationship between angles and distances. The Euclidian distance is the most common distance used for numerical data, for two data

points x and y in d-dimensional space. For two-dimensional calculations at the coordinates (x1, y1) and (x2, y2). Then this value will be compared with a generic value of distance to check if the social distancing is not violated.

Some interesting approaches to the distance detection using YOLO as object detector are:

- Dist-YOLO [7], an improvement on the existing architecture in order to predict the absolute distance of objects using only information from a monocular camera. The proposed scheme yields a mean relative error of 11% considering all the eight considered classes and the distance range within [0, 150] m, which makes the solution highly competitive with existing approaches.

- A social distance monitoring solution in low light environments in a pandemic situation [8] working on YOLOv4. Experimental results show that the proposed model exhibits good performance with 97.84% mean average precision (MAP) score and the observed mean absolute error (MAE) between actual and measured social distance values is 1.01 cm. However, the proposed model uses an already outdated architecture and seems to work well on only one type of camera angle.

3. Dataset

The videos chosen to test the model must have specific constraints to obtain reliable results. At first the shot must not be moving but stationary. The quality of the video must be medium-high, because YOLO with a bad video quality could detect people as false positive or lose a large amount of people observable inside the video.

Fortunately, the resolution of the video is not crucial, because the model foresees a solution to resize each video captured by the camera or loaded inside the storage. The datasets used try to cover all the possible situation in which a camera may be installed.

Specific video with high density of people in each frame or with a specific shot (too angled or decentralized) could create problems during the phase of setting the system to prepare it for the detection phase.

In this dataset [9] was really useful to test the system. Different situations are reproduced that reflect real world context and even different framings to check the reliability of the model, especially with different perspective and bird eye view to verify.

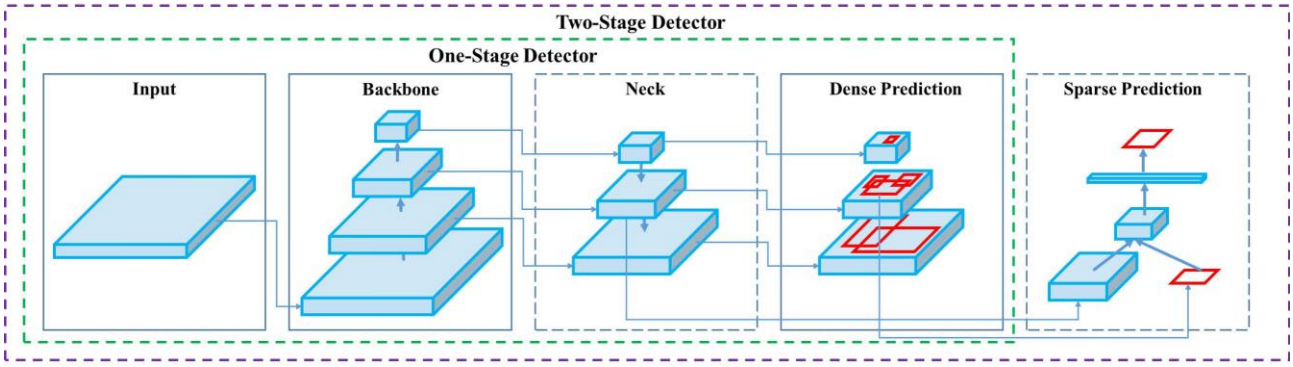
The choice of the videos was affected by the perspective. The model to compute correctly the distance, mustn't have a complex perspective (inside the frame of the video), to avoid a complex computation of it and provide a more reliable result.

Other videos used to test the model will be attached to the project delivered for the exam. These videos had been found surfing on the web, through different provider of generical videos (like YouTube).

4. Models and methods

Social distancing is described as a method by enforcing physical distance between one another and hence reducing the spread of contagious disease. Naturally, to track how social distancing is enforced, detection technologies are developed. Some common methods include the uses of wearable devices, voluntarily downloaded apps, wi-fi usage analysis, and image processing technologies. For our distance detection project, we would specifically look for image-based solutions.

The current state of the art object detection algorithms can be divided into two distinct classes, which are **two-step** detectrons and **one-step** detectron. They differ by the number passes the image would go through a neural network. Two-step detectrons are represented by approaches such as the R-CNN[10] [11] (regional based convolutional neural network) class of algorithms, and one-step detectrons are represented by YOLO (you only look once) and SSD (single shot detector)[12]. **Figure 1**



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 1: Object detector

Two-step methods typically involve generating unlabeled bounding boxes first (called regional proposals), then using a learned model such as a support vector machine or convolutional neural network to classify each region. Due to the multi-step approach, R-CNNs are typically slower. In the original model the bounding boxes (called regional proposals) are generated with a traditional computer vision algorithm called selective search, then the regional proposals are classified with a mixture of convolutional neural networks and support vector machines. The initial approach is improved over time. In the most recent iteration, Faster R-CNN, the regional proposals steps are replaced with a Resnet-Based Convolutional neural network, which significantly improves the speed for generating the proposals, and brings the speed of the overall program to close to real time.

The one step models on the other hand uses a different approach, which only one pass through a neural network is needed for YOLO, the image is first divided into a S by S grid, then a convolutional neural network is used to generate bounding boxes and class prediction for each cell in the grid. The result is then integrated by the network to form the final prediction.

Unlike the R-CNN approach, since there is no need to run a neural network model with various proposals, the YOLO model is significantly faster. The current iteration of it is the YOLO v5 model released on July 24th (**Table 1**), though it isn't developed and trained by the author of the original paper, it has a similar architecture as the original models and it has slightly improved precision and run time compared to the previous generations, as shown by a comparison by the author with the coco dataset.

Model	size (pixels)	mAPval 0.5:0.95	mAPval 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Table 1: Yolov5 pretrained models

To choose the best object detection model that will be used by our project, we first need to pick the model that has a reasonable speed. From a comparative study [8] for the performances of these models in different papers. The YOLO model has a significantly higher FPS, while R-CNN tend to be very slow. This is expected, since faster R-CNN requires two passes in two different neural networks in series, while YOLO model only require one pass. Since the goal for the project is for developing a real-time algorithm, and hopefully without the need of a GPU, an R-CNN typed model would be infeasible to implement.

For this reason, as well as the fact that YOLO code is very well documented on GitHub, we have chosen the most recent YOLOv5 (ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite (github.com)) as our code for object detection.

4.1 System Flow

The whole project has been developed in Python with the use of PyCharm as IDE. The developed system uses YOLO as object detector model for the reasons explained above. The flow can be summarized in the following phases:

1) Open stream video

After the *tkinter* module import, the *askopenfilename* function allows the selection of the video that must be analyzed. **Figure 2**

```
filename = askopenfilename(title='Select a video file...',
                           filetypes=[("all video format", ".mp4"),
                                      ("all video format", ".flv"),
                                      ("all video format", ".avi")])
```

Figure 22: Open stream video source code

2) Convert video

Using *ffmpeg* libraries, the video selected in the previous phase is processed. The goal of this task is to reduce the video shape reduce the time complexity of the entire inference process. **Figure 3**

```
def convert_video(path):
    """
    Converts the video in a compressed mp4 version
    :param path: The path of the video to convert
    :return:
    """
    # split a given input string into different substrings based on a delimiter
    # and take the first cell of the array
    filename = os.path.basename(os.path.normpath(path))
    compressed_path = filename.split('.')[0]  # "test"
    compressed_path = 'compressed_' + compressed_path + '.mp4'  # "test.mp4"

    # remove the file if already exists
    if os.path.exists(compressed_path):
        os.remove(compressed_path)

    # Convert video in a specific format
    os.system(f"ffmpeg -i {path} -vcodec libx264 -vf scale=500:-2 {compressed_path}")
    return compressed_path
```

Figure 33: Convert video source code

3) Display video

After the video has been converted, a window displaying the video automatically appears as showed in the following image. **Figure 4**

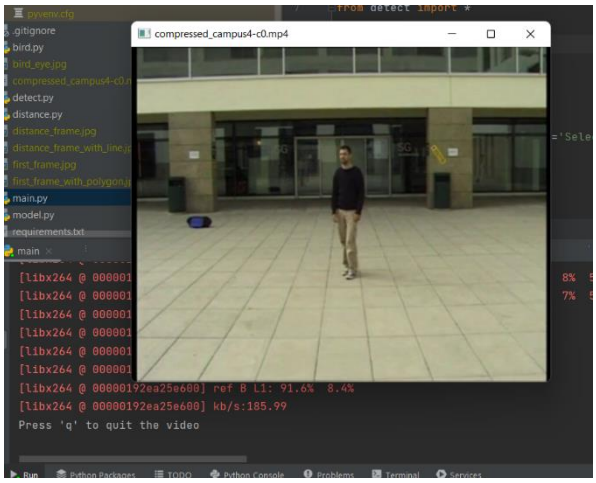


Figure 44: Displaying video

The user can watch the entire video or skip it by pressing the ‘q’ button on the keyboard.

4) Import YOLO model

Pytorch enables to import the specific YOLO model from the official GitHub repository, choosing the different pretrained s, m, l, or x. To achieve the best compromise between high efficiency and effectiveness, the imported model was the "l" one. However, by changing the parameter of the function, any other type can be imported.

In addition, the inference process can be executed using either the CPU or GPU, according to the device specs. **Figure 5**

```
def load_model(model):
    """
    Loads the specific Yolo model
    :param model: the model to load
    :return:
    """
    model = torch.hub.load('ultralytics/yolov5',
                           'yolov5' + model,
                           pretrained=True,
                           verbose=False)
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    return model
```

Figure 55: Import YOLO model source code

5) Recover the first frame of the video

This function purpose is to recover the first frame of the selected video and saves it in the project path. In this way, the next two step can be correctly executed. **Figure 6**

```python
def recover_first_frame(filename):
    """
    Recovers the first frame of the selected video and saves it in the project path
    :param filename: The path of the video
    :return:
    """
    cap = cv2.VideoCapture(filename)

    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            cv2.imwrite('first_frame.jpg', frame)
            break
    cap.release()
```

Figure 66: Shot the first frame of the video source code

6) Set the ROI

The ROI, Region of Interest, are samples within a data set identified for a particular purpose. The concept of a ROI is commonly used in many application areas. In computer vision and optical character recognition, the ROI defines the borders of an object under consideration. In many applications, symbolic (textual) labels are added to a ROI, to describe its content in a compact manner. Within a ROI may lie individual points of interest (POIs). [13]

To achieve a better result, points of interest must be identified directly by the user that wants to operate the system. To this end, the drawing on the first frame of the video is manually carried out, and so even the region of interest needed to detect people and compute distance between them, with perspective.

As showed below in the **Figure 7**, the *recover_roi_points* function is called until the user accepts the result of its drawing.

```
# Recover the ROI and ask to confirm the choice
answer = False
while not answer:
    mouse_pts = recover_roi_points()
    window_name = 'first_frame_with_polygon.jpg'
    answer = ask_to_confirm_roi(window_name)
```

Figure 77: Set the ROI source code - part 1

To track the drawing, the *draw_polygon* function become a parameter of the cv2 *setMouseCallback* function, allowing to detect mouse events until the polygon has been completely created by the user.

To improve visibility, the draw_polygon function is not shown in this document. However, it can be found in the source code attached to this document, and more precisely in the file *roi.py.* **Figure 8**

```
def recover_roi_points():
    """
    Function to recover the four points of the polygon drawn on the image
    :return: The four points
    """
    global mouse_pts, img, filled

    # Takes only the name of the file without its extension
    window_name = 'first_frame'

    # Opens the window to start drawing the 4 points
    cv2.namedWindow(window_name)
    cv2.setMouseCallback(window_name, draw_polygon)

    # Recovers and saves into the project path the first frame of the selected video
    img = cv2.imread('first_frame.jpg')
    # img = cv2.imread('first_frame.jpg')
    while not filled:
        # if we are drawing show preview, otherwise the image
        if preview is None:
            cv2.imshow('first_frame', img)
        else:
            cv2.imshow('first_frame', preview)
        k = cv2.waitKey(1) & 0xFF
        if k == ord('q'):
            break

    cv2.imwrite(window_name + '_with_polygon.jpg', img)
    cv2.destroyWindow(window_name)
    return mouse_pts
```

Figure 88: Set the ROI source code - part 2

The result of this interaction can be summarized in the two following figures. On the left, the first frame of the video where the user must draw the polygon. On the right (**Figure 9**), the region of interest drawn (**Figure 10**).

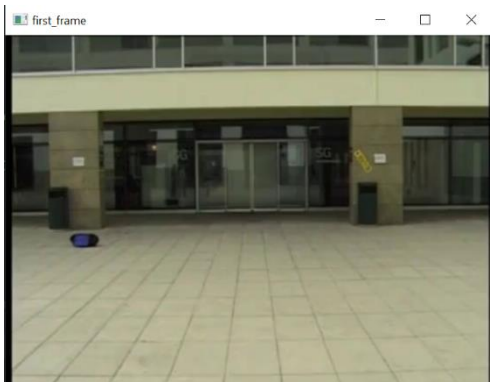
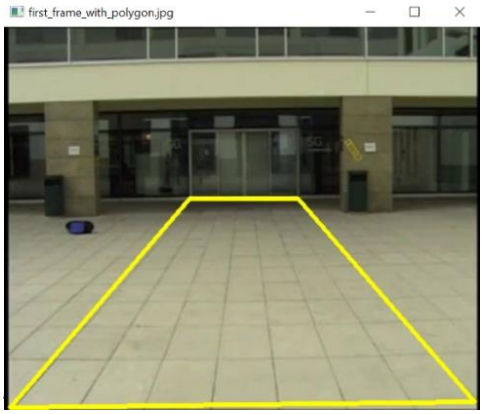


Figure 99: The first frame of the video



the ROI

7) Set the distance

As in the previous step, also the distance is set by the user. However, the method used for this step is slightly different. In this case, the user will not use the first frame, but can choose the one he prefers using the key “s”, moving freely within the video with the use of the keys “z” and “x”. The function that is in charge of doing this is *choose_distance_frame.*
Figure 11-12

```python
def choose_distance_frame(filename):
    """
    :param filename:
    :return:
    """
    cap = cv2.VideoCapture(filename)

    # Check if camera opened successfully
    if not cap.isOpened():
        print("Error opening video stream or file")
        sys.exit()
    frame_list = []
    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            # Display the resulting frame
            frame_list.append(frame)
        else:
            break
    cap.release()
    print("Choose the frame on which you will draw the distance",
          "Press 'x' to display the next frame",
          "Press 'z' to display the previous frame",
          "Press 's' to choose the frame")
    next_frame = 1
    current_frame = 0
    previous_frame = -1
    cv2.imshow("Choose the frame on which you will draw the distance", frame_list[current_frame])
```

Figure 11: Set the distance source code - part 1

```python
while True:
    key = cv2.waitKey(0) & 0xFF

    if next_frame <= len(frame_list) - 1 and key == ord('x'):

        previous_frame = current_frame
        current_frame = next_frame
        next_frame = next_frame + 1
        cv2.imshow("Choose the frame on which you will draw the distance", frame_list[current_frame])

    elif previous_frame >= 0 and key == ord('z'):

        next_frame = current_frame
        current_frame = previous_frame
        previous_frame = previous_frame - 1
        cv2.imshow("Choose the frame on which you will draw the distance", frame_list[current_frame])

    elif key == ord('s'):
        cv2.imwrite("distance_frame.jpg", frame_list[current_frame])
        cv2.destroyWindow("Choose the frame on which you will draw the distance")
        break
```

Figure 12: Set the distance source code - part 2

Once the ideal frame is chosen (**Figure 13)** (the one within which the consumer knows the exact value of a segment), the user draws the line and enters the measurement of that segment into the command prompt. **Figure 14-15**

Same as before, the result of the interaction can be summarized in the two following figures. On the left, the first frame of the video where the user must draw the line to set the distance. On the right, the line drawn.

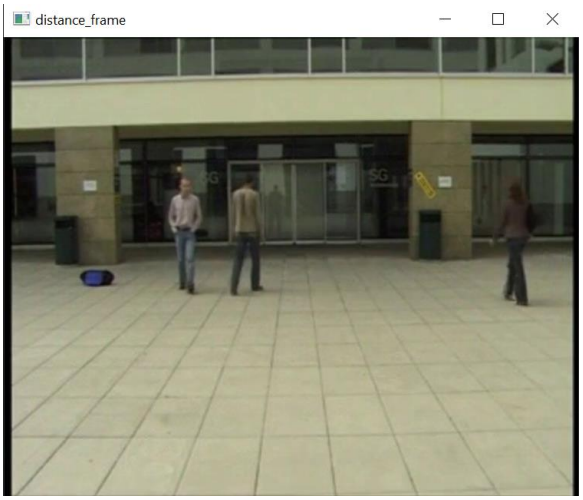


Figure 13: Frame where the user chooses to draw the line

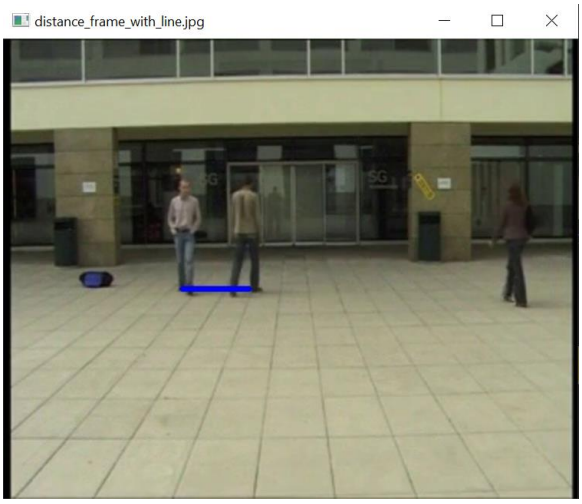


Figure 14: Frame with the line drawn

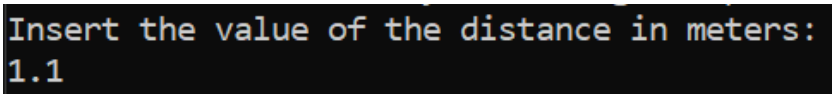


Figure 1105: Terminal screen of the user inserting the drawn measure

Once the measurement has been recorded, the system will calculate, in the specific video, the actual value of pixels that correspond to one meter.

The team took this decision in accordance with the **World Health Organization** point of view. They declared that people should maintain at least 1 meter (m) distance from each other to control the spread of this disease [14]. In this way we will know the threshold not to be exceeded to respect the social distance and to correctly proceed with the inference process.

8) Perform the inference process

Once the input video, its region of interest, and the known distance value have been retrieved, in this last phase, the designed model allows to process the video identifying the people within it and showing, in real time, the number of distance violations.

To have a benchmark, the final interface will show 3 horizontally concatenated frames:

1. **Yolov5**: represents the running model of the same name that implements distance calculation without considering perspective estimation.
2. **SDD (Social distancing detection)**: shows on screen the same frame of Yolov5. In this case, however, the calculation of distances considers the perspective of the frame and is therefore as realistic as can be obtained.
3. **Bird Eye view**: shows how the perspective of the SDD frame is calculated. A bird's-eye view is an elevated view of an object from above, with a perspective as though the observer were a bird. In this way the distances are calculated correctly, and the violations detected are accurate.

The functions responsible for the implementation of the above-mentioned functionalities are located within the file *detect.py* in the project folder attached to this report. However, some code fragments will be shown to better understand the functioning of this inference phase.

The core of the entire distance detection algorithm lies within the functions *detect_people_on_video and detect_people_on_frame.* In the figure below, the entire *detect_people_on_video* is shown. **Figure 16**

```python
def detect_people_on_video(model, filename, fps, height, width, pts, confidence):

    frame_number = 0
    one_meter_threshold_bird = 0
    one_meter_threshold_yolo = 0

    # Capture video
    cap = cv2.VideoCapture(filename)
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = save_video(fourcc, fps, width, height)
    filter_m = compute_bird_eye(pts)
    img_bird_eye = cv2.imread('bird_eye.jpg')
    # Iterate through frames and detect people
    vidlen = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    with tqdm(total=vidlen) as pbar:
        while cap.isOpened():
            # Read a frame
            ret, frame = cap.read()
            # If it's ok
            if ret:
                bird_eye_frame = img_bird_eye.copy()
                frame_number = frame_number + 1
                centers, bird_centers, frame, one_meter_threshold_bird, one_meter_threshold_yolo = \
                    detect_people_on_frame(
                        model,
                        frame,
                        confidence,
                        height,
                        width,
                        frame_number, one_meter_threshold_bird, one_meter_threshold_yolo, filter_m, bird_eye_frame)

                # Write new video
                out.write(frame)
                cv2.imshow("Detecting people", frame)
                cv2.waitKey(1)
                pbar.update(1)
            else:
                break
```

Figure 16: detect people on video source code

This function first computes the bird eye view using *compute_bird_eye* (**Figure 17**), then iterates through the entire video and, for each frame, calls the *detect_people_on_frame* function passing appropriate parameters.

```python
def compute_bird_eye(pts):
    """
    Computes the bird's eye view of the selected area of the original image bounded by the four points of the polygon
    drawn by the user
    :param pts: The four points of the polygon drawn by the user
    :return:
    """
    img = cv2.imread('first_frame.jpg')
    height = img.shape[0]
    width = img.shape[1]

    # mapping the ROI (region of interest) into a rectangle from bottom left,bottom right,top right,top left
    input_pts = np.float32([pts[0], pts[1], pts[2], pts[3]])

    width_out = width * 2
    if height == 282:
        height_out = height*4

    else:
        height_out = height * 3
    final_width = width_out + width
    final_height = height_out + height

    output_pts = np.float32([[width, height], [width_out, height], [width_out, height_out], [width, height_out]])

    # Compute the transformation matrix
    filter_m = cv2.getPerspectiveTransform(input_pts, output_pts)
    out = cv2.warpPerspective(img, filter_m, (final_width, final_height))
    cv2.imwrite('bird_eye.jpg', out)
    return filter_m
```

Figure 1711: compute bird eye function

After retrieving the dimensions (height and width) of the compressed video, the ROI points are converted to an appropriate format and the bird eye perspective calculation is then carried out.

A very accurate work was required for choosing the coordinates where to translate the user-drawn polygon. This approach is not perfect for all videos and shots, but it is the one that produced the best results.

The final choice of points is driven by a series of challenges to which we tried to find the best possible solution.

It is important to remember that due to the compression of the video in phase 2, the videos with high resolution will be reduced and will have a maximum size of 500 width, and consequently 282 of height, in proportion.

In general, these "big" videos will need a bigger stretch in terms of length, and this is the motivation behind the two if statements within the function.

A problem presented to the team was to "fit" this bird eye view within one of the 3 frames to center the information content and exclude non-core areas. After carefully evaluating these issues, the coordinates chosen are plotted in the **Figure 18** using Geogebra [15] as tool. The red polygon represents the ROI input points and the blue polygon represents the translated points.

Figure 18: input ROI points and output bird eye points in Geogebra

Finally, the *getPerspectiveTransform* and *warpPerspective* methods are called on the previous points to obtain the final bird eye view. Considering the example above, the calculated bird eye view will be as follows (**Figure 19**):

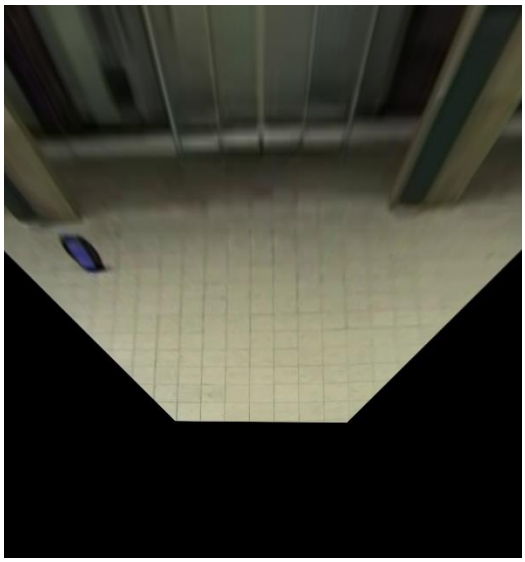


Figure 19: computed bird eye view

Yolo parses the single frame to get the bounding boxes of the objects it is able to find within it. The returned *xyxy* array will therefore contain the coordinates of the objects with the specific class and confidence. The information content that is needed to proceed with the algorithm is retrieved, i.e. objects of class 0 (people) with confidence above a certain threshold (the team decided to use 0.5, i.e. 50%). **Figure 20**

```
def detect_people_on_frame(model, sdd_frame, confidence, height, width, frame_number,
                           one_meter_threshold_bird, one_meter_threshold_yolo,
                           filter_m, bird_eye_frame):

    sdd_violations = 0
    yolo_violations = 0

    # Pass the frame through the model and get the boxes
    results = model([sdd_frame[:, :, ::-1]])

    xyxy = results.xyxy[0].cpu().numpy()  # xyxy are the box coordinates
    #          x1 (pixels)  y1 (pixels)  x2 (pixels)  y2 (pixels)   confidence        class
    # tensor([[7.47613e+02, 4.01168e+01, 1.14978e+03, 7.12016e+02, 8.71210e-01, 0.00000e+00],
    #         [1.17464e+02, 1.96875e+02, 1.00145e+03, 7.11802e+02, 8.08795e-01, 0.00000e+00],
    #         [4.23969e+02, 4.30401e+02, 5.16833e+02, 7.20000e+02, 7.77376e-01, 2.70000e+01],
    #         [9.81310e+02, 3.10712e+02, 1.03111e+03, 4.19273e+02, 2.86850e-01, 2.70000e+01]])

    xyxy = xyxy[xyxy[:, 4] >= confidence]  # Filter desired confidence
    xyxy = xyxy[xyxy[:, 5] == 0]  # Consider only people
    xyxy = xyxy[:, :4]
```

Figure 20: detect people on frame source code – run the yolo model

The centers of the bounding box detected are computed, both in the classic yolo view and in the bird eye view. In this way we obtain two distinct arrays: the first, *yolo_centers*, of distances calculated without considering the perspective calculation. The second, *bird_centers*, of distances calculated considering the perspective calculation (i.e. the bird eye view), as in **Figure 21**. Moreover, the respective thresholds (*one_meter_threshold_bird* and *one_meter_threshold_yolo*) are also calculated in the first frame analysis, as in **Figure 22**.

```
# Calculate the yolo_centers of the bottom of the boxes
yolo_centers = []
for x1, y1, x2, y2 in xyxy:
    center = [np.mean([x1, x2]), y2]
    yolo_centers.append(center)

# copy the sdd_frame to save the reference
yolo_frame = sdd_frame.copy()

if frame_number == 1:
    one_meter_threshold_bird = compute_bird_distance(filter_m)
    one_meter_threshold_yolo = compute_yolo_distance()

# Convert to bird so we can calculate the usual distance
bird_centers = convert_to_bird(yolo_centers, filter_m)
```

Figure 21: detect people on frame source code – computing bounding boxes centers

```
def compute_bird_distance(filter_m):

    convert_to_bird_distance = convert_to_bird(distance_pts, filter_m)
    distance_bird = compute_distance(convert_to_bird_distance[0], convert_to_bird_distance[1])
    one_meter_threshold_bird = float(distance_bird) / float(meters)

    return one_meter_threshold_bird


def compute_yolo_distance():

    yolo_distance = compute_distance(distance_pts[0], distance_pts[1])
    one_meter_threshold_yolo = float(yolo_distance) / float(meters)

    return one_meter_threshold_yolo
```

Figure 22: detect people on frame source code - computing distances thresholds

It starts the iteration over the xyxy array size for checking distances. The number of rows will correspond to the number of objects of class 0 (people) present within the scene.

The *bird_colors* and *yolo_colors* arrays are used to save the positions and color values of the bounding boxes in the yolo and ssd frames, and of the dots in the bird eye frame. In case of distance violation, the field will be *red*, otherwise it will remain *green*. **Figure 23**

```
# initialize bird_colors array
bird_colors = ['green'] * len(bird_centers)

# initialize bird_colors array
yolo_colors = ['green'] * len(yolo_centers)

for i in range(len(bird_centers)):
    for j in range(i + 1, len(bird_centers)):

        # Calculate distance of the yolo_centers
        dist = compute_distance(bird_centers[i], bird_centers[j])
        dist_1, x1_1, y1_1, x2_1, y2_1 = center_distance(xyxy[i], xyxy[j])
```

Figure 23: detect people on frame source code - initializing colors array and starting the loop

For the first if statement, a red line is drawn between the people who are violating the distance, both in the *sdd* frame and *bird eye* frame. **Figure 24**

```
if dist < one_meter_threshold_bird:
    # If dist < distance, boxes are red and a line is drawn
    bird_colors[i] = 'red'
    bird_colors[j] = 'red'

    x1, y1 = bird_centers[i]
    x2, y2 = bird_centers[j]

    # Increments the number of violations
    sdd_violations = sdd_violations + 1

    # Draws a red line in the bird_eye frame between the two persons which are violating the distance
    bird_eye_frame = cv2.line(bird_eye_frame,
                              (int(x1), int(y1)),
                              (int(x2), int(y2)),
                              (0, 0, 255), 2)

    # Draws a red line in the sdd_frame between the two persons which are violating the distance
    sdd_frame = cv2.line(sdd_frame,
                         (int(x1_1), int(y1_1)),
                         (int(x2_1), int(y2_1)),
                         (0, 0, 255), 2)
```

Figure 24: detect people on frame source code - drawing violation lines in ssd and bird eye frame

For the second if statement, a red line is drawn between the people who are violating the distance the yolo frame only. **Figure 25**

```
if dist_1 < one_meter_threshold_yolo:
    # If dist < distance, boxes are red and a line is drawn
    yolo_colors[i] = 'red'
    yolo_colors[j] = 'red'

    # Increments the number of violations
    yolo_violations = yolo_violations + 1

    # Draws a red line in the yolo_frame between the two persons which are violating the distance
    yolo_frame = cv2.line(yolo_frame,
                          (int(x1_1), int(y1_1)),
                          (int(x2_1), int(y2_1)),
                          (0, 0, 255), 2)
```

Figure 25: detect people on frame source code - drawing violation lines in yolo frame

As with the lines, green or red bounding boxes are also added on yolo and ssd frames, in accordance with the reference saved in the color arrays; and the circles in the bird eye one. **Figure 26**

```
# draw the circles in the bird_eye frame
for i, bird_center in enumerate(bird_centers):

    if bird_colors[i] == 'green':
        bird_color = (0, 255, 0)
    else:
        bird_color = (0, 0, 255)

    x, y = bird_center
    x = int(x)
    y = int(y)

    bird_eye_frame = cv2.circle(bird_eye_frame, (int(x), int(y)), 16, bird_color, -1)

# draw the rectangles in yolo and sdd frames
for i, (x1, y1, x2, y2) in enumerate(xyxy):

    if bird_colors[i] == 'green':
        bird_color = (0, 255, 0)
    else:
        bird_color = (0, 0, 255)

    if yolo_colors[i] == 'green':
        yolo_color = (0, 255, 0)
    else:
        yolo_color = (0, 0, 255)

    sdd_frame = cv2.rectangle(sdd_frame, (int(x1), int(y1)), (int(x2), int(y2)), bird_color, 2)
    yolo_frame = cv2.rectangle(yolo_frame, (int(x1), int(y1)), (int(x2), int(y2)), yolo_color, 2)
```

Figure 26: detect people on frame source code - drawing rectangles and circles

Finally, the 3 frames are horizontally concatenated and titles, counters of people and violations are added using the *add_text* function. **Figure 27**

```
# Concat the yolo, bird-eye and ssd frames into one
new_bird_eye_frame = cv2.resize(bird_eye_frame, (width, height))
yolo_frame = cv2.hconcat([yolo_frame, sdd_frame])
new_bird_eye_frame = cv2.hconcat(([yolo_frame, new_bird_eye_frame]))

# add border for titles and description
color = (0, 0, 0)
bottom, up = [50] * 2
new_bird_eye_frame = cv2.copyMakeBorder(new_bird_eye_frame, bottom, up, 0, 0, cv2.BORDER_CONSTANT, value=color)

# display titles and counter
add_text(new_bird_eye_frame, height, width, shape, sdd_violations, yolo_violations)

return yolo_centers, bird_centers, new_bird_eye_frame, one_meter_threshold_bird, one_meter_threshold_yolo
```

Figure 27: detect people on frame source code – adding titles and counters to the concatenated frame

5. Results

Test is the process of evaluating and verifying that an application or software product is working properly against expectations. To make a good evaluation of the program has been used different videos from different online dataset or taken from youtube. Ten screen that summarize the results of the test are showed. **Figure 28-35**

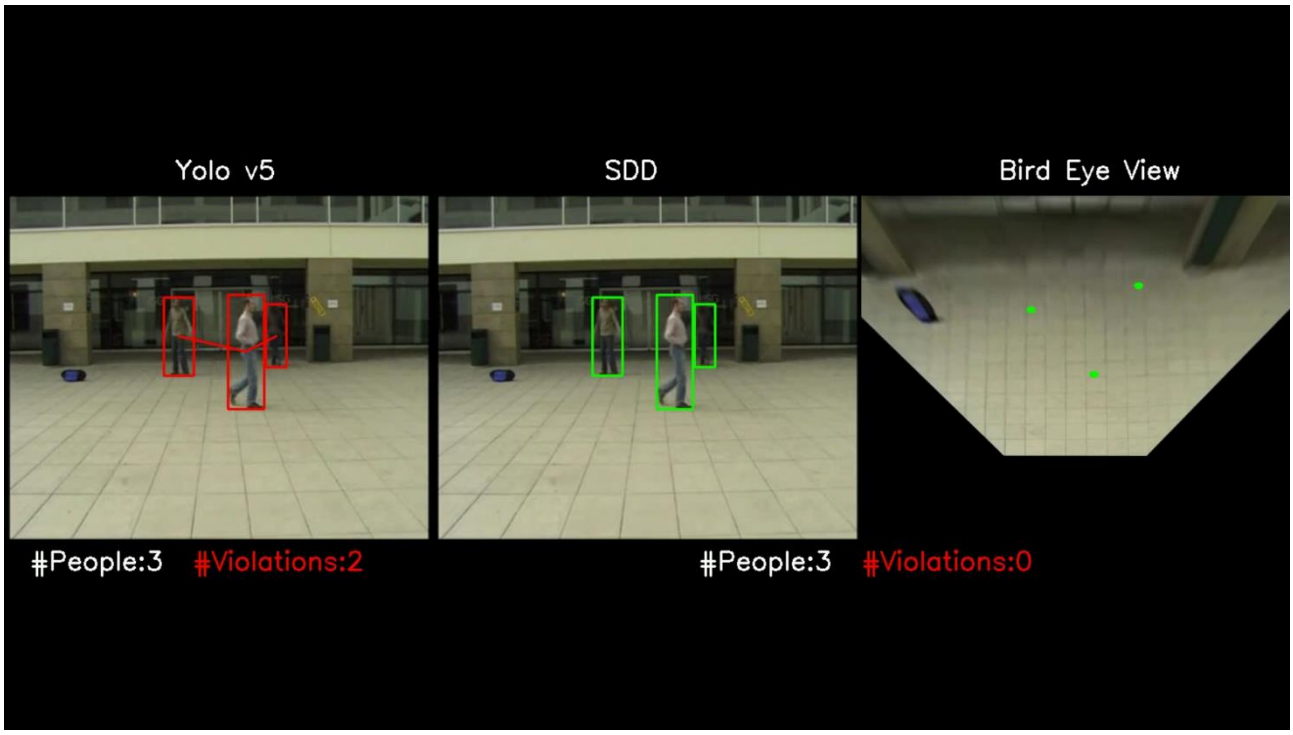


Figure 28: screen result experiment1.avi

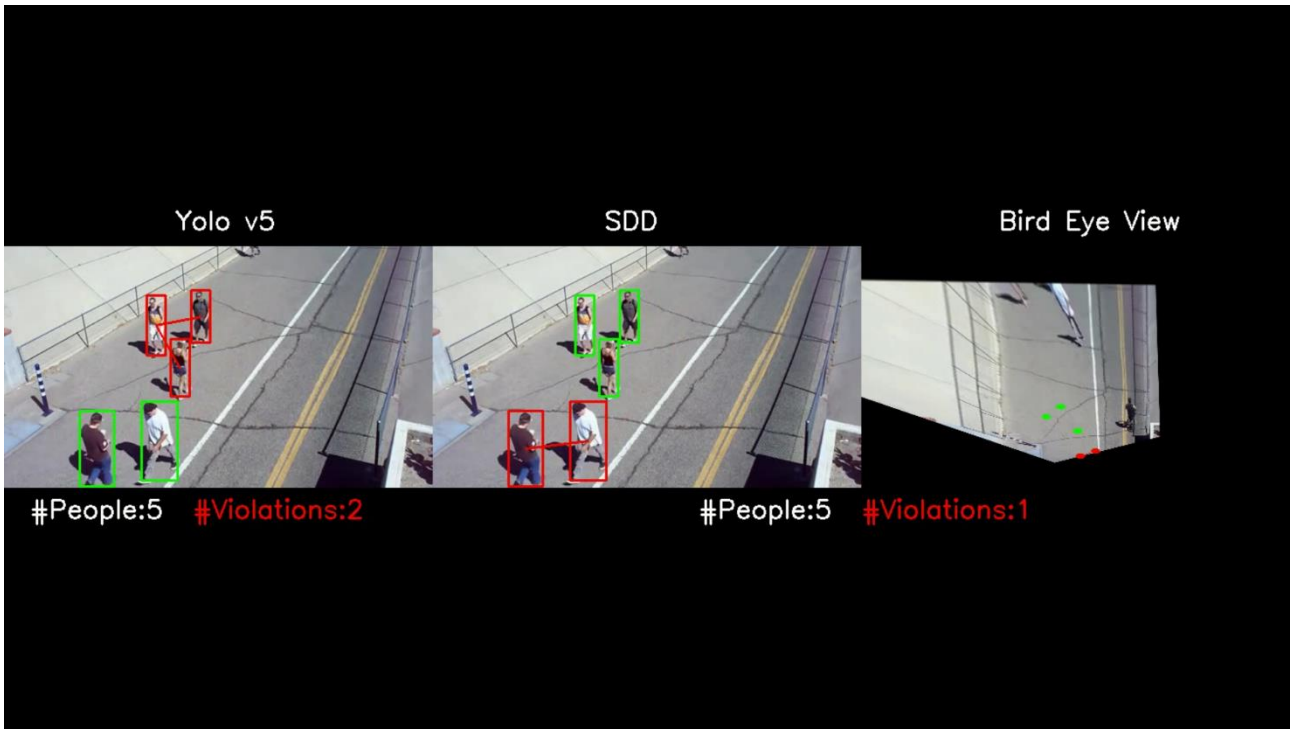


Figure 29: screen result experiment2.avi

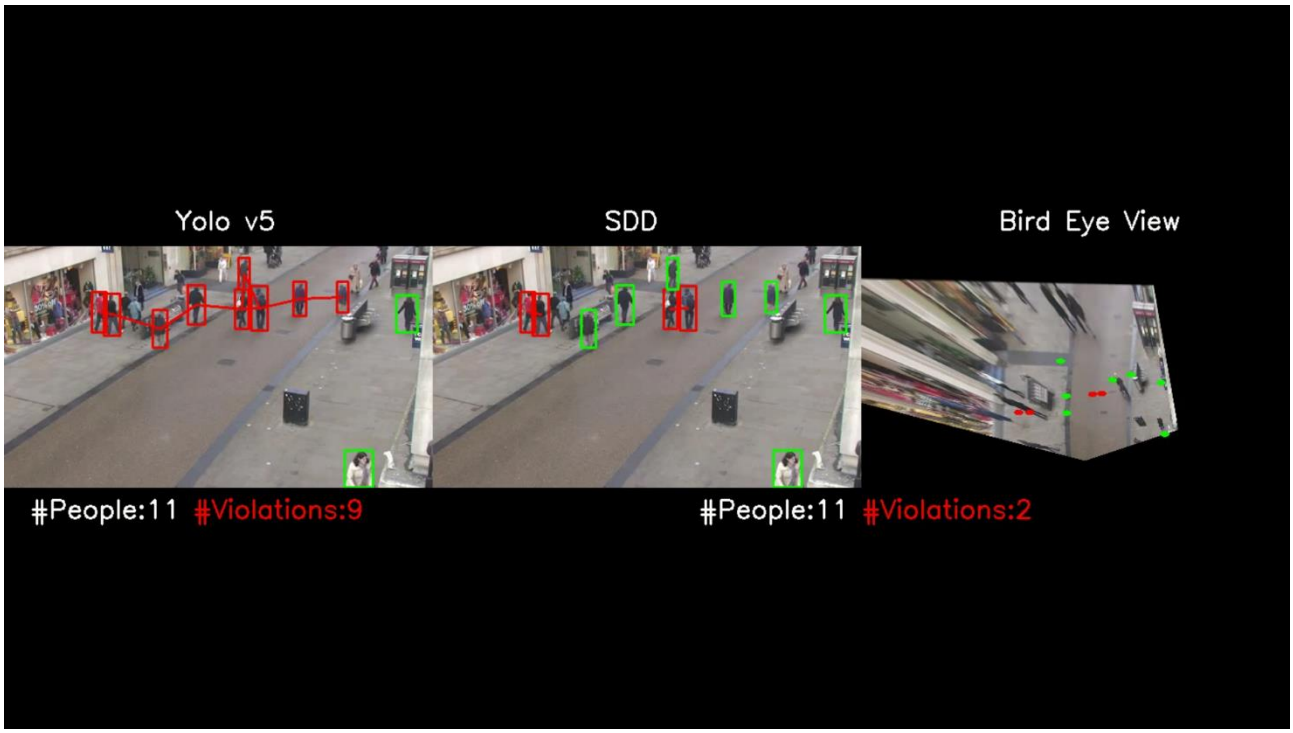


Figure 30: screen result experiment3.avi

Figure 31: screen result experiment4.avi

Figure 32: screen result experiment5.avi

Figure 33: screen result experiment6.avi

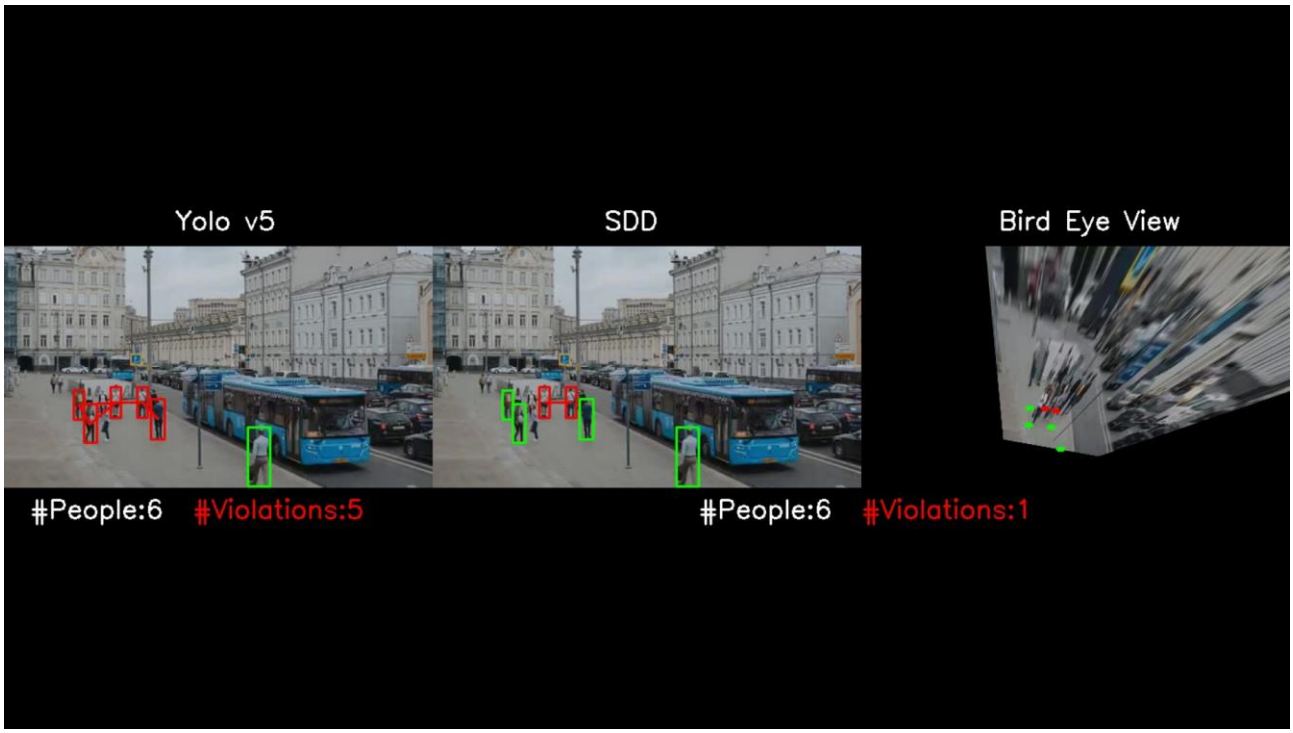


Figure 34: screen result experiment7.avi

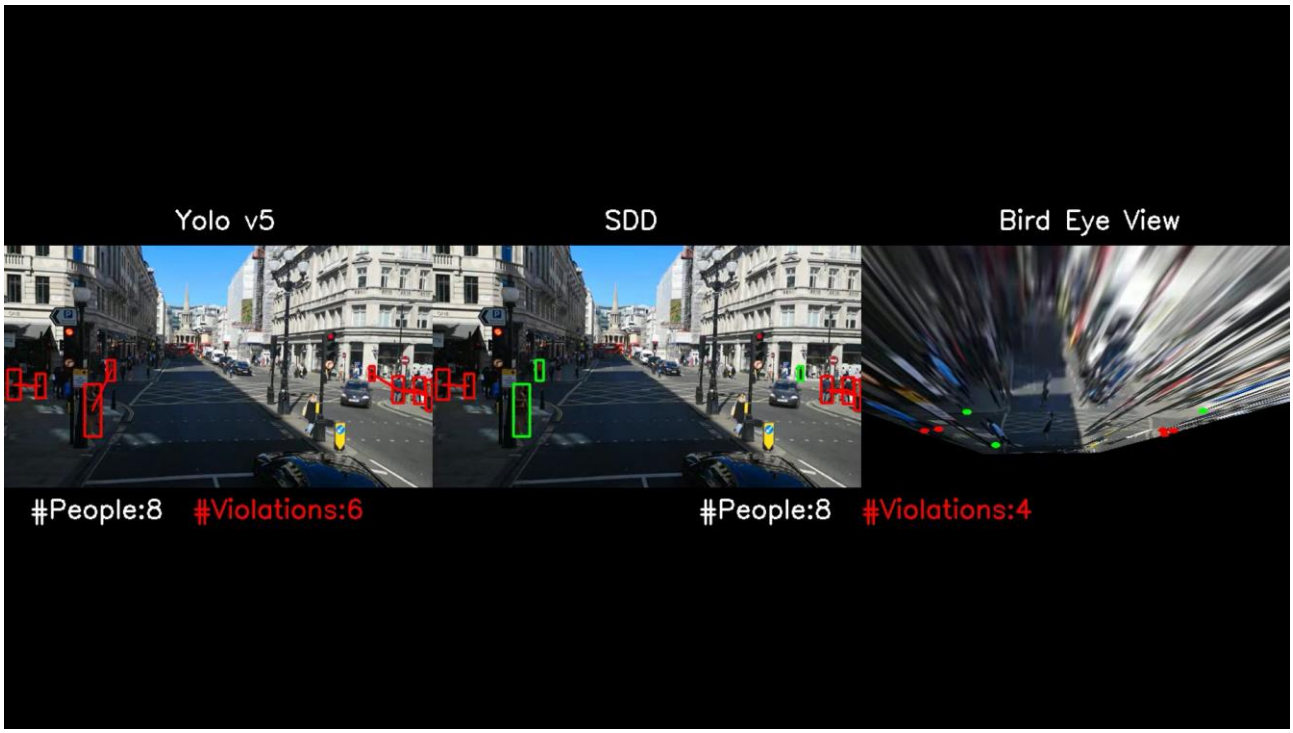


Figure 35: screen result experiment8.avi

After testing, the team is convinced that the designed model achieves very good performance on those cameras positioned in a way that defines a top-down or frontal perspective. The worst results of using this model are when the camera elements are too far away, or the camera is positioned too high.

The tests make it clear how drawing the right polygon and the accuracy with which distances are represented will or will not favor the success of the experiment. These constraints are essential to ensure that the model can be used on any type of camera or framing.

6. Conclusion and future works

With the spread of Covid-19, social distancing has become very important in preventing the spread of the virus and staying healthy. The model used in this project follow the results seen in the state of the art.

The system developed can be improved in accuracy and results with many approaches. For example, YOLOv5 is currently performing the pretrained model, but if we tune it to less object class detection and confine the CNN layer based on the video quality, we may save more inference time and obtain higher in-time processing speed.

The computation of the distance depends on how the bird-eye is computed and if the ROI is drawn correctly. In the future, it could be used a neural net to detect ground in images and transform to bird-eye view automatically.

The team were not able to remove any distortion in the analyzed frames, so a future improvement could be adding a proper camera calibration to address this challenge.

The results show a reliable and truthful real-time calculation, useful if the model is installed on a camera set in crowded closed places.

7. References

[1] J. C. S. Jacques, S. R. Musseff, and C. R. Jung, “Crowd analysis using computer vision techniques,” *IEEE Signal Processing Magazine*, vol. 27, no. 5, pp. 66–77, 2010, doi: 10.1109/MSP.2010.937394.

[2]] Shivashree and] Dr Anuradha, “Crowd Analysis Using Computer Vision Techniques,” 2018.

[3] A. Fitwi, Y. Chen, H. Sun, and R. Harrod, “Estimating interpersonal distance and crowd density with a single-edge camera,” *Computers*, vol. 10, no. 11, Nov. 2021, doi: 10.3390/computers10110143.

[4] A. Antonucci, V. Magnago, L. Palopoli, and D. Fontanelli, *Performance Assessment of a People Tracker for Social Robots*. 2019. doi: 10.1109/I2MTC.2019.8826999.

[5] M. Al-Sa’d, S. Kiranyaz, I. Ahmad, C. Sundell, M. Vakkuri, and M. Gabbouj, “A Social Distance Estimation and Crowd Monitoring System for Surveillance Cameras,” *Sensors*, vol. 22, no. 2, p. 418, Jan. 2022, doi: 10.3390/s22020418.

[6] F. Hikamudin Arby and H. al Amin, “Implementation of YOLO-v5 for a real-time Social Distancing Detection,” 2022. [Online]. Available: http://jurnal.polibatam.ac.id/index.php/JAIC

[7] M. Vajgl, P. Hurtik, and T. Nejezchleba, “Dist-YOLO: Fast Object Detection with Distance Estimation,” *Applied Sciences*, vol. 12, no. 3, p. 1354, Jan. 2022, doi: 10.3390/app12031354.

[8] A. Rahim, A. Maqbool, and T. Rana, “Monitoring social distancing under various low light conditions with deep learning and a single motionless time of flight camera,” *PLoS ONE*, vol. 16, no. 2 February, Feb. 2021, doi: 10.1371/journal.pone.0247440.

[9] EPFL, “Multi-camera pedestrians video.” https://www.epfl.ch/labs/cvlab/data/data-pom-index-php/ (accessed Apr. 22, 2022).

[10] J.Hui, “object detection accurancy”, Accessed: Apr. 22, 2022. [Online]. Available: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[11] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Dec. 2015, [Online]. Available: http://arxiv.org/abs/1512.03385

[12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.10934

[13] Wikipedia, “Region of interest (ROI).”

[14] “Coronavirus Disease (Covid-19),” *World Health Organization*. https://www.who.int/health-topics/coronavirus#tab=tab_1 (accessed Apr. 22, 2022).

[15] Geogebra, “Geogebra.” https://www.geogebra.org/ (accessed Apr. 22, 2022).